MATT WEISFELD

# OBJEKTNO ORIJENTISANI NAČIN MIŠLJENJA

PETO IZDANJE

Računarski fakultet

CET

**OBJEKTNO ORIJENTISANI NAČIN MIŠLJENJA**
**Matt Weisfeld**
Prevod petog izdanja

ISBN 978-86-7991-426-2

Autorizovan prevod sa engleskog jezika petog izdanja knjige
The Object-Oriented Thought Process



| | |
|---|---|
| Prevod | Jasna Gonda |
| Recenzent | dr Bojana Dimić Surla,<br>profesor na Računarskom fakultetu |
| Tehnički urednik | Vesna Petrinović |
| Prelom | Zoran Stanković |
| Gl. i odg. urednik | Jovana Ristić |
| Izdavači | **Računarski fakultet**<br>Beograd, Knez Mihaila 6/VI<br>tel. (011) 2627-613, 2633-321<br>www.raf.edu.rs<br><br>**CET**<br>Computer Equipment and Trade<br>Beograd, Skadarska 45<br>tel/fax: (011) 3243-043, 3235-139, 3237-246<br>www.cet.rs |
| Za izdavača | Dragan Stojanović, direktor |
| Tiraž | 1000 |
| Štampa | „Pekograf", Beograd |

Nastavno-naučno veće Računarskog fakulteta na 142. elektronskoj sednici održanoj 27. 4. 2020. godine donelo je odluku da prevod knjige "Objekto orijentisani način mišljenja" autora Matt Weisfeld-a bude štampana kao univerzitetski udžbenik.

# Sadržaj

**1**
Uvod u objektno orijentisane koncepte ..... 5

**2**
Kako razmišljati na jeziku objekata ..... 33

**3**
Još neki objektno orijentisani koncepti ..... 47

**4**
Anatomija klase ..... 67

**5**
Smernice za projektovanje klasa ..... 77

**6**
Projektovanje korišćenjem objekata ..... 91

**7**
Ovladajte nasleđivanjem i kompozicijom ..... 105

**8**
Radna okruženja i ponovno korišćenje: projektovanje pomoću interfejsa i apstraktnih klasa ..... 125

**9**
Pravljenje objekata i objektno orijentisano projektovanje ..... 147

**10**
Projektni obrasci ..... 161

**11**
Izbegavanje zavisnosti i visoko spregnutih klasa ..... 175

**12**
SOLID principi objektno orijentisanog projektovanja ..... 187

Index ..... 205

# Uvod

## Opseg ove knjige

Kao što naslov nagoveštava, ova knjiga se bavi objektno orijentisanim (OO) procesom razmišljanja. Mada su izbor teme i naslova knjige važne odluke, te odluke nisu uopšte jednostavne kada je reč o visoko konceptualnoj temi. Mnoge knjige se bave raznim nivoima programiranja i objektno orijentisanog razvoja. Više popularnih knjiga pokriva teme koje obuhvataju OO analizu, OO dizajn, OO programiranje, obrasce projektovanja, OO podatke (XML), objedinjeni jezik za modelovanje (UML – Unified Modeling Language), OO veb razvoj, OO mobilni razvoj, različite OO programske jezike i mnoge druge teme povezane sa OO programiranjem (OOP).

Međutim, mada pažljivo proučavaju ove knjige, mnogi zaboravljaju da sve ove teme imaju zajedničku osnovu – kako se razmišlja na OO način. Često mnogi softverski profesionalci, kao i studenti, proučavaju ove knjige, a da pri tom ne ulažu odgovarajuće vreme i napor da zaista shvate projektne koncepte iza tog koda.

Tvrdim da se učenje OO koncepata ne postiže učenjem određenog razvojnog metoda, nekog programskog jezika ili skupa projektnih alata. Objektno orijentisani razvoj je, jednostavno rečeno, jedan način razmišljanja. Ova knjiga se u potpunosti bavi objektno orijentisanim načinom razmišljanja.

Nije lako razgraničiti jezike, metode i alate od OO načina razmišljanja. Često se ljudi upoznaju sa OO konceptima tako što prosto uskoče u neki programski jezik. Na primer, pre mnogo godina veliki broj C programera se upoznao sa objektno orijentisanim programiranjem tako što su prešli direktno na C++, a da nisu bili čak ni površno upoznati sa OO konceptima.

