„Često razmišljaj kako je svet samo biće koje ima jednu jedinu suštinu i dušu, kako sve što se dešava stane u samo jedno opažanje, kako je sve zajednički uzrok svega; misli kako je sve ono što se dešava isprepleteno i protkano.” (IV knjiga, stav 40.)

Marko Aurelije (121. – 180.): »Meditacije (samom sebi)«

S a d r ž a j

Predgovor ... 9

0. Uvod ... 11

1. Klasično .NET okruženje ... 16

1.1 Zajedničko izvršno okruženje – CLR ... 19

1.2 Zajednička biblioteka klasa – CCL ... 20

1.3 Zajednički međujezik – CIL ... 21

1.4 Zajednički sistem tipova – CTS ... 24

1.5 Zajednička jezička specifikacija – CLS ... 26

1.6 Sklopovi ... 28

1.7 Izvršavanje .NET programa ... 29

1.8 Struktura upravljanog modula ... 32

2. ASP.NET i veb aplikacije ... 35

2.1 Osobine ASP.NET-a ... 35

2.2 ASP.NET i HTTP ... 37

2.3 Kompajliranje u ASP.NET-u ... 38

2.4 Instaliranje u ASP.NET-u ... 40

2.5 Veb aplikacije ... 41

3. Arhitektura klasične ASP.NET stranice ... 45

3.1 Uloga veb servera ... 47

3.2 ASP stilovi programiranja ... 56

3.3 Veb konfiguracija ... 62

4. Osnovni objekat Page ... 69

4.1 Životni ciklus ASPX veb stranice ... 70

4.2 Događaji ASPX veb stranice ... 72

4.3 Serverske kontrole ASPX stranice ... 78

4.4 Fundamentalni objekti ASPX stranice ... 79

4.4.1 Page.Trace 80
4.4.2 Page.Application 80
4.4.3 Page.Session 80
4.4.4 Page.Context 80
4.4.5 Page.Server 82
4.4.6 Page.Request 84
4.4.7 Page.Response 87
5. Serverske kontrole 92
5.1 HTML serverske kontrole 100
5.2 ASPX serverske kontrole 103
5.2.1 Jednostavne ASPX kontrole 107
5.2.1.1 Unos i prikaz teksta 107
5.2.1.2 Izbor predefinisanih vrednosti 112
5.2.1.3 Iniciranje HTTP zahteva 115
5.2.1.4 Tabele 119
5.2.1.5 Prostori 125
5.2.1.6 Automatski PostBack 128
5.2.2 Složene ASPX kontrole 129
6. Validacija ulaza 147
6.1 RequiredFieldValidator kontrola 152
6.2 RangeValidator kontrola 155
6.3 CompareValidator kontrola 157
6.4 RegularExpression kontrola 158
6.5 CustomValidator kontrola 160
6.6 Validacione grupe 163
6.7 Sumarni prikaz i naglašavanje grešaka 165
7. Navigacioni sistem 172
7.1 Navigacione metode 172
7.1.1 Obična navigacija 173
7.1.2 Unakrsna navigacija 175

7.1.3 Redirekcija ... 177
7.1.4 Transfer ... 178
7.1.5 Remapiranje ... 178
7.2 Navigacione kontrole ... 182
7.2.1 Meni kontrola ... 182
7.2.2 Stablo kontrola ... 187
7.2.3 Putanja kontrola ... 192
7.2.4 Mapa sajta ... 193
8. Upravljanje stanjem ... 200
8.1 Klijentska strana upravljanja stanjem ... 202
8.1.1 Skrivena polja (HTML) ... 202
8.1.2 Upitni stringovi (HTTP) ... 204
8.1.3 Kolačići (HTTP) ... 206
8.1.4 Pregledno stanje (ASP.NET) ... 208
8.1.5 Kontrolno stanje (ASP.NET) ... 214
8.2 Serverska strana upravljanja stanjem ... 214
8.2.1 Sesijsko stanje ... 215
8.2.2 Aplikativno stanje ... 218
8.2.3 Keširanje podataka ... 221
9. Master stranice ... 226
10. Teme ... 240
11. Veb parčad ... 253
12. Korisničke serverske kontrole ... 267
13. Prilagođene serverske kontrole ... 281
13.1 Prilagođene kontrole dobijene modifikacijom jedne postojeće serverske kontrole ... 286
13.2 Prilagođene kontrole dobijene kompozicijom više postojećih serverskih kontrola ... 288
13.3 Prilagođene kontrole dobijene nasleđivanjem više svojstava ... 291
13.4 Prilagođene kontrole dobijene nasleđivanjem manje svojstava ... 293
13.5 Podrška prilagođenim kontrolama u Visual Studio programu ... 296

14. Veb servisi 303
14.1 XML SOAP veb servisi 305
14.2 WCF veb servisi 314
14.3 Web API servisi 321
15. Dinamička grafika 328
15.1 Isporuka dinamičke grafike 329
15.2 GDI+ grafičke primitive 335
15.3 ASP:Chart kontrola 343
16. Rad sa bazama podataka 356
16.1 ADO.NET klase 360
16.2 Izvori podataka 367
16.3 Jednostavne kontrole za podatke 374
16.4 Složene kontrole za podatke 380
16.5 Hijerarhijske kontrole za podatke 388
17. Rad sa XML podacima 392
18. Monitoring rada veb aplikacije 412
18.1 Praćenje rada veb stranica 413
18.2 Nadgledanje rada veb aplikacije 423
19. Greške i izuzeci 434
20. Korisnički profili i sigurnost 449
21. Kriptografija 480
22. Ostale teme 503
22.1 AJAX 503
22.2 Keširanje 510
22.3 Lokalizacija 512
22.4 MVC 515
22.5 LINQ 520
22.6 EFM 526
Literatura 530

Predgovor

Dobrodošli u knjigu *Klasično ASP.NET programiranje* i hvala na ukazanom poverenju. Ovo je još jedna knjiga o veb programiranju. Razlika ove knjige u odnosu na druge slične je u tome što ona obrađuje jedan specifičan programski jezik i platformu koja postaje i ostaje za duži period klasična. Pojam klasična se ovde upotrebljava kao sinonim za: „izvrsna u svom polju" i kao „dugotrajna čija važnost ne prolazi". Slede tri koncepta i izuma čija vrednost ne prolazi i posle desetina i stotina godina, iako se izvorne sprave više se proizvode, niti se koriste.

- **QWERTY tastatura.** Kada je američki pronalazač Lejtam Šouls 1868. patentirao prvu upotrebljivu pisaću mašinu, a malo kasnije osmislio QWERTY raspored tastera u četiri reda, nije mogao biti svestan da će i nakon vek i po isti raspored biti i dalje aktuelan na tastaturama računara i ekranima mobilnih telefona.
- **Binarni podaci.** Američki preduzetnik Herman Holerit je 1889. godine u svom saveznom patentu izložio ideju i mehanizam čitanja i pisanja binarnih podataka upisanih na bušene kartice. Binarni podatak je predstavljao postojanje (jedinica) ili odsustvo (nula) rupe na određenoj poziciji bušene kartice. Mašina koja je koristila binarne podatke sa bušenih kartica je korišćena za automatsku obradu podataka sa popisa u SAD 1890-te godine. Danas svaki računar i mobilni telefon kao osnovu procesiranja ima upravo binarne podatke.
- **8.3 format imena datoteka.** Za imena datoteka je usvojena konvencija imenovanja 1974. godine u CP/M operativnom sistemu za mikro računare da bude 8 znakova za ime datoteke i 3 znaka za nastavak imena, koji govori o kom tipu datoteke se radi, dok je između ta dva elementa bila tačka kao separator. Kada je Majkrosoft dizajnirao DOS operativni sistem za IBM-ov proizvod PC (engl. personal computer, lični računar) ugledao se na ovaj format, te je on od tada preživeo sve inkarnacije DOS-a i Windows-a pa ga stoga imamo na svakom računaru sa Windows sistemom i posle 50 godina od početnog dizajna.

Klasično je vredno, jer koncepti ostaju i preživljavaju sve prepreke. Neka klasični ASP.NET bude za veb programiranje isto kao jezik C u svetu programiranja!

Knjiga ima 22 poglavlja koliko elemenata ima još jedna klasika – feničansko pismo, koje je nastalo na prelazu iz bronzanog doba u gvozdeno i po kome su se skoro sva upotrebljiva pisma danas kreirala: grčki alfabet, latinica, ćirilica, hebrejsko pismo, arapsko pismo. Feničanskim slovom započinje svako poglavlje.



0. Uvod

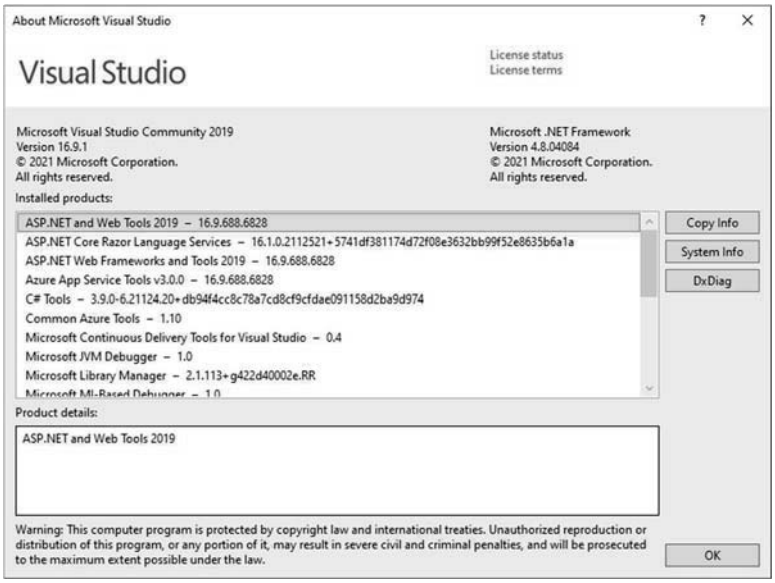
Knjiga *Klasično ASP.NET programiranje* je nastala kao plod autorovog višegodišnjeg profesionalnog iskustva sa veb programiranjem u Majkrosoftovoj .NET tehnologiji, i to kao predavača na visokoškolskim institucijama, ali i kao praktičara u privredi. Stoga je knjiga namenjena kako studentima na fakultetima i višim školama za ovladavanje predmetnom materijom, tako i profesionalnim programerima u svakodnevnom radu i pripremi za njega.

Nivo složenosti materije i brzine savladavanja sadržaja knjige je ocenjen kao srednje težak. Potrebna su bar osnovna predznanja vezana za sledeće termine:

- HTTP,
- HTML i
- C#.

Struktura knjige se ogleda u podeli na više kraćih tema. Svaka tema ima teorijski i praktični deo. Teorijski deo predstavlja stručni uvod i pojašnjenja pojedinih tehnologija, pristupa, podešavanja, programskih biblioteka sa svojim jezicima, naredbama, svojstvima itd. Praktični deo predstavljaju delovi kôda od interesa kao i celi programski listinzi.

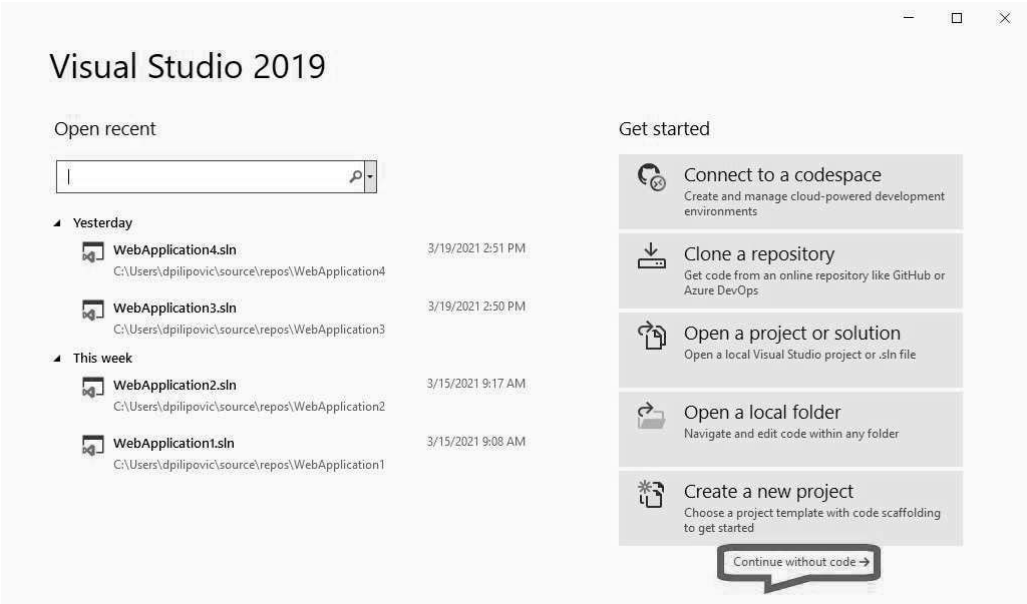
Programski kôdovi su isprobani u razvojnom okruženju *MS Visual Studio 2019* izdanje *Community*, koja je besplatna verzija ovog okruženja namenjena za studente, individualce i timove do 5 članova. Kada se *Visual Studio* skine sa zvanične veb lokacije i instalira na računar, potrebno je izabrati da se instaliraju još dodaci sa Slike 0-1.



Slika 0-1: Komponente *Visual Studio 2019 Community* verzije

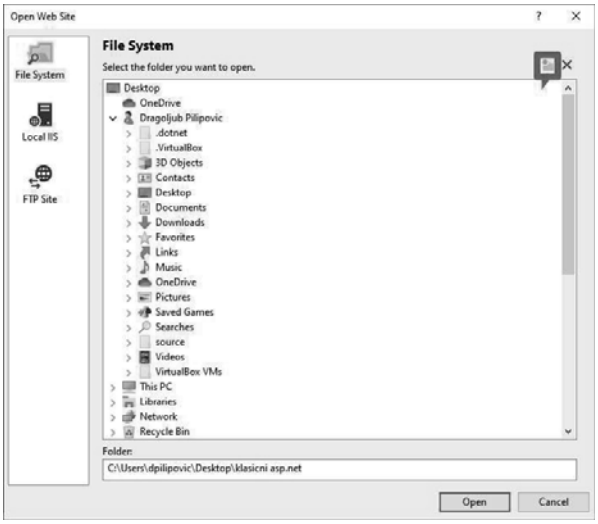
Svi primeri koje se nalaze u knjizi započinju i unose se na sledeći način.

Korak 1. Pri startovanju Visual Studio razvojnog okruženja treba izabrati opciju „Continue without code“, što će da otvori potpunu radnu površinu ovog alata.



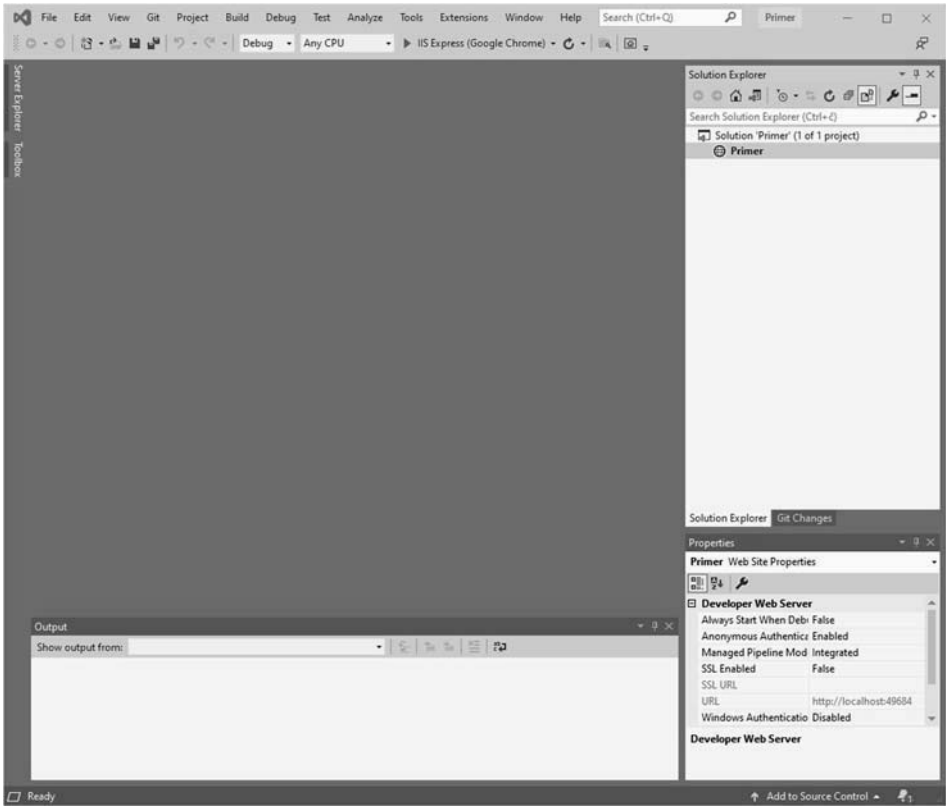
Slika 0-2: Pripremni ekran *Visual Studio* alata za razvoj

Korak 2. Ide se na meni File – Open – Web Site ili skraćeno sa tastature Shift + Alt + O da bi se dobio prozor sa naslovom „Open Web Site“.



Slika 0-3: Pravljenje i otvaranje novog direktorijuma za veb sajt

Korak 3. Napraviti novi direktorijum sa ikonicom „Create New Folder“. Dati mu ime pa ići na taster „Open“. Tada se dobija potpuno spremno radno okruženje kao na Slici 0-4.



Slika 0-4: Osnova radnog okruženja u VS 2019

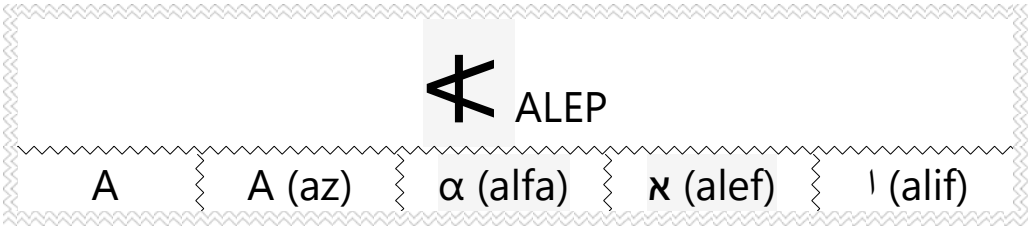
Prva gotova verzija .NET okruženja se pojavila 2002. godine i bila je namenjena samo Windows operativnom sistemu. Poslednja verzija .NET okruženja je 4.8 izašla u javnost 18. aprila 2019. godine i ona predstavlja krajnju verziju koje se neće više menjati, s tim što će za nju izlaziti bitne sigurnosne i manje popravke još dugo vremena. U Tabeli 0-1. se vidi odnos tri linije implementacija .NET-a:

- .NET Framework,
- .NET Core i
- Mono.

	.NET Framework	Mono	.NET Core
2002.	1.0		
2003.	1.1		
2004.		1.0, 1.1	
2005.	2.0		
2006.	3.0	1.2	
2007.	3.5		
2008.		2.0	
2009.		2.2, 2.4, 2.6	
2010.	4.0	2.8	
2011.		2.10	
2012.	4.5	3.0	
2013.		3.2	
2014.		3.4, 3.6, 3.8, 3.10	
2015.	4.6	3.12, 4.0	
2016.			1.0, 1.1
2017.	4.7	5.0, 5.2, 5.4	2.0, 2.1
2018.		5.8, 5.10	2.2
2019.	4.8	6.0	3.0, 3.1
2020.	----------		5.0
2021.			6.0

Tabela 0-1: Hronološki redosled implementacija .NET-a

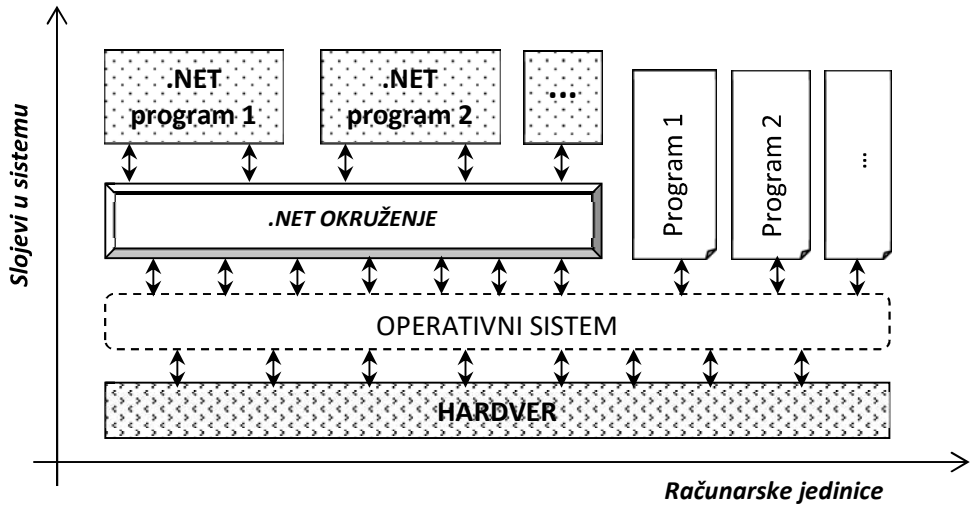
Smatramo da će .NET (i ASP.NET kao deo njega) će opstati uspešno u upotrebi u narednom periodu jer postoje na tržištu dvadeset godina, pa svako ko se bavi programiranjem na računarima, a posebno ko se bavi veb programiranjem treba da ga proučava i zna. U ovoj knjizi se za sve primere koriste .NET Framework verzije 4.8.


ALEP
A
A (az)
α (alfa)
א (alef)
ا (alif)

1. Klasično .NET okruženje

Definicija: **.NET okruženje** predstavlja složeno softversko okruženje namenjeno međuplatformskom razvoju i izvršenju softvera. Može se posmatrati kao dodatni softverski sloj direktno naslonjen na operativni sistem na kome je instaliran.

U ovom poglavlju će se dati odrednica .NET okruženja, onda će se opisati glavne komponente .NET okruženja i predstaviti način izvršenja .NET softvera.



Slika 1-1: Mesto .NET okruženja u računarskom sistemu

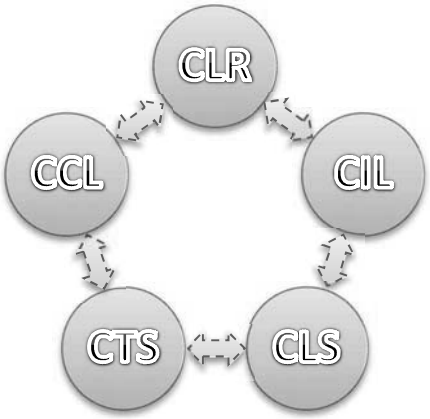
Dva su glavna aspekta posmatranja .NET okruženja. Prvi aspekt je opšte gledište korisnika računarskog sistema, a drugi je onaj aspekat naprednog korisnika tj. programera na .NET. Da bi se pokrenuo i izvršio .NET softver, računar u vidu hardvera i operativnog sistema mora da prepozna da je u pitanju posebna vrsta softvera, te da pokrene zajedničko izvršno okruženje za .NET – **Common Language Runtime (CLR)**. Programer je kreativni element dvosistema čovek-računar i njemu je potrebno da isprogramira rad .NET izvršnog okruženja preko skupa dostupnih biblioteka funkcija – **Common Class Library (CCL)** i tako dobije gotov i izvršiv .NET softver.

Softver napisan za .NET je izvršiv na više različitih platformi koje se ogledaju u instaliranim operativnim sistemima: Windows, Linux, macOS. Karakteristika portabilnosti se ogleda u mogućnosti da jednom napisan .NET program prenosi i izvršava se na bilo kojoj od spomenutih operativnih platformi bez dodatnih prilagođavanja toj platformi. Dodatno, .NET softver može da bude distribuiran na više

mesta, i može da bude razdvojen na komponente, koje se mogu nezavisno dodavati i menjati.

Portabilnost .NET softvera je omogućena putem dodatnog sloja između, s jedne strane, hardvera i operativnog sistema, i, s druge strane, korisnika .NET-a, bilo krajnjih korisnika bilo programera. Dodatni sloj je vrsta međujezika – **Common Intermediate Language (CIL)**. Programiranje se obavlja pisanjem programskog kôda u nekom od više .NET dijalekata. To su programski jezici koji na kraju proizvode softver razumljiv .NET zajedničkom izvršnom okruženju. Najpoznatiji .NET dijalekti su C#, Visual Basic, F#. Za podatke u svim .NET programima se koristi zajednički sistem tipova – **Common Type System (CTS)**, što omogućava laku i interoperabilnu razmenu podataka između programa napisanih u različitim .NET jezicima. Za direktnu interakciju između programa napisanih u različitim .NET dijalektima[1] se koristi zajednička jezička specifikacija – **Common Language Specification (CLS)**. Specifikacijom se definišu pravila koja moraju da ispoštuju pojedini .NET jezici i njihovi kompajleri[2] da bi se uspostavila interoperabilnost dva ili više .NET programa.

Ovih pet komponenti, sa skraćenicama CLR, CCL, CIL, CTS i CLS, čine .NET petorku koja predstavlja i pokreće celi .NET ekosistem.



Slika 1-2: .NET petorka

[1] U knjizi se koriste naporedo i često naizmenično sinonimi, kao što su softver/program/aplikacija, .NET programski jezik/dijalekat i sl. Smatramo da je bolje čitaoca uputiti da ovi termini ovde imaju isto/slično značenje, nego težiti jezičkom čistunstvu i naučnoj preciznosti i doslednosti u korišćenju termina.

[2] Termin je mogao da bude naveden kao prevodilac. Međutim, stavljena je transkribovana verzija engleskog izvornika koja jednoznačno određuje o čemu je reč. Smatramo da je ovaj pojam 1) već dovoljno ušao u svakodnevni govorni jezik programera (slično poput pojma hardver itd.), 2) asocira na postojeći pojam (kompilacija) 3) ali je ujedno različit od njega, 4) u potpunosti je transkribovan, 5) ne zvuči rogobatno itd.

Kao što je već rečeno, godine 2002. se pojavila prva gotova verzija .NET okruženja i bila je namenjena samo Windows operativnom sistemu. Njegova pojava je usko vezana za programski jezik C# koji je nastao kao posledica razvoja tokom DOT COM buma. Još jedna bitna stvar o postanku .NET okruženja i C# progjezika je želja Majkrosofta da adekvatno odgovori na pojavu i potencijal Java progjezika.

Verzija .NET okruženja:	Verzija CLR komponente:	Godina objave:
1.0	1.0	2002.
1.1	1.1	2003.
2.0	2.0	2005.
3.0	2.0	2006.
3.5	2.0	2007.
4.0	4	2010.
4.5	4	2012.
4.6	4	2015.
4.7	4	2017.
4.8	4	2019.

Tabela 1-1: Verzije .NET okruženja

.NET okruženje je standardizovano od strane međunarodnih tela, za razliku od direktnog konkurenta Jave, pa imamo:

- CLI (Common Language Infrastructure) koji opisuje opštu jezičku infrastrukturu koja je jezički i platformski nezavisna i koja omogućava izvršavanje softvera pisanog na jezicima visokog nivoa kroz ECMA-335 i ISO-23271 standarde, oba iz 2012. godine.
- C# progjezik je opisan u standardima ECMA-334 (2017. godina) i ISO-23270 (2018. godina).[3]

Softverska licenca je pravni instrument koji korisniku softvera propisuje kako ga može i mora koristiti. Svrha softverske licence je zaštita autorskih prava pisca softvera. Sve licence se ugrubo mogu podeliti na vlasničke licence i licence slobodnog softvera, a kriterijum po kome se one kategorišu je način na koji se softveri mogu umnožavati i distribuirati. Tako vlasničke licence dozvoljavaju korišćenje softvera pod svojim uslovima, ali vlasništvo nad softverom uvek ostaje njihovim piscima (proizvođačima).

[3] Ipak, ovde treba biti oprezan: nije standardizovano .NET okruženje nego CIL, a .NET okruženje je najpopularnija implementacija CIL standarda.

Licence za .NET okruženje su mešovitog tipa, nešto je pod vlasničkom licencom koju je propisao Majkrosoft kao proizvođač softvera, a pojedini delovi su izdati pod otvorenim licencama i to: MIT, Apač i BSD (Berkli softverska distribucija). Postoje dve bitne CLI implementacije koje su u potpunosti izdate pod otvorenom i slobodnom licencom:

- Mono projekat predstavlja nezavisno razvijano okruženje slično .NET-u sa ciljem da se izvršava na svim hardverskim i softverskim platformama: Windows, Linux, BSD, macOS, Solaris, Android, iOS, Nintendo, Playstation.
- .NET Core je još jedan Majkrosoftov projekat sa drugačijim principima, koji implementira elemente CIL i CCL.

1.1 Zajedničko izvršno okruženje – CLR

Definicija: **Zajedničko izvršno okruženje** (CLR – Common Language Runtime) je onaj operativni element .NET okruženja koji pronalazi, prepoznaje, učitava i upravlja .NET izvršivim entitetima. On je glavni element .NET okruženja i on se instalira na računar[4] – sve ostale .NET komponente su ili uključene u njega, ili je on implementirao njihovu funkcionalnost po specifikaciji. CLR izvršava upravljani kôd, tj. programski kôd koji je napisan na .NET međujeziku.

CLR se može posmatrati kao virtuelna računarska mašina[5], koja se pokreće u okviru običnog procesa operativnog sistema[6]. Svrha ove virtuelne mašine je da kroz dodatni sloj apstrakcije pruži platformski nezavisnu računarsku arhitekturu koja je ista za bilo koji hardver ili operativni sistem. Softver u .NET međujeziku, dobijen kompajliranjem sa viših .NET progjezika će se, bez dodatnih akcija, izvršiti kroz upravljanje od strane CLR-a na potpuno isti način na svakoj hardversko-softverskoj platformi.

Poslovi kojima CLR upravlja su sledeći:

- rad sa memorijom (virtuelna memorija, sakupljanje memorijskog otpada),
- učitavanje .NET aplikacija,
- rad sa programskim nitima,
- rukovanje izuzecima,
- provera tipova,
- sigurnost (zasnovana na ulogama i na kôdu).

[4] Glavna datoteka je mscoree.dll čije ime je skraćenica od MS Common Object Runtime Execution Engine. Na sistemu Windows nalazi se u %SystemRoot%\System32 folderu.
[5] Slična je Javina virtuelna mašina.
[6] Spominju se još termini procesna virtuelna mašina i aplikacijska virtuelna mašina.

Na CLR okruženju se mogu izvršavati sledeće vrste programa:

<table>
<tr><th colspan="2">Sa korisničkim interfejsom:</th><th>Bez korisničkog interfejsa:</th></tr>
<tr><td colspan="2">Konzolni (tekstualni, komandni)</td><td rowspan="2">Servisi operativnog sistema</td></tr>
<tr><td rowspan="2">Grafički:</td><td>Win32 (WinForms, Windows Forms)</td></tr>
<tr><td>WPF (Windows Presentation Foundation)</td><td rowspan="2">Veb servisi</td></tr>
<tr><td colspan="2">Veb (HTTP/HTML) – ASP.NET[7] aplikacije</td></tr>
</table>

1.2 Zajednička biblioteka klasa – CCL

Definicija: **Zajednička biblioteka klasa** (CCL – Common Class Library) predstavlja ogroman i koristan skup osnovnih i dodatnih klasa za korišćenje u .NET okruženju. Ove klase predstavljaju zamenu za skoro sve funkcionalnosti operativnog sistema a u njima su i klase sa novom funkcionalnošću. Na ovaj način CCL je zamena i nadogradnja operativnog sistema sa stanovišta programera.

CCL biblioteka je zasnovana na objektnom pristupu, te ih njihov korisnik može instancirati u svoje objekte, kao i nasleđivati. Imena klasa i njenih članova su logično i jednostavno nazvana. Poznavati .NET klase je isto kao poznavati šta sve može .NET pružiti programeru.

Interno je CCL je izdeljen na BCL i FCL. BCL (Base Class Library) što je osnovni skup klasa koje se implementiraju u svim alternativnim .NET i CIL okruženjima, npr. u NET Core, Mono, .NET Micro, Silverlight. FCL[8] (Framework Class Library) je skup svih .NET klasa, što uključuje BCL jer je on njegov nadskup. Većina osnovnih klasa se nalazi u datotekama MsCorLib.dll i System.dll.

Prostor imena:	Opis:
System.Data	Pristup podacima preko ADO.NET[9] mehanizma
System.Drawing	Izrada dvodimenzionalne grafike
System.Globalization	Lokalizacija
System.IO	Rad sa datotekama i folderima
System.Management	Kontrolisanje računara preko WMI mehanizma
System.Net	Mrežna komunikacija
System.Security	Zaštita podataka

[7] Active Server Pages.
[8] Ovo je pravo komercijalno ime za .NET biblioteku klasa, a CCL je stavljeno u ovu knjigu radi jednoobraznosti naziva .NET komponenti – da sve počinju sa common.
[9] ActiveX Data Objects.

System.Threading	Rad sa nitima.
System.XML	Rad sa XML podacima i šemama.

Tabela 1-2: Primeri prostora imena .NET klasa

Prostor imena:	Vrste programa:
System.Console	Konzolni
System.Windows.Forms	Grafički – Windows Forms
System.Windows	Grafički – Windows Presentation Foundation
System.Web.UI	Veb (ASP.NET) aplikacije
System.ServiceProcess	Servisi operativnog sistema
System.Web.Services	Veb servisi

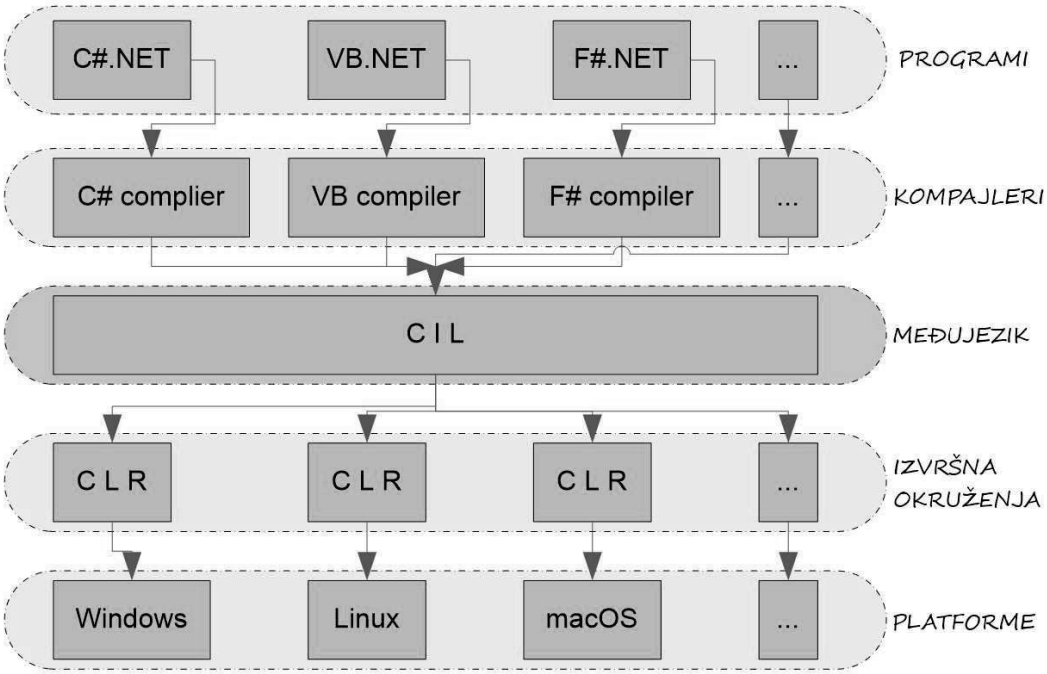
Tabela 1-3: Prostori imena za vrste programa

1.3 Zajednički međujezik – CIL

Definicija: **Zajednički međujezik** (CIL – Common Intermediate Language) je asemblerski jezik srednjeg nivoa, predviđen da se izvršava u zajedničkom izvršnom okruženju. On je agnostički nastrojen prema konkretnim izvršnim softversko-hardverskim platformama i nezavisan je od njih, iako je napravljen da liči na druge asemblerske jezike čiji se prevedeni programi direktno izvršavaju na hardverskim komponentama (od kojih je glavna komponenta računarski procesor). Koncept .NET međujezika liči na bajtkod u Java okruženju.

CIL je maternji jezik kojim govori .NET okruženje.

CIL je objektno-orijentisani progjezik. On poseduje tipove objekata i instrukcije koje kreiraju i inicijalizuju objekte, može da napravi nove klase, da ih nasledi, da definiše interfejse, kao i ostale objektne koncepte. Takođe sadrži naredbe za kreiranje i obradu izuzetaka, može da obrađuje elemente nizova itd. Međujezik poznaje sve klase iz zajedničke biblioteke klasa i zato može ih koristiti.



Slika 1-3: Međujezik je samo jedan

Programeri obično pišu softvere u višim progjezicima, to su .NET dijalekti poput C#, Visual Basic, F#. Potom ide kompajliranje – proces prevođenja iz viših progjezika u jezike niskog nivoa bliskih hardveru, koji se onda mogu pokrenuti i izvršiti. Za .NET okruženje jezik niskog nivoa je upravo zajednički međujezik – CIL. Kompajlirani CIL progkôd se zove upravljani kôd po Majkrosoftovoj terminologiji.

Međujezik koristi virtuelnu računarsku mašinu zasnovanu na steku[10]. To znači da njegove instrukcije prvo stavljaju (push) svoje operande na stek, a potom ih uzimaju (pop) kao rezultate obrade. Međujezik ne nudi instrukcije za rad sa memorijskim registrima procesora. Kao posledicu toga instrukcije .NET međujezika su poprilično jednostavne. Sledi primer (listing) sabiranja dva operanda (A + B):

Listing 1-1: Sabiranje u međujeziku

```
IL_0001:  ldagr.0
IL_0002:  ldagr.1
IL_0003:  add
```

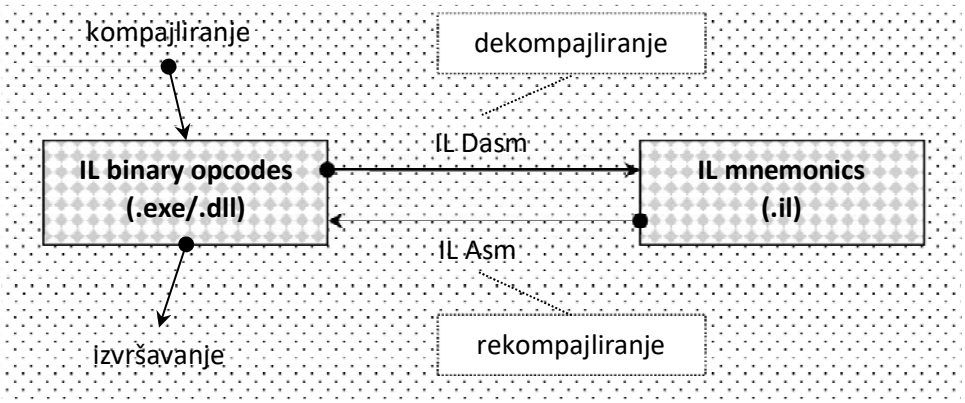
[10] LIFO (Last In, First Out) lista elemenata, gde poslednji u listu stavljeni element mora prvi da se iskoristi.

Prva naredba na IL_0001 liniji stavlja prvi operand A na stek, sledeća linija drugi operand B, te onda sledi naredba za sabiranje koja uzima drugi pa prvi operand sa steka i sabira ih. Pošto je CLR zasnovan na radu sa stekom instrukcija add *ne navodi operande* (jer zna da se mogu nalaziti samo na steku) ali *ni njihov tip* jer ova instrukcija prima različite tipove podataka da bi ih sabrala.

Majkrosoft je opskrbio .NET sa dva programerska alata za rad sa međujezikom:

- IL Assembler (ILAsm, ilasm.exe) koji služi za kompajliranje i rekompajliranje IL progkôda.
- IL Disassembler (ILDasm, ildasm.exe) koji služi za dekompajliranje prevedenih sklopova u čitljiv IL progkôd.

Kao proizvod kompajliranja programa napisanog u nekom .NET dijalektu se dobija sklop u kome je između ostalog IL progkôd tj. izvršni .NET kôd opisan u međujeziku. Kada se upotrebi spomenuti dekompajler, dobija se tekstuelna .il datoteka sa čitljivim (C)IL instrukcijama. Sada se može izvršiti izmena programa direktno sa instrukcijama međujezika. Sledeći korak je upotreba ILAsm kompajlera i dobijanje sklopa koji je izvršiv u .NET okruženju (najčešće je to jedna .exe datoteka).



Slika 1-4: Dekompajliranje, kompajliranje i rekompajliranje međukoda

U obe datoteke, i u .exe i u .il, se nalazi međukod. Razlika je u tome da su u izvršnoj datoteci samo binarne nule i jedinice, a IL instrukcije su tu zapisane binarnim opkodovima (operacionalnim kôdovima), dok su u .il datoteci one prikazane odgovarajućim mnemonicima jer je predstavljanje na mnemonički način ljudima prirodnije i lakše za koristiti. Npr. instrukciji sa opkodom 0x58 je pridružen mnemonik add.

Iz svakog kompajliranog sklopa je moguće izvući međukod postupkom reverznog inženjeringa. Na taj način je dostupan algoritam programa, njegovi objekti, metode i sve ostalo. Obfuskatori su pomoćni programi koji uzimaju međukod, onda prikrivaju pojedine informacije i zamagljuju procese i algoritme u što većem obimu, da bi na kraju se dobio izvršni program sa istom funkcionalnošću, a sa međukodom koji je teško ili skoro nemoguće čitljiv i razumljiv.

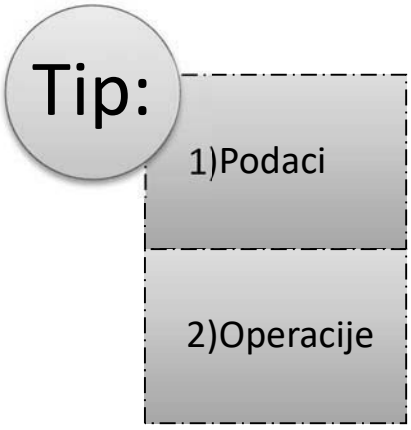
CIL instrukcija ima ukupno 100, a razvrstane su na osnovne (67 instrukcija) i instrukcije za objektni model (33 takvih).

♬ Poznavati međujezik je dovoljan uslov za programiranje u .NET okruženju.

1.4 Zajednički sistem tipova – CTS

Definicija: **Zajednički sistem tipova** (CTS – Common Type System) predstavlja specifikaciju tipova stvari kojima .NET može manipulisati. Tipovi i vrste tipova su opisani tako, što mogu od vrednosti sadržati, koje operacije postoje nad njima i među tipovima, i, na kraju, koji je odnos među tipovima vidljiv putem njihove hijerarhijske organizacije. Cilj je napraviti sigurne tipove, sa stanovišta jedne aplikacije, i razmenjive tipove, sa stanovišta dve ili više zasebnih .NET aplikacija.

Uopšteno gledano bilo koji tip se sastoji od: 1) podataka i 2) operacija. Podaci se (uslovno) mogu smatrati statičnim delom tipa, a operacije aktivnim delom odnosno ponašanjem tipa.



Slika 1-5: Struktura tipa

Tip može biti bilo koja stvar koja je opisiva[11]. Sa programerske tačke stanovišta tip je klasa koja se može instancirati ili interfejs koji se mora implementirati. Pored ovoga, tip može da bude struktura, enumeracija, niz i delegat. Pošto zajednički sistem tipova podržava funkcionalnu paradigmu programiranja kao i objektnu, postoje dve vrste tipova sa tog stanovišta:

1) **Vrednosni tipovi**. Ovi tipovi sadrže sekvencu bitova (bajtova) koji su njihova práva sadržina. Namenjeni su za definisanje jednostavnih i najčešće jednosložnih tipova kao što su celi brojevi. Ovakvi tipovi postoje u mnogobrojnim drugim progjezicima. Ovde spadaju još strukture i enumeracije.

2) **Referencni tipovi**. Ovi tipovi sadrže vrednost koja ukazuje na memorijsku adresu gde se nalazi prava sadržina. Pored ukazne vrednosti (reference) postoji naveden tačno tip o kome se radi, što podrazumeva da se zna sve o tipu, pa se kaže da su ovakvi tipovi samoopisujući. Ovde spadaju klase, interfejsi, delegati, nizovi. Predstavljaju osnovu O-O programiranja.

CTS definiše više pravila zbog kojih slede bitnija:

- **Kardinalnost tipa**: tip može da sadrži nijedan, jedan ili više članova.
- **Vrste članova** nekog tipa mogu biti neki iz sledećeg skupa:

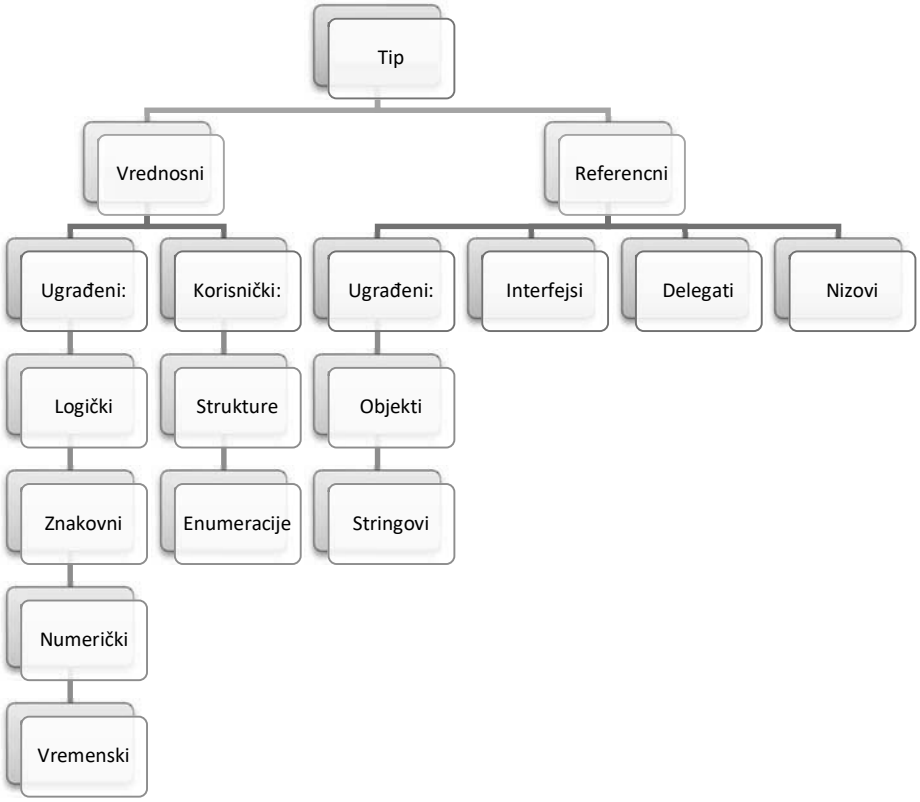
Članovi tipa = {konstruktor, finalizator, operator, metoda, događaj, indekser, svojstvo, polje, konstanta}

- **Vidljivost i pristupačnost** tipu i njegovim članovima. Programer može odrediti da je tip potpuno vidljiv (javan) ili ne, uz više gradacija između ove dve krajnosti. Pristup članovima tipa može biti neki iz skupa:

Pristup članovima tipa = {private, family, family-and-assembly, assembly, family-or-assembly, public}

- **Jednostruko nasleđivanje**: samo jedno nasleđivanje može da postoji između dva tipa.
- **Sve je objekat**: svi tipovi u konačnici moraju poteći od vrhovnog System.Object tipa.

[11] U klasičnoj terminologiji se spominje tip podataka, koji opisuje: 1. – kakva je memorijska reprezentacija podataka (npr. zauzima 1 bajt); 2. – skup vrednosti koje može da ima (za spomenuti bajt to je od 0 do 255) i 3 – koje operacije se mogu izvesti sa njima (npr. sabiranje i oduzimanje).



Slika 1-6: Hijerarhija vrsta tipova

1.5 Zajednička jezička specifikacija – CLS

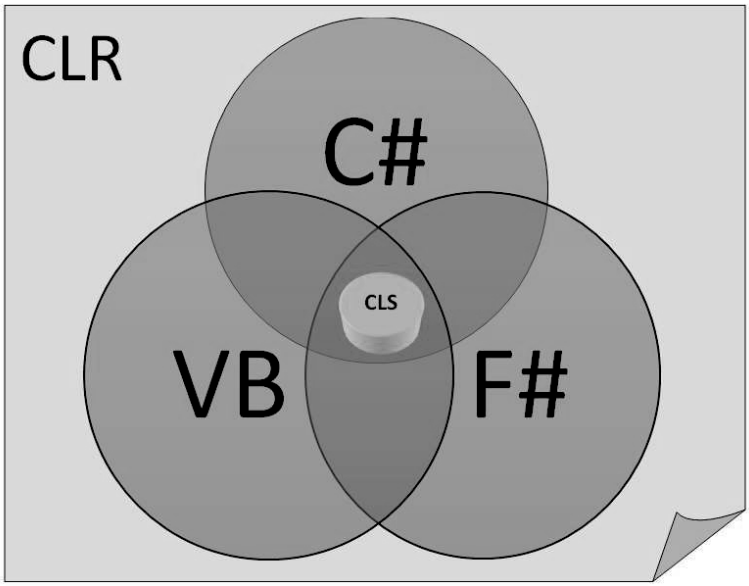
Uloga kompajlera u .NET ekosistemu je veoma važna jer pri prevođenju sa višeg progjezika na međujezik se mora obratiti pažnja na način i tačnost prevođenja. Tvorci kompajlera tu imaju glavnu ulogu. Moguće je da kompajleri za različite .NET dijalekte proizvode različite programe u međujeziku u smislu da nisu sve mogućnosti podjednake u svim rezultujućim kôdovima. Da bi se ovo sprečilo, osmišljena su pravila koja usklađuju mogućnosti .NET progjezika i njihovih kompajlera.

Definicija: **Zajednička jezička specifikacija** (CLS – Common Language Specification) je skup minimalnih pravila koje moraju zadovoljiti .NET programski jezici i njihovi kompajleri da bi međusobno mogli razmenjivati tipove i druge jezičke konstrukte. Cilj je potpuna programska interoperabilnost.

Pojedini .NET programski dijalekti ne prave razliku između velikih i malih slova u nazivima promenljivih, drugi nemaju za svaki brojčani tip označene vrednosti, već kreću uvek od nule u pozitivnom smeru, treći imaju neke objektne apstrakcija visokog nivoa

koji drugi jezici nemaju, pa čak nema ga ni međujezik itd. Da bi se tipovi i drugi programski konstrukti mogli u potpunosti ispravno videti u CIL programima dobijenim od različitih kompajlera i jezika, treba koristiti samo one odlike jezika koje su zajedničke za sve njih.

Progjezici u .NET-u i njihovi kompajleri ne mogu iskoristiti sve mogućnosti što ih nudi zajedničko izvršno okruženje, zajednički sistem tipova i zajednički međujezik; obično je u pitanju njihov podskup mogućnosti. Dalje, progjezici i kompajleri obično imaju nadskup mogućnosti koje propisuje zajednička jezička specifikacija. I još, taj nadskup različitih jezika/kompajlera ne mora nužno biti međusobno identičan.



Slika 1-7: Mesto zajedničke jezičke specifikacije spram .NET jezika i .NET okruženja

Prvo CLS pravilo je značajno i služiće kao primer za uvid u strukturu ostalih pravila[12]:

CLS pravilo 1:	Sva pravila se primenjuju samo na one delove tipa koji su vidljivi i kojima se može pristupiti van sopstvenog sklopa.

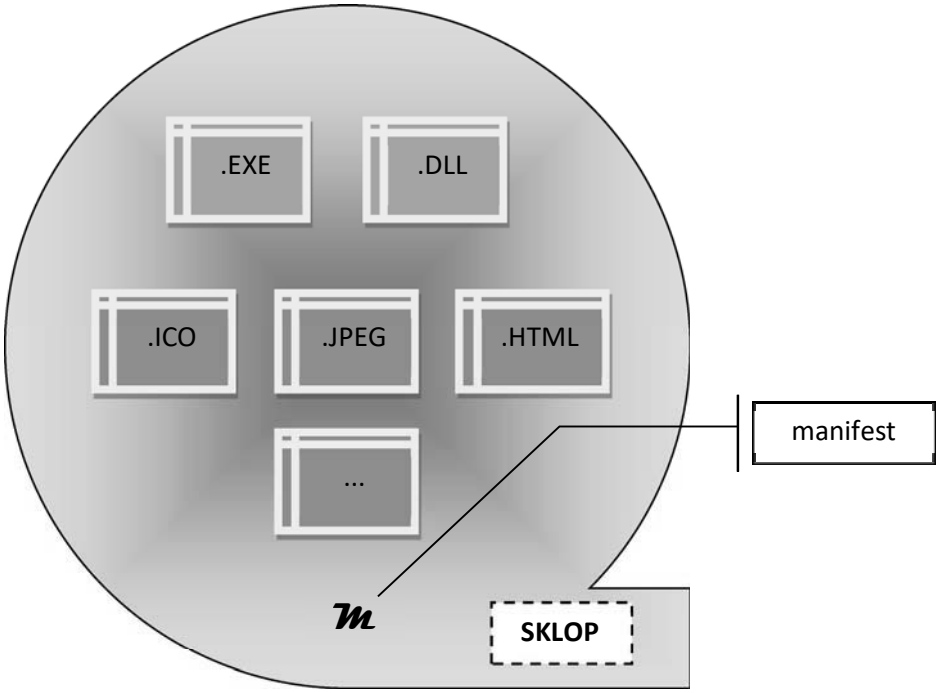
Većina pravila se može opisno sažeti na sledeći način. Uzimajući u obzir da je ono što se može razmeniti između različitih programa upravo tip, svi delovi nekog programa se prevode u tipove koji mogu imati samo dve osnovne vrste članova: podatak i operaciju.

[12] Po standardu definisanom u referenci [2012, 1] pravila ima ukupno 48.

Tako svaki program (u stvari, .NET jezik u kome je pisan i njegov kompajler) mora biti u stanju da pristupi poljima sa podacima i da pozove metodu sa traženom operacijom.

1.6 Sklopovi

Definicija: **Sklop** (engl. assembly) je apstraktna jedinica koja služi za logičko grupisanje izvršnih i resursnih datoteka. Sklop se dobija kao rezultat kompajliranja .NET programa i to je najmanja upotrebna jedinica posmatrana od strane .NET okruženja.



Slika 1-8: .NET sklop

Sklop može da bude sastavljen od jedne ili više datoteka. Obavezna datoteka u sklopu mora da bude izvršnog tipa i to: 1) neposredno izvršne u vidu .exe datoteke ili 2) posredno izvršne u vidu funkcionalne biblioteke, gde datoteka ima nastavak .dll. Ostale datoteke su resursnog tipa u formatima slika, teksta i sl.

Svaki sklop u sebi ima metapodatke koji ga čine samoopisivim. Manifest je zapisan metapodacima[13]. Kreira ga kompajler .NET jezika. Neke metapodatke u manifestu generiše sâm kompajler, a drugi se održavaju preko .NET atributa[14]. Manifest je zapisan

[13] Metapodaci su u obliku bloka binarnih podataka koji su u .NET-u izdeljeni na "tabele" (kako ih MS zove).
[14] Atributi su van opsega ove uvodne knjige, kasnije će se samo na kratko spomenuti.

u jednoj od datoteka sklopa sa nekom drugom namenom ili u posebnoj datoteci koja služi baš u tu svrhu.

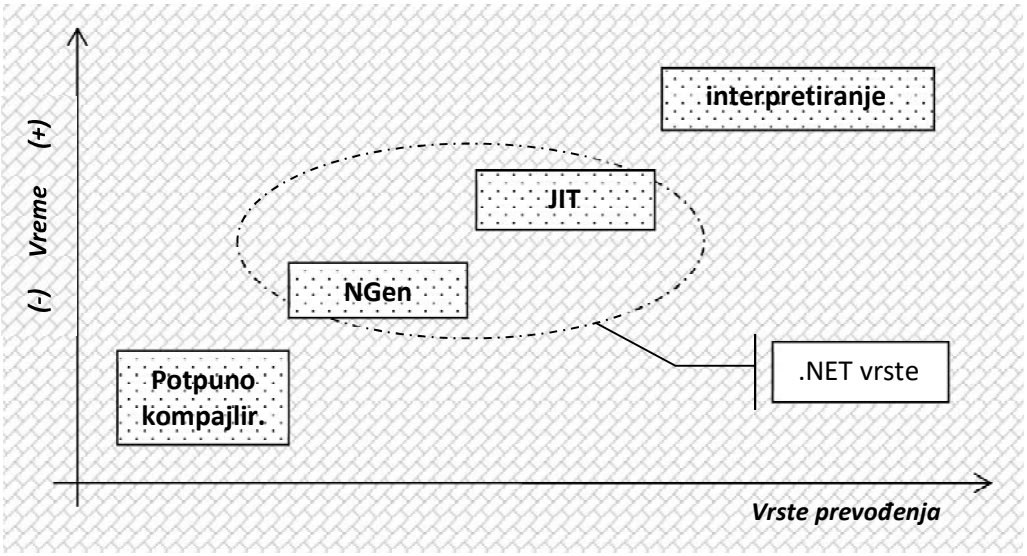
U manifestu se, između ostalih, nalaze sledeći metapodaci:

1. **ime** sklopa;
2. **verzija** sklopa se sastoji od četiri cifre razdvojene tačkama;
3. **kultura** sklopa;
4. **sigurnosni zahtevi** sklopa;
5. **spisak datoteka** koji čine sklop;
6. **spisak drugih sklopova** potrebnih za rad sklopa;
7. **spisak tipova** koji se mogu izvesti van sklopa;
8. **digitalni potpis** sklopa.

1.7 Izvršavanje .NET programa

Spominjanje kompajliranja u poglavlju o međujeziku je bilo tačno, ali nije bilo potpuno. U stvari, postoje dve vrste (ili faze) kompajliranja u .NET-u:

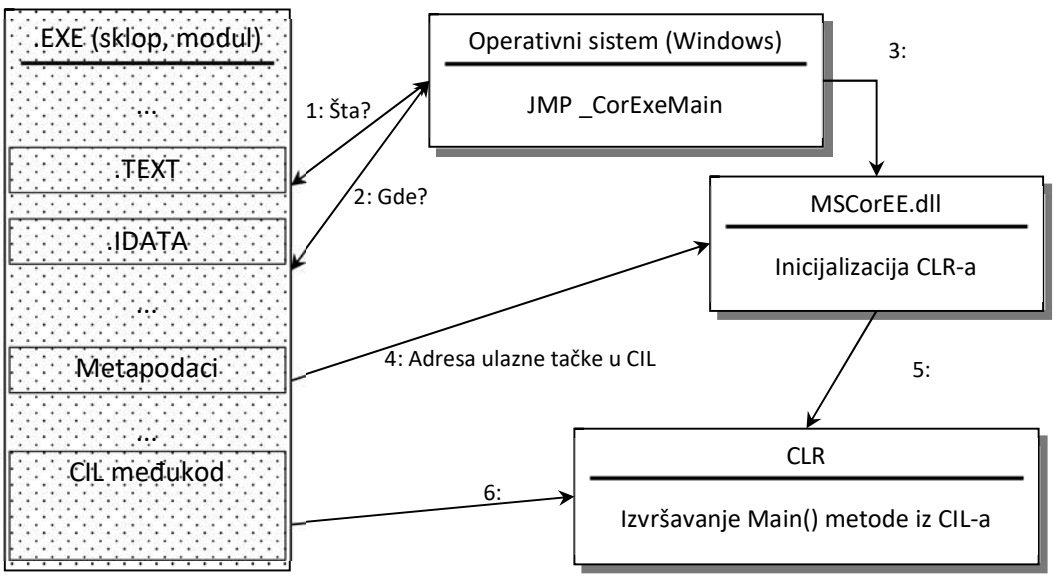
1. **Kompajliranje u međujezik**, gde se kao ulaz uzima programski listing napisan u višem programskom jeziku .NET ekosistema, a kao rezultat se dobije .exe/.dll datoteka sa međukodom.
2. **Kompajliranje u izvršni oblik**, koje postoji jer međujezik ne poznaje izvršnu platformu pa je potrebno instrukcije međujezika prevesti na potpuno izvršive mašinske instrukcije ciljne platforme.



Slika 1-9: Vremena prevođenja programa u izvršni oblik

Prvo kompajliranje izvodi kompajler progjezika, a drugo izvodi .NET izvršno okruženje. Međutim, kod drugog kompajliranja postoje razlike, pa će se napraviti *poređenje vrsta prevođenja* programskog kôda:

- **Potpuno kompajliranje**, gde se na kraju dobije mašinski program koji je potpuno izvršiv na konkretnoj platformi (konkretan procesor, familija procesora ili isti skup instrukcija). Npr.: za .NET ovo može da uradi Mono okruženje.
- **Kompajliranje pri instaliranju**, gde se međukod prevodi u izvršiv kada se .NET program instalira na računar. Ovo je jedna od opcija .NET okruženja, a izvodi se dodatnim alatom ngen.exe (Native Image Generator).
- **Kompajliranje po pokretanju**, gde se pri prvom pokretanju programa on u hodu prevodi u izvršni oblik, što izvodi dio .NET izvršnog okruženja pod imenom Jitter. Ovo je upravo-na-vreme tj. JIT (Just-In-Time) kompajliranje. Uobičajena opcija jer je najčešća u .NET okruženju.
- **Interpretiranje**, gde je program uvek u izvornom obliku i prevodi se u izvršni oblik naredbu po naredbu, neprestano. Ne postoji u .NET-u za više progjezike.

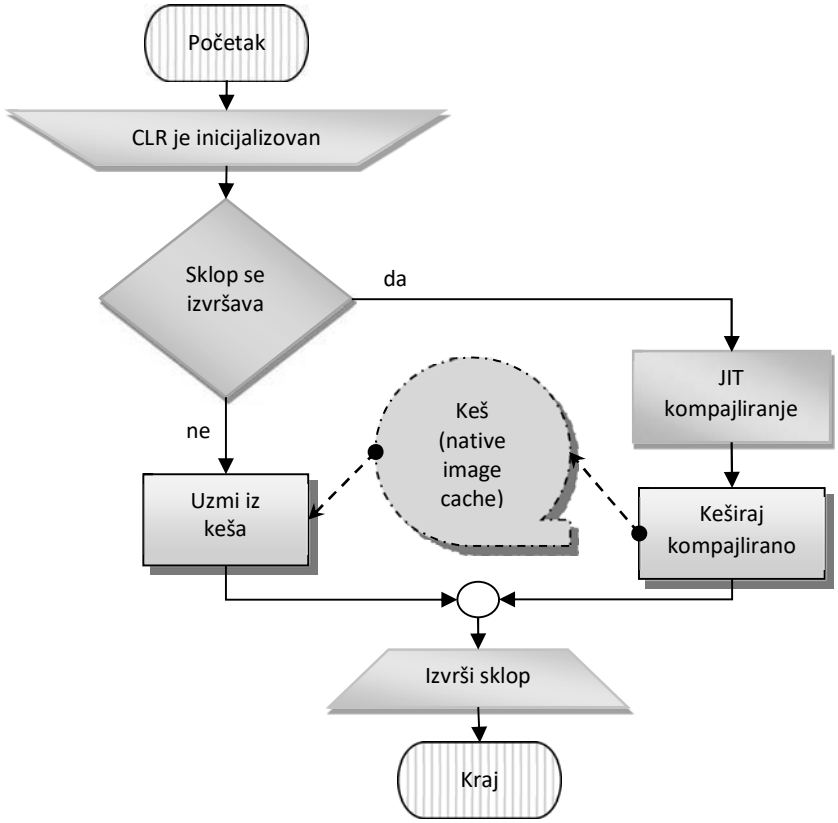


Slika 1-10: Pokretanje .NET programa – inicijalizacija izvršnog okruženja

Pre nego što počne JIT kompajliranje, postoje određeni koraci koji će se odigrati za neki .NET softver. Kada se pokreće .NET sklop u vidu izvršne datoteke, operativni sistem će da ispita njen sadržaj. U odeljku .TEXT je zapisana instrukcija koja će pokrenuti .NET

izvršno okruženje[15], a u odeljku .IDATA je zapisano u kojoj datoteci[16] se nalazi izvršni kôd potreban za inicijalizaciju CLR-a i njegovo dalje izvršavanje. Potom se traži ulazna tačka u .NET sklop, traži se obično Main() metoda, i onda se njen CIL kôd kompajlira u niz mašinski izvršivih instrukcija putem Jitter-a.

JIT kompajliranje je operativna srž izvršnog CLR okruženja. Ovo kompajliranje koristi razne napredne tehnike, poput dinamičkog kompajliranja, rekompajliranja, adaptivne optimizacije, prediktivne analize itd. Poprilično je brzo u izvršavanju, osim kada se prvi put pokreće .NET sklop. Posle prvog puta prevedena slika u binarnom obliku se smešta u keš memoriju[17] i odatle se po potrebi koristi, te ima najveću brzinu izvršavanja koje hardver može da pruži jer su instrukcije iz keša sada u mašinskom obliku.

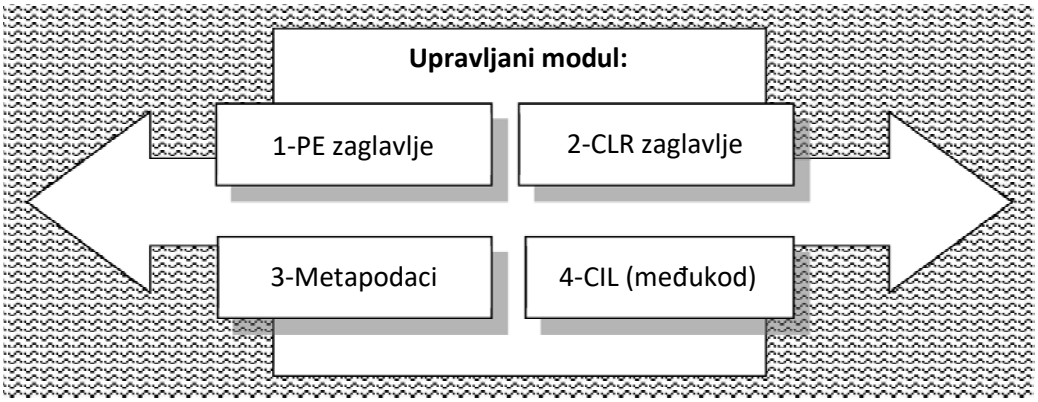


Slika 1-11: JIT kompajliranje

[15] Znači da se .NET okruženje pokreće po potrebi, nije stalno u radnoj memoriji.
[16] To je MSCorEE.dll datoteka.
[17] Naziva se .NET Native Image Cache.

1.8 Struktura upravljanog modula

Definicija: **Upravljani modul** (managed module) je osnovna jedinica dobijena kao proizvod kompajliranja izvornog programa napisanog u višem .NET progjeziku. Upravljani modul ima format PE – portable executable.



Slika 1-12: Struktura upravljanog modula

Sada se može precizirati da se sklop sastoji od jednog ili više upravljanih modula i nula ili više resursnih datoteka. Manifest govori koji moduli pripadaju sklopu.

Modul može da bude konzolni, grafički (GUI) program ili biblioteka funkcija (DLL), što je zapisano u RE zaglavlju. Prva dva su izvršna u užem smislu jer imaju ulaznu tačku u program tj. poseduju Main() metodu, a treći ne poseduje takvu ulaznu tačku već njegove tipove/metode mora da pozove neki drugi izvršivi program.

PE zaglavlje, koje je standardno za Windows operativni sistem, u slučaju .NET programa je poprilično zanemareno, jer su bitne informacije sada u CLR zaglavlju. Odeljci .TEXT i .IDATA su zapisani u RE zaglavlju.

Ulazna tačka u neki .NET program se fiksno zove „Main() metoda“ i njen položaj u vidu heksadecimalne adrese odnosno heksadecimalnog pomeraja se nalazi u CLR zaglavlju, dok se sama metoda u svom međujezičkom obliku nalazi u CIL odeljku modula.

U metapodacima modula postoji niz tabela koje se mogu grupisati u dve kategorije:

- metatabele sa definicijama;
- metatabele sa referencama.

Metatabela:	Opis:
ModuleDef	Samo jedan red postoji u ovoj tabeli i on identifikuje modul.
TypeDef	Za svaki tip u modulu postoji jedan red. Dodatno ukazuje na vezu sa redovima u sledeće četiri metatabele.
MethodDef	Za svaku metodu unutar modula postoji red sa podacima.
FieldDef	Svako polje sa podacima ima jedan red sa podacima o sebi u ovoj metatabeli.
PropertyDef	Svako svojstvo ima po jedan red o sebi u ovoj metatabeli.
EventDef	Za svaki događaj definisan u modulu postoji po jedan red u ovoj metatabeli.

Tabela 1-4: Značajne metatabele sa definicijama entiteta modula

Metatabela:	Opis:
AssemblyRef	Za svaki sklop koji je referenciran u modulu se nalazi po jedan red sa potrebnim podacima.
ModuleRef	Postoji po jedan red za svaki spoljni RE modul koji poseduje tipove koji se koriste i referenciraju u tekućem upravljanom modulu.
TypeRef	Za svaki tip referenciran u modulu postoji jedan red sa podacima o njemu.
MemberRef	Svaki član (polje, svojstvo, metoda) koji je referenciran u modulu se ovde opisuje.

Tabela 1-5. Značajne metatabele sa referencama entiteta modula