

1

Testiranje, primjena i održavanje sustava

2018/19.14a

Provjera ispravnosti

2

- Testiranje programa, provjeravanje programa, ispitivanje programa
 - provjera programa izvođenjem, uz uporabu ispitnih podataka te analizom rezultata obrade
 - cilj testiranja je otkrivanje pogrešaka odnosno nedostataka unutar programa
 - uspješnost testa razmjerna je broju pronađenih pogrešaka
- Stupnjevi, stadiji, faze
 - testiranje jedinica, integracijsko testiranje, test sustava i test prihvatljivosti
- Verifikacija - provjera ispravnosti
 - „*Are we building the product right?*”
 - dokazivanje da je faza dobro provedena ili da je proizvod dobro napravljen, tj. da odgovara specifikaciji zahtjeva (slučajevima korištenja)
- Validacija - potvrda valjanosti
 - „*Are we building the right product?*”
 - kojom se utvrđuje da je napravljen pravi proizvod, koji odgovara namjeni te da je prihvatljiv korisniku

- Test – provjerava je li neki aspekt softvera ispravan
 - pr. test da radi login, test da utrošak memorije ne premašuje 500Mb
- Pogreška (*error*) – propust programera, npr. radi nerazumijevanja
 - dovodi do jednog ili više kvarova
 - razlikujemo u odnosu na "pogrešku" koja znači neželjeno stanje, tj. kvar
- Kvar (*fault*), defekt (*defect*), neformalno *bug* - neispravan dio koda
 - npr. pogrešna pretpostavka da se polje indeksira od 1 umjesto 0 izaziva kvar pristupa elementu polja
- Zastoj u radu (*failure*) – stanje izazvano jednim ili više kvarova
 - npr. prestanak rada sustava zbog "buffer overrun" kvara
- Ispravak (*Fix*) – stanje popravka

Problem testiranja objektno orijentiranih sustava

4

➡ Problem: kako provjeriti da sustav zadovoljava potrebe korisnika

➡ rješenje: testiranjem poslovnih procesa

- ➡ poslovni proces je raspodijeljen u skupu međudjelujućih razreda i sadržan u postupcima tih razreda
- ➡ jedini način da se sazna učinak poslovnog procesa na sustav je zagledavanje u promjene stanja u sustavu
- ➡ ućahurivanje međutim sakriva podatke i obradu iza izloženih sučelja !

➡ Drugi problem: što je osnovna "jedinica" za testiranje

- ➡ paket, razred ili metoda ?
- ➡ u tradicionalnim pristupima proces je sadržan u funkciji
- ➡ u OO sustavima zasebno testiranje pojedinačnih metoda nema puno smisla
 - ➡ uslijed nasljeđivanja i višeobličja postoje različita ponašanja, a bugovi nadređenog razreda propagiraju u neposredno i posredno izvedene
 - ➡ pri dinamičkom povezivanju ne zna se unaprijed koja će implementacija razreda biti izvršena

Plan testiranja

5

- Provjera počinje planom testiranja - plan definira niz testova
 - plan treba napraviti na početku razvoja i stalno ažurirati
- Primjer: plan jedinične provjere razreda

Class Test Plan	Page ____ of ____	
Class Name: _____	Version Number: _____	CRC Card ID: _____
Tester: _____	Date Designed : _____	Date Conducted : _____
Class Objective:		
Associated Contract IDs:		
Associated Use Case IDs:		
Associated Superclass(es):		
Testing Objectives:		
Walkthrough Test Requirements:		
Invariant-Based Test Requirements:		
State-Based Test Requirements:		
Contract-Based Test Requirements:		

Plan testiranja (2)

6

► Primjer: plan provjere invarijanti razreda

- navodi se izvorna i nova vrijednost atributa, događaj koji izaziva promjenu te rezultat tj. posljedica promjene. Evidentira se uspješnost testa (*Pass/Fail*)

Class Invariant Test Specification						Page ____ of ____
Class Name: _____		Version Number: _____		CRC Card ID: _____		
Tester: _____		Date Designed : _____		Date Conducted : _____		
Testing Objectives:						
Test Cases						
Invariant Description	Original Attribute Value	Event	New Attribute Value	Expected Result	Result P/F	
Attribute Name:						
1) _____	_____	_____	_____	_____	_____	
2) _____	_____	_____	_____	_____	_____	
3) _____	_____	_____	_____	_____	_____	
Attribute Name:						
1) _____	_____	_____	_____	_____	_____	

Testiranje jedinica

7

➤ Testiranje jedinica (unit testing), pojedinačno testiranje

➤ **najmanja jedinica mjere je razred !**

➤ testovi primjenjivi na nadređeni razred primjenjivi su na iz njega izvedene razrede, u dijelovima koji nisu preopterećeni nasljeđivanjem

➤ posebnosti (višeobličje) – provjeriti u zasebnom kontekstu

Testiranje jedinica – vrste testiranja

8

➤ Funkcionalno (black-box testing)

- provjera se što cjelina radi, to jest da li zadovoljava zahtjeve
- probni slučajevi izvode se iz specifikacija
- provodi osoblje proizvođača ili korisnici
- osnovica plana testiranja: CRC kartice, dijagrami razreda, ugovori

➤ Strukturalno (white-box, clear box testing)

- provjera kako cjelina radi
- probni slučajevi izvode se uvidom u programski kôd (inspekcija koda)
- provode programeri
- osnovica plana testiranja: specifikacije metoda

➤ Integracijska provjera (integration testing)

- jedinica je komponenta !
- razine: razredi koji tvore logičku cjelinu, sloj, paket, knjižnica
- ispitivanje provodi tim (analitičara i programera) za testiranje

Integracijsko testiranje – vrste testiranja (1)

10

➤ Testiranje korisničkog sučelja (User Interface Testing)

- provjerava se svaka funkcija sučelja
- osnovica: dizajn sučelja
- testiranje se radi prolaskom kroz svaku stavku izbornika sučelja

➤ Testiranje slučajeva korištenja (Use-Case Testing)

- provjerava se svaki slučaj korištenja
- osnovica: slučajevi korištenja
- testiranje se radi prolaskom kroz svaki slučaj korištenja
- često se kombinira s testom korisničkog sučelja jer UC ne testira sva sučelja

Integracijsko testiranje – vrste testiranja (2)

11

- Testiranje interakcije (Interaction Testing)
 - testiranje svakog procesa korak po korak
 - osnovica: dijagrami razreda, dijagrami slijeda, dijagrami komunikacije
 - cijeli sustav započinje kao skup okrajaka
 - razredi se dodaju pojedinačno i rezultati se uspoređuju s očekivanim
 - nakon što neki razred prođe testove, dodaje se sljedeći i ponavljaju testovi
 - postupak se provodi za svaki paket
 - kad svi paketi prođu sve testove, proces se ponavlja za integriranje paketa
- Testiranje sučelja sustava (System Interface Testing)
 - testiranje razmjene podataka s drugim sustavima
 - osnovica: dijagrami slučajeva korištenja
 - prijenosi podataka između sustava često automatizirani
 - korisnici ih izravno ne nadziru – dodatna pažnja na provjeru / ispravnost

➡ Provjera sustava (System Testing)

- ➡ provjera rada sustava kao cjeline, kojom se osigurava da svi nezavisno razvijeni aplikacijski programi rade ispravno te sukladno specifikacijama

Testiranje sustava – vrste testiranja (1)

13

➤ Testiranje zahtjeva (Requirements Testing)

- testiranje jesu li zadovoljeni izvorni poslovni zahtjevi
- osnovica: dizajn sustava, testovi komponenti i integracijski testovi
- osigurava da promjene tijekom integracije nisu dovele do novih pogrešaka
- tester se pretvara da su nekompetentni korisnici i izvode neprikladne radnje da testiraju robustnost (npr. dodavanje praznih ili duplih zapisa)

➤ Testiranje uporabivosti (Usability Testing)

- testiranje prikladnosti sustava za korištenje
- osnovica: dizajn sučelja i slučajevi korištenja
- analitičar koji razumije korisnika i poznaje (dobar) dizajn sučelja

Testiranje sustava – vrste testiranja (2)

14

➤ Testiranje sigurnosti (Security Testing)

- testiranje mogućnosti oporavka i neautoriziranog pristupa
- osnovica: dizajn infrastrukture
- analitičar infrastrukture, u ekstremnim slučajevima zasebna tvrtka

➤ Testiranje performansi (Performance Testing)

- ispitivanje sposobnosti izvođenja pod velikim opterećenjem
 - stress testing - velik broj interakcija (simulacijom pristupa)
 - load testing – velika količina podataka (npr. generatorima podataka)
- osnovica: prijedlog sustava, dizajn infrastrukture

➤ Testiranje dokumentacije (Documentation Testing)

- testiranje ispravnosti dokumentacije
- osnovica: sustav pomoći, postupci, priručnici

➤ Provjera prihvatljivosti (Acceptance Testing)

- dokazivanje da proizvod zadovoljava zahtjeve i uvjete preuzimanja
- iscrpan i konačan test nad stvarnim podacima

➤ Alfa-testiranje (Alpha Testing) - verifikacijsko

- probna uporaba koju provode korisnici kod izvođača
- simulacija stvarnog okruženja
- traženje pogrešaka i propusta

➤ Beta-testiranje (Beta Testing) – validacijsko

- provode korisnici kod sebe, bez nazočnosti izvođača
- provjera u stvarnim uvjetima
 - performanse sustava
 - vršna opterećenja
 - provjera upotrebljivosti i lakoće uporabe
 - radne procedure
 - izrada rezervnih kopija i oporavak sustava

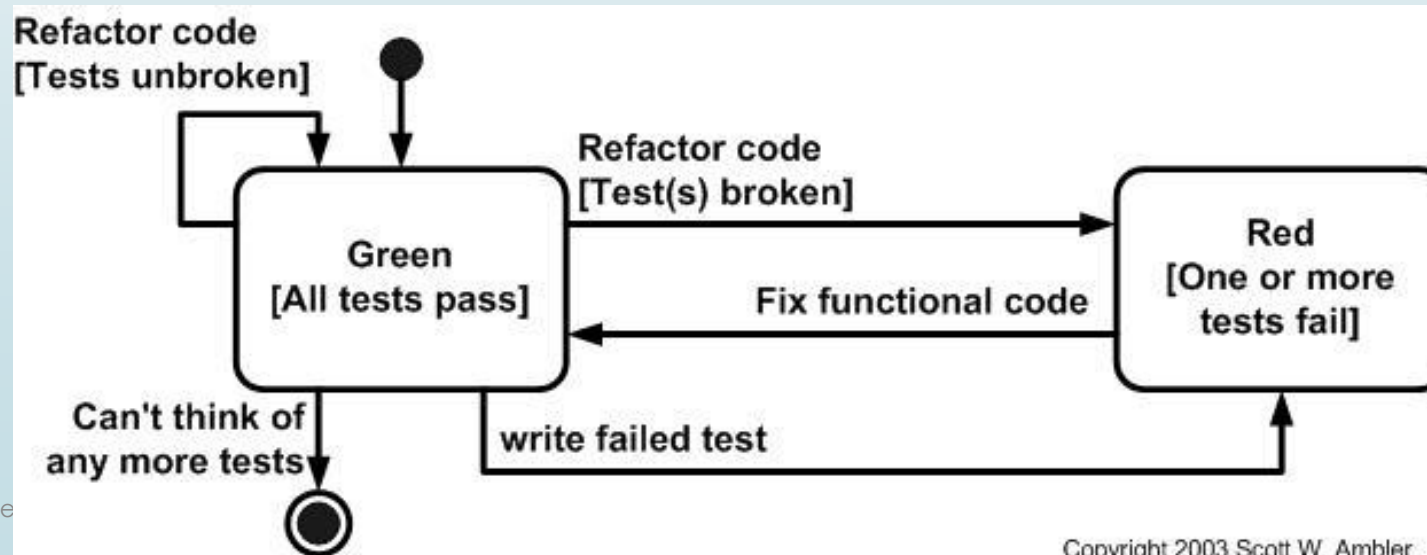
➤ Nadzorni test (Audit Test) – provodi se opcionalno

- potvrda da je sustav gotov, ispravan i spreman za primjenu
- provode nezavisne tvrtke ili odjeli za osiguranje kvalitete

Razvoj vođen testiranjem

16

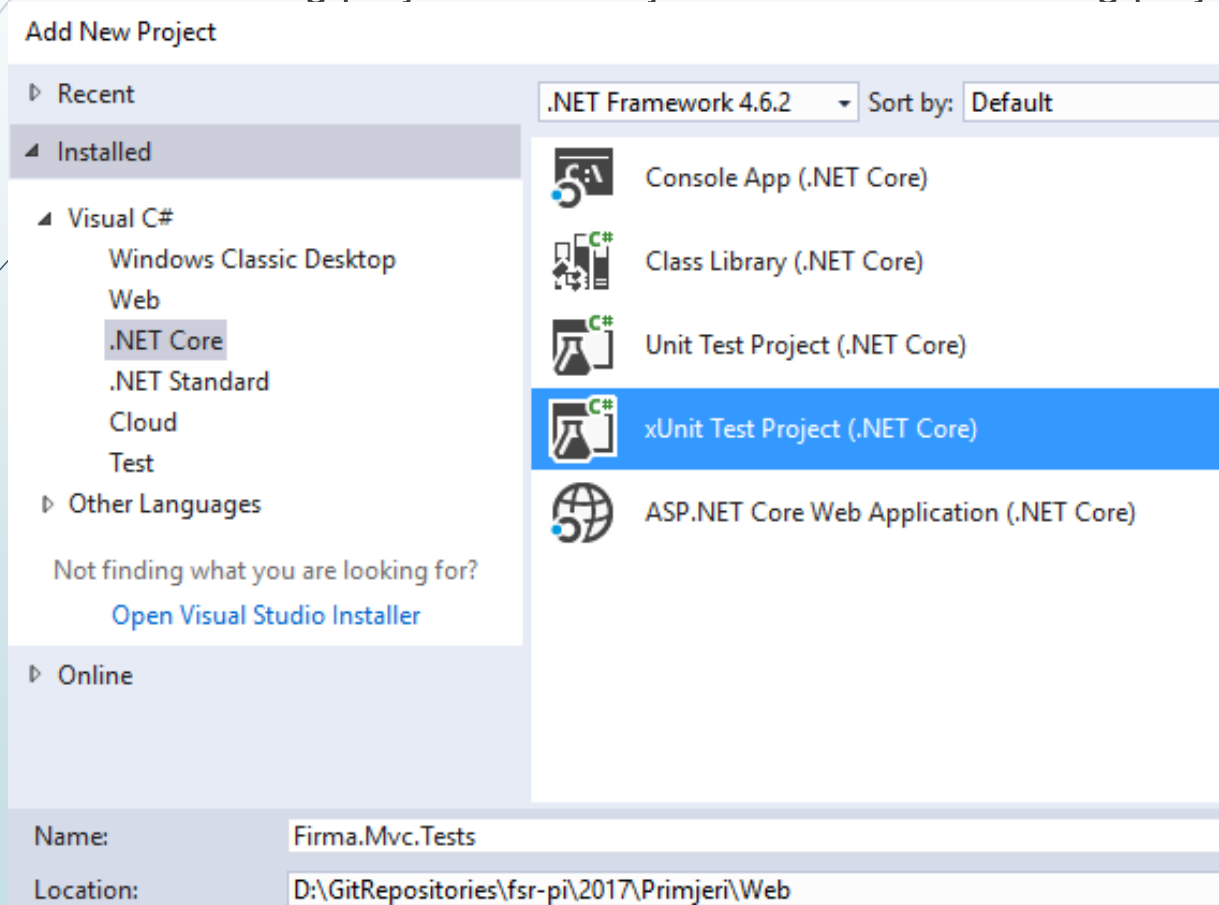
- Razvoj vođen testiranjem (Test Driven Development)
 - Testovi se pišu prije koda, tj. ne kodira se nešto za što ne postoji test
 - U svakoj iteraciji po obavljanju testa provodi se refaktoriranje
- Automatizacija testiranja
 - alati za automatsko testiranje - rezultate testa uspoređuju s očekivanim rezultatima
 - *Unit (JUnit, NUnit, CSUnit, XUnit, CppUnit, PyUnit, PHPUnit, ...) – open source
- Regresijsko testiranje (regression testing), retesting
 - provjera kojom se dokazuje da softver nije nazadovao (engl. *regressed*)
 - pokretanje svih testova (starih i nanovo dodanih) pri promjeni softvera



Primjeri jediničnog testiranja

17


- Stvara se novi testni projekt paralelno s projektom koji se želi testirati
- Koristi se alat xUnit
- Naziv testnog projekta obično jednak nazivu testiranog projekta uz sufiks .Tests



Primjer:  Web \ Firma.Mvc.Tests

Primjer testiranja – test razreda za filtriranje podataka (1)

18

- Skupom testnih postupaka testira se ispravnost pretvorbe aktivnog filtra u string i obrnuto
- Testni postupak označen atributom Fact, a atributom Trait svrstava se u neku kategoriju
- Test je ispravan ako su sve tvrdnje zadovoljene
 - Različiti postupci iz razreda Assert: *IsTrue*, *Equal*, *Contains*, *IsType*, ...
 - Primjer:  Web \ Firma.Mvc.Tests \ DocumentFilterTests

```
public class DocumentFilterTests {  
    [Fact]  
    [Trait("Category", "DocumentFilter")]  
    public void PrazniFilter() {  
        string filterString = "-----";  
        DokumentFilter filter = DokumentFilter.FromString(filterString);  
        Assert.True(filter.IsEmpty());  
    }  
}
```

Primjer testiranja – test razreda za filtriranje podataka (2)

19

► Primjer:  Web \ Firma.Mvc.Tests \ DocumentFilterTests

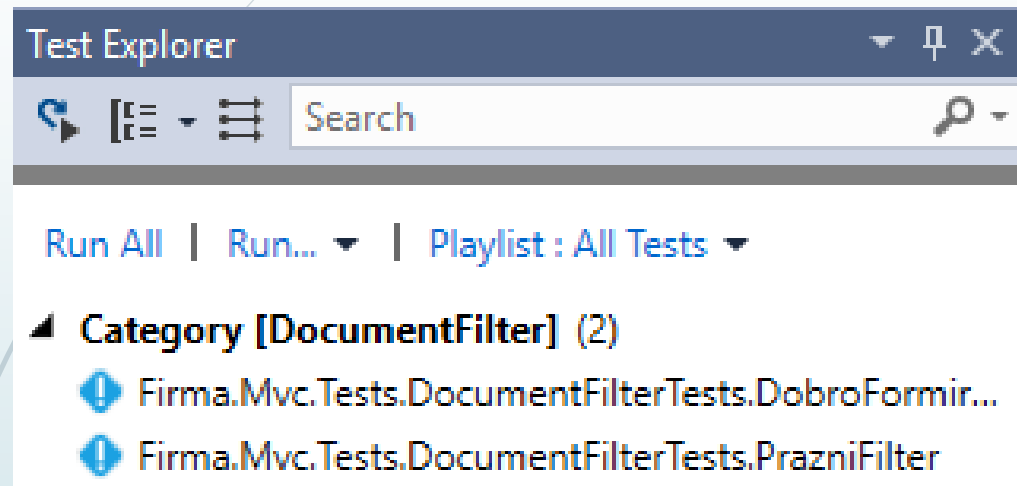
```
public class DocumentFilterTests {  
    ...  
    [Fact]  
    [Trait("Category", "DocumentFilter")]  
    public void DobroFormiranString() {  
        DokumentFilter filter = new DokumentFilter();  
        filter.IdPartnera = 1;  
        filter.IznosOd = 300;  
        filter.IznosDo = 500;  
        filter.NazPartnera = "Nebitno";  
        filter.DatumOd = new DateTime(1911, 2, 13, 9, 15, 0);  
        filter.DatumDo = new DateTime(2011, 2, 13, 20, 15, 0);  
        string filterString = filter.ToString();  
        string expected = "1-13.2.1911-13.2.2011-300-500"; //neispravno!  
        Assert.Equal(expected, filterString);  
    }  
}
```

Izvođenje testa

20

➡ Visual Studio nakon uspješne kompilacije prepozna je testove unutar projekta

➡ View → Test Explorer



➡ Što je neispravno?

➡ Krivo napisan test?

➡ Neispravan stvarni kôd?

Category [DocumentFilter] (2)

- ✗ Firma.Mvc.Tests.DocumentFilterTests.Dobr... 160 ms
- ✓ Firma.Mvc.Tests.DocumentFilterTests.Prazni... 83 ms

Firma.Mvc.Tests.DocumentFilterTests.DobroForm

Source: DocumentFilterTests.cs line 21

✗ Test Failed - Firma.Mvc.Tests.DocumentFilterTests.Dc

Message: Assert.Equal() Failure

↓ (pos 5)


Expected: 1-13.2.1911-13.2.2011-300-500

Actual: 1-13.02.1911-13.02.2011-300-500

↑ (pos 5)

Primjer testiranja nad stvarnim ili nalik stvarnom okruženju

21

- Preporuča se koristiti testnu bazu podataka
- Testira se rezultat upravljača koji služi za nadopunjavanje
 - appsettings.json kopirati iz pravog projekta i odabrati opciju CopyAlways
 - U konstruktoru testa postavlja se kontekst i konfiguracijske postavke slično kao u Startup.cs
 - Koncept *Arrange – Act – Assert*
- Primjer:  Web \ Firma.Mvc.Tests \ AutoCompleteMjestoTests

```
public class AutoCompleteMjestoTests {  
    FirmaContext ctx; IOptions<AppSettings> options;  
    public AutoCompleteMjestoTests() {  
        var builder = new ConfigurationBuilder()  
            .AddUserSecrets("Firma")  
            .AddJsonFile("appsettings.json");  
        var Configuration = builder.Build();  
        ...  
        ctx = new FirmaContext(dbContextBuilder.Options);  
    }  
}
```

Ponavljanje istog testa s različitim podacima

22

➡ Umjesto pisanja više istih testova, koriste se atributi *Theory* i *InlineData*


➡ Podatak iz *InlineData* se koristi kao argument testnog postupka

➡ Primjer:  Web \ Firma.Mvc.Tests \ AutoCompleteMjestoTests

```
[Theory]
[InlineData("varaždin")]
[InlineData("VARAŽDIN")]
[InlineData("ara")]
public void PronadjenNazivMjesta(string value) {
    var controller = new AutoComplete.MjestoController(ctx, options);
    IEnumerable<IdLabel> result = controller.Get(value);
    string naziv = "Varaždin";
    var containsValue = result.Select(idlabel => idlabel.Label)
                             .Any(label => label.IndexOf(naziv,
                             StringComparison.CurrentCultureIgnoreCase) != -1);
    Assert.True(containsValue);
    string nazivKojegNema = "Split"; containsValue = ...
    Assert.False(containsValue);
```

Primjer testiranja – testiranje upravljača (1)

23

- Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti rezultata na akciju *Create* ako u bazi podataka nema podataka
 - FirmaContext se stvara s privremenom bazom podataka smještenom u memoriji
 - Ne postavlja se u konstruktoru, jer za svaki test treba posebni primjerak
- Korištenjem razreda *Mock* moguće stvoriti lažni objekt i definirati vlastito ponašanje pojedinih postupaka
- Primjer:  Web \ Firma.Mvc.Tests \ DrzavaControllerTests

```
public class DocumentFilterTests {  
    [Fact]  
    public void TestNemaDrzava() {  
        var mockLogger = new Mock<ILogger<DrzavaController>>();  
  
        var dbOptions = new DbContextOptionsBuilder<FirmaContext>()  
            .UseInMemoryDatabase(databaseName: "FirmaMemory1")  
            .Options;
```

Primjer testiranja – testiranje upravljača (2)

24

➤ Upravljač *Drzava* pri izvođenju akcije *Index* treba preusmjeriti rezultata na akciju *Create* ako u bazi podataka nema podataka

➤ Provjera tipa rezultata

➤ Primjer:  Web \ Firma.Mvc.Tests \ DrzavaControllerTests

```
public class DrzavaControllerTests {  
    [Fact]  
    public void TestNemaDrzava() {  
        ...  
        var controller = new DrzavaController(context, options,  
                                                mockLogger.Object);  
        var tempDataMock = new Mock<ITempDataDictionary>();  
        controller.TempData = tempDataMock.Object;  
        var result = controller.Index();  
  
        var redirectToActionResult =  
Assert.IsType<RedirectToActionResult>(result);  
Assert.Equal("Create", redirectToActionResult.ActionName);  
    }  
}
```


Primjer integracijskog testiranja (1)

25

- U prethodnim primjerima testiran je web-api upravljač za dohvat mjesta.
 - Test se mogao izvršiti na privremenoj bazi u memoriji ili na stvarnoj bazi
- Integracijskim testiranjem se provjerava radi li upravljač kad se integrira u web-server
 - Odaziva li se na pravoj adresi?
 - Prima li ispravno postavke za spajanje na bazu podataka?
 - Vraća li podatke u ispravnom formatu
- U testu se simulira web server korištenjem NuGet paketa *Microsoft.AspNetCore.TestHost* (koristi konfiguracijski razred iz testiranog projekta)
 - Primjer: `Web \ Firma.Mvc.IntegrationTests \ AutoCompleteMjesto`

```
namespace Firma.Mvc.IntegrationTest {  
    public class AutoCompleteMjesto {  
        public AutoCompleteMjesto() {  
            server = new TestServer(new WebHostBuilder().UseStartup<Startup>());  
            client = server.CreateClient();  
        }  
    }  
}
```

Primjer integracijskog testiranja (2)

26


➡ Test simulira otvaranje stranice na određenoj adresi i provjerava rezultat

➡ Primjer:  Web \ Firma.Mvc.IntegrationTests \ AutoCompleteMjesto

```
public async Task VracaViseGradova(string naziv) {  
    // Act  
    var response = await client.GetAsync(addressPrefix + naziv);  
    response.EnsureSuccessStatusCode();  
  
    var stream = await response.Content.ReadAsStreamAsync();  
    var serializer = new DataContractJsonSerializer(typeof(IEnumerable<IdLabel>));  
    var gradovi = serializer.ReadObject(stream) as IEnumerable<IdLabel>;  
    Assert.NotEmpty(gradovi);  
    Assert.True(gradovi.Count() > 1);  
    foreach (var grad in gradovi)  
    {  
        Assert.Contains(naziv, grad.Label, StringComparison.CurrentCultureIgnoreCase);  
        Assert.NotEqual(default(int), grad.Id);  
    }  
}
```


Primjer integracijskog testiranja (3)

27

- ➡ Integracijsko testiranje otkrilo je da Json koji nastane iz popisa gradova ima mala slova za id i label te klijent koji bi vršio deserijalizaciju dobije prazne podatke.
- ➡ Rješenje: Eksplicitno navesti nazive ključeva prilikom (de)serijalizacije
 - ➡ Primjer:  Web \ Firma.Mvc \ Controllers \ AutoComplete \ IdLabel.cs

```
namespace Firma.Mvc.Controllers.AutoComplete
{
    [DataContract]
    public class IdLabel
    {
        [DataMember(Name = "label")]
        public string Label { get; set; }
        [DataMember(Name = "id")]
        public int Id { get; set; }
        ...
    }
}
```

➤ Fluent Assertions <https://fluentassertions.com/>

- NuGet paket FluentAssertions
- Omogućava pisanje provjera koje izgledaju prirodnije ljudskom jeziku
 - Should().NotBe(), Should().BeGreaterThan(value, "because..."), Should().BeNull(); ...
- Primjer:  Web \ Firma.Mvc.Tests \ AutoCompleteMjestoTest

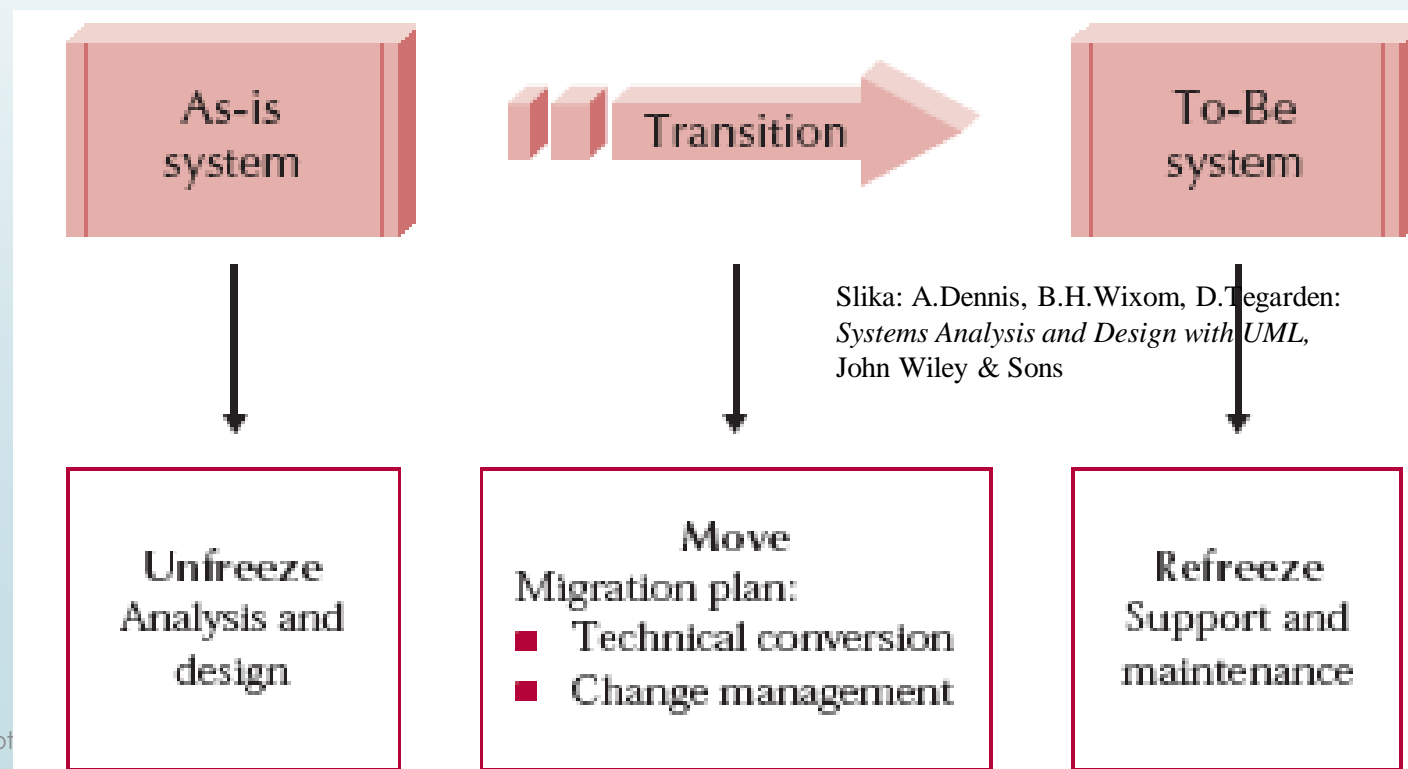
```
[Theory]
[InlineData("a", 20)]
[InlineData("šđžć", 0)]
[InlineData("a", 50)]
public void VracaOgranicenSkupRezultata(string value,int count) {
    options.Value.AutoCompleteCount = count;
    var controller = new AutoComplete.MjestoController(ctx, options);
    IEnumerable<IdLabel> result = controller.Get(value);
    result.Should().HaveCount(count,
        $"because there should at least {count}
        places containing text {value}");
```

Primjena

Uvođenje u primjenu

30

- Uvođenje u primjenu podrazumijeva promjenu u sustavu
- Upravljanje promjenama smatra se najtežim zadatkom te uključuje
 - Odmrzavanje (Unfreezing) – napuštanje starih navika i normi
 - Prijelaz (Moving) – prijelaz sa starog na novi sustav
 - Zamrzavanje (Refreezing) – usvajanje i uhodavanje novog načina rada
- Plan migracije uključuje tehničke ali i organizacijske aspekte



- Konverzija sustava
 - tehnički proces u kojem novi sustav zamjenjuje stari sustav
- Glavni koraci – najčešće slijedni na pojedinoj lokaciji
 - instalacija hardvera
 - instalacija softvera
 - konverzija podataka – najsloženija uslijed promjene strukture podataka
 - inicijalni unos podataka, npr. novih šifrnika
 - prijenos postojećih podataka uz konverziju, najčešće matičnih podataka
 - prijenos zbirnih stanja poslovna transakcija, za tzv. "prometne" podatke
 - odgovarajući plan testiranja
- Konverzija se provodi u 3 "dimenzije"
 - stil konverzije (conversion style) – način uvođenja
 - lokacija konverzije (conversion location) - gdje
 - moduli konverzije (conversion modules) – što, koji dijelovi

Način (stil) konverzije

32

➤ Izravno uvođenje (direct conversion, cold turkey, big bang)

- početak rada novog sustava uz istovremeni prestanak rada starog sustava
- u poslovnim sustavima provodi se na određeni dan, uobičajeno datum završetka poslovnog razdoblja, po mogućnosti na kraju tjedna
- mogući problemi: pojava pogrešaka koje nisu bile uočene tijekom testiranja, nepredviđeno preopterećenje opreme u punom pogonu
- nedostatak: neposredna izloženost korisnika pogreškama sustava

➤ Paralelno uvođenje (parallel conversion)

- istovremeni rad starog i novog sustava tako dugo dok se ne pokaže da novi sustav ispravno radi i da su se korisnici navikli na novi način rada
- bitno manje rizičan postupak u odnosu na izravno uvođenje
- nedostatak: potreba za dvostrukom obradom istih podataka, u starom i u novom sustavu → otpor korisnika

➤ Lokacija konverzije

- dijelovi organizacije smješteni na različitim zemljopisnim lokacijama
- ponekad se zasebnim lokacijama smatraju različite organizacijske cjeline u istom kompleksu (npr. prodaja, otprema, nabava)

➤ Probno uvođenje (*pilot conversion*)

- izravno/paralelno uvođenje sustava na jednoj lokaciji
- nakon uspješnog pilota, uvodi se na ostalim lokacijama
- prednost: dodatna razina provjere prije širenja, ograničenje problema na probu
- nedostaci: dulje trajanje, razdoblje u kojem dijelovi sustava koriste različite verzije

➤ Postupno uvođenje, fazno uvođenje (*phased conversion*)

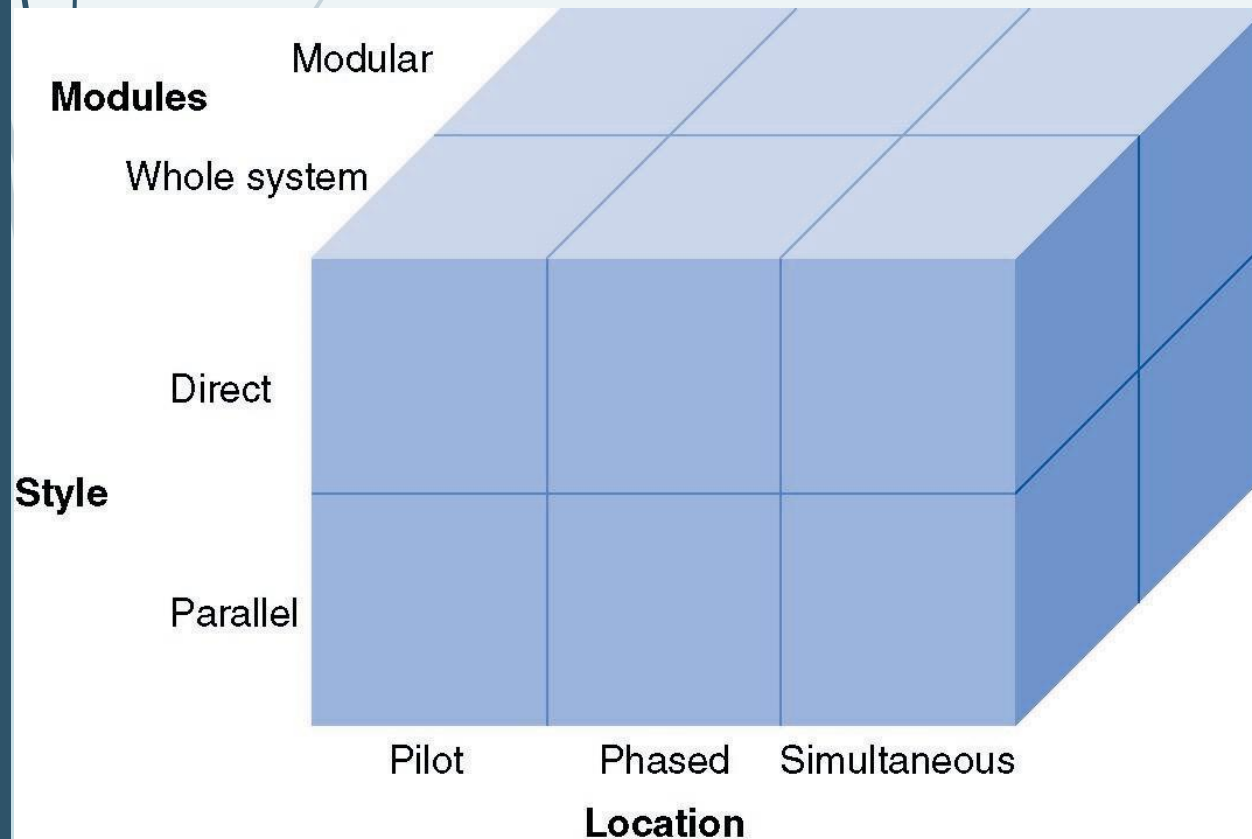
- uvođenje slijedno po grupama lokacija
- karakteristike slične probnom, ali zahtijeva manje ljudi

➤ Istovremeno uvođenje (*simultaneous conversion*)

- istovremeno uvođenje na svim lokacijama – najčešće izravno
- uklanja heterogenost, ali zahtijeva više ljudi

- Uvođenje cijelog sustava (*whole system conversion*)
 - čitav sustav instalira se odjednom – najčešći način
 - u slučaju velikih sustava (npr. ERP), može biti naporno za korisnike
- Modularno uvođenje (*modular conversion, staged conversion*)
 - postupna zamjena starog sustava novim, uvođenjem po dijelovima
 - izvedivo samo ako je moguć istovremeni rad oba (nekompletna) sustava
 - problemi: potreba za spojnim programima, tj. premošćivanjem
 - svaki modul treba moći raditi i sa starim i s novim sustavom, ili
 - novi sustav mora moći učahuriti funkcionalnost starog
 - prednost: postupni prijelaz, lakša poduka korisnika
 - nedostatak: dulje trajanje

Uvođenje	Rizik	Trošak	Trajanje
Izravno	visok	nizak (ako uspije)	kratko (ako uspije)
Paralelno	nizak	visok	dugo
Probno	nizak	srednji	srednje
Fazno	srednji	srednji	dugo
Istovremeno	srednji	srednji	kratko
Cijeli sustav	visok	srednji	kratko
Modularno	srednji	visok	dugo



Odabir strategije uvođenja u primjenu

- Upravljanje promjenama (Change Management)
 - podrška procesa prilagodbe korisnika na novi sustav
- Ključne uloge
 - sponzor promjena (sponsor of the change) – osoba koja želi promjenu
 - najčešće poslovnjak koji je pokrenuo zahtjev za novim sustavom ili viši rukovoditelj organizacijske cjeline u koju se uvodi sustav
 - presudno je da bude aktivan jer upravlja onima koji usvajaju sustav
 - agent promjena (change agent) – osoba/osobe koje vode promjenu
 - obično netko izvan organizacijske jedinice na koju se promjena odnosi
 - usvojitelj promjena (adopter) – osoba/osobe na koje se promjena odnosi
 - ljudi koji će u konačnici koristiti novi sustav
- Otpor promjenama
 - što je dobro za organizaciju, ne mora biti dobro za pojedinca
 - promjena radne procedure (drukčiji posao ili više posla) za istu plaću
 - promjena zahtijeva napor prilagodbe

- Priprema, obuka (training)
 - usvojitelji će prihvatiti sustav (u)koliko ih se osposobi
- Sadržaj poduke
 - **kako obaviti svoj posao (a ne kako koristiti aplikacije) !**
 - što korisnik treba/može napraviti (a ne što sustav može)
- Poduka krajnjih korisnika može uključivati:
 - opću informatičku kulturu (npr. uporaba osobnih računala)
 - funkcije i način upotrebe - korištenje aplikacija
 - posebna znanja za obavljanje posla (npr. optimizacija, CAD, GIS)
- Poduka tehničkog osoblja može uključivati:
 - operacijski sustav i uslužne programe
 - administriranje baze podataka
 - programske jezike, razvojne alate i alate za projektiranje

➤ Redoslijed poduke (preporuka)

- poduka tehničkog osoblja za održavanje i potporu korisnika, da ubrza primjenu
- zatim (niže) rukovodstvo, da podupre poduku i primjenu ostalih korisnika
- slijedi poduka (krajnjih) korisnika, prilagođena poslovnim procesima

➤ Postupci i tehnike poduke

- tečajevi
- probni rad fazi provjere rada sustava
- kvalitetni sustav interaktivne pomoći
- prikladna dokumentacija
- potpora tijekom primjene

➤ Izvođači poduke mogu biti

- unutarnji izvođači - npr. odjel informatike ili grupa za to osposobljenih djelatnika
- vanjski izvođači – konzultanti, specijalizirane ustanove ili visokoobrazovne ustanove

Održavanje

Sistemska potpora i održavanje sustava

40

- Post-implementacija, post-produkcija – nakon uvođenja, varijante
 - sistemska potpora (*system support*)
 - održavanje sustava (*system maintenance*)
 - ocjena projekta (*project assessment*)
- Sistemska potpora
 - nakon uvođenja sustav preuzima "operativna grupa" – služba informacijske potpore ili oformljeni centar kompetencija
 - pomoć korisnicima korištenju sustava (on-demand training)
 - pomoć korisnicima "kad nešto zapne" (hardver, mreža, virusi, ...)
 - računalna podrška (online support) – web stranice koje sadrže dokumentaciju, odgovore na često postavljana pitanja (FAQ, ...)
 - odjel pomoći (help desk)

- Višerazinska organizacija podrške
 - Prva razina podrške (1st level support) – odgovara na široki raspon zahtjeva
 - očekuje se da razriješi 80% problema
 - inače sastavlja zapisnik problema (problem report) i prosljeđuje 2. razini
 - Druga razina podrške (2nd level support) – složenija stručna pomoć
 - mogu biti timovi po struci: npr. desktop tim, mrežni tim
 - Ukoliko se ne pronađe rješenje nego se ispostavi da postoji bug
 - zapisnik problema postaje zahtjev za promjenom (change request) koji se prosljeđuje odjelu održavanja
- Sustav za praćenje problema (issue tracking system, trouble ticket system, incident ticket system)
 - sadrži bazu znanja o problemima, rješenjima, korisnicima, ...
 - omogućuje praćenje problema korisnika
 - sprema kartone (ticket), jednoznačno označene (unique reference number)
 - karton evidentira problem i intervencije

- Održavanje (IEEE, 1993):
 - Modifikacija programskog proizvoda nakon isporuke da bi se ispravili kvarovi, popravile performanse ili druga svojstva ili da bi se proizvod prilagodio promjenama okruženja
- Održavanje je trajna aktivnost koja započinje odmah po uvođenju
 - Bez obzira na to koliko je sustav dobar, kvarovi su neizbježni!
- Zahtjev za promjenom (change request)
 - manja verzija zahtjeva na sustav (system request) s početka životnog ciklusa
- Najčešći razlozi (po prioritetu)
 - prijava kvara
 - zahtjev za poboljšanje, npr. ugradnja nove funkcije
 - zahtjevi vanjskih sustava s kojima se očekuje integracija
 - nove inačice sistemskog softvera, npr. baze podataka ili OS
 - zahtjevi višeg rukovodstva, obično radi promjena strategije organizacije

➤ Preventivno

- podrazumijeva zaštitu od mogućih problema
- redovita izrada sigurnosnih kopija (backup)
- obavlja se periodički (dnevno, tjedno, mjesečno)

➤ Korektivno

- podrazumijeva popravak nakon što se problem pojavio
- vraćanje podataka iz sigurnosne kopije (restore)
- uklanjanje uzroka pogreške (ispravljanje programa)

➤ Adaptivno

- prilagodba funkcionalnosti (načina posluživanja)
- prilagodba strukture (promjene strukture podataka)
- poboljšanje performansi (optimizacija programa)

➤ Perfektivno

- nadgradnja sustava da bi se riješili novi problemi
- ugradnja novih mogućnosti (features)