Važno je razumeti značajnu razliku između učenja OO koncepata i programiranja u OO jeziku. Ovo mi je postalo jasno mnog pre nego što sam počeo da radim na prvom izdanju ove knjige, kada sam čitao članke kao što je članak Craiga Larmana „Šta UML jeste, a šta nije" (What the UML Is—and Isn't). U tom članku Larman izjavljuje:

> „Nažalost, u kontekstu softverskog inženjerstva i jezika UML za izradu dijagrama, ponekad se veština čitanja i pisanja UML notacije izjednačava sa umećem objektno orijentisane analize i projektovanja. To, naravno, nije tačno, jer je ova druga sposobnost mnogo važnija od prve. Prema tome, preporučujem obuku i obrazovne materijale koji se zasnivaju na intelektualnoj veštini objektno orijentisane analize i projektovanja, a ne učenje UML notacije ili upotrebe konkretnih alata."

Prema tome, iako učenje jezika za modelovanje predstavlja važan korak, daleko je važnije najpre naučiti OO veštine. Učiti UML pre potpunog savladavanja OO koncepata liči na proučavanje električnih dijagrama bez prethodnog osnovnog znanja o elektricitetu.

Isti problem se javlja sa programskim jezicima. Kao što je već rečeno, mnogi C programeri su prešli u oblast objektno orijentisanog programiranja tako što su prešli na C++ pre direktnog izlaganja OO konceptima. To se odmah pokaže na intervjuu. Često su programeri koji tvrde da rade u C++ jednostavno radili kao C programeri koji koriste C++ kompajlere. Čak i sada, sa dobro utemeljenim jezicima kao što su C# .NET, VB .NET, Objective-C, Swift i Java, nekoliko ključnih pitanja u intervjuu za posao može lako da otkrije nedostatak razumevanja OO koncepata.

Rane verzije Visual Basic-a nisu OO. C nije OO, a C++ je razvijen tako da bude kompatibilan unazad sa C. Zbog toga je sasvim moguće da se koristi C++ kompajler, a da se pri tom upotrebljava samo C sintaksa i zanemaruju sve OO mogućnosti jezika C++. Objective-C je projektovan kao proširenje standardnog jezika ANSI C. Što je još gore, programer može da upotrebi neke OO elemente i da napravi program koji će biti nerazumljiv kako za OO programere, tako i za one koji to nisu.

Zato je od suštinskog značaja da kod učenja OO razvojnih okruženja najpre naučite osnovne OO koncepte. Oduprite se iskušenju da pređete odmah na programski jezik i umesto toga najpre posvetite vreme učenju objektno orijentisanog načina razmišljanja.

## Šta je novo u petom izdanju

Kao što je u ovom uvodu više puta rečeno, moja zamisao za prvo izdanje je bila da se držim koncepata, a ne da se fokusiram na neku određenu tehnologiju u nastajanju. Mada se i za peto izdanje i dalje držim tog cilja, takođe uvodim više „kontraargumenata" nego u ranijim izdajima. Pod tim mislim da, iako je objektno orijentisan razvoj daleko najpopularniji, on nije jedini.

Od vremena kad je prvo izdanje ove knjige završeno 1999. godine, pojavile su se mnoge tehnologije, a neke su se izgubile. U to vreme Java se tek razvijala i bila je primarni jezik za OO razvoj. Veb stranice su ubrzo postale deo svakodnevnog života i poslovanja. Svi znamo koliko su mobilni uređaji postali sveprisutni. U proteklih 20 godina razvijači softvera su upoznali XML, JSON, CSS, XSLT, SOAP i RESTful Web Services. Android uređaji koriste Javu i sada Kotlin, dok iOS uređaji koriste Objective-C i Swift.

Pokušavam da naglasim da smo u poslednjih 20 godina (i četiri izdanja ove knjige) prihvatili mnogo tehnologija. Moj prvenstveni cilj za ovo izdanje jeste da sve to svedem na prvobitnu namenu prvog izdanja, na osnovne objektno orijentisane koncepte. Smatram da sav uspeh prvog izdanja knjige može da se pripiše tome što se usredsredilo na temeljne objektno orijentisane koncepte. Na neki način, obišli smo pun krug, zato što ovo izdanje obuhvata sve gorepomenute tehnologije.

Konačno, koncepti koji u krajnjoj liniji objedinjavaju te tehnologije u jednu metodologiju projektovanja su predstavljeni kao SOLID, koji je utkan u sva poglavlja ovog izdanja, kao i u dva nova poglavlja na kraju knjige.

Pet SOLID principa su:

- **SRP** – Single Responsibility Principle (jedan i samo jedan razlog za izmenu)
- **OCP** – Open/Close Principle (klasa treba da bude zatvorena za izmene, ali otvorena za proširenja)
- **LSP** – Liskov Substitution Principle (klase se mogu bez problema zameniti osnovnim klasama)
- **ISP** – Interface Segregation Principle (uvek je bolje deklarisati više interfejsa sa manjim brojem metoda)
- **DIP** – Dependency Inversion Principle (klasa treba da zavisi od apstrakcija, a ne od konkretnih implementacija)

Često mislim da prvih devet poglavlja predstavljaju ono što ja smatram klasičnim objektno orijentisanim principima. Poslednja tri poglavlja, o projektnim obrascima, izbegavanju zavisnosti i SOLID principima, oslanjaju se na klasične principe i predstavljaju jednu snažnu metodologiju.

## Kome je knjiga namenjena?

Ova knjiga predstavlja uvod u koncepte objektno orijentisanog programiranja. Izraz koncepti je značajan jer, iako se kôd svakako koristi da podvuče teme o kojima je reč, prvenstveni fokus ove knjige jeste da usadi kod čitaoca objektno orijentisan način razmišljanja.Takođe je važno da programeri shvate da OOP ne predstavlja zasebnu paradigmu (kako mnogi veruju) – OOP je jednostavno jedan deo ogromnog skupa alatki, koji je dostupan savremenim programerima softvera.

Kada je 1995. godine prvobitno stvoren materijal za prvo izdanje ove knjige, OOP je bio u povoju. Ovo mogu da kažem zato što, osim začetaka OO jezika kakav je Smalltalk, u to vreme nije bilo pravih objektno orijentisanih jezika u igri. C++, koji ne nameće objektno orijentisane strukture, bio je preovlađujući jezik zasnovan na jeziku C. Java 1.0 je izdata 1996, a C# 1.0 tek 2002. U stvari, kada je 1999. godine izašlo prvo izdanje ove knjige, nije bilo izvesno da će OO zaista postati vodeća razvojna paradigma. (Java 2 je izdata tek decembra 1998.) Uprkos trenutnoj dominaciji, u sklopu OOP-a postoje neke zanimljive pukotine kojima se treba pozabaviti.

Prema tome, publika za prvo izdanje se razlikuje od današnje publike.

Od 1995. pa čak do 2010. u suštini sam ponovo obučavao mnoge strukturne programere za umetnost OOP-a. Velika većina tih studenata je odrasla uz COBOL, FORTRAN, C i VB kako na fakultetu, tako i na poslu. Danas studenti koji završavaju fakultet, pišu video igrice, prave veb sajtove ili proizvode mobilne aplikacije skoro su sigurno naučili da programiraju koristeći neki objektno orijentisani jezik. Zato se pristup petom izdanju ove knjige značajno razlikuje od prvog izdanja ili drugog itd. Umesto da strukturne programere učimo da postaju OO razvijači, mi sada učimo programere koji su odrasli uz OO jezike.

Namenjena publika za ovu knjigu uključuje poslovne menadžere, dizajnere, programere, i rukovodioce projekata: ukratko, sve one koji žele da steknu opšte znanje o tome šta je to objektno orijentisan način razmišljanja. Nadam se da će čitanjem ove knjige dobiti čvrstu osnovu za prelazak na druge knjige koje se bave naprednijim temama.

## Pristup u knjizi

Do sada treba da je očigledno da ja čvrsto verujem da treba dobro savladati objektno orijentisan način razmišljanja pre prelaska na programski jezik ili jezik za modelovanje. Ova knjiga je puna primera koda i UML dijagrama klasa; međutim, ne morate da znate nijedan konkretan programski jezik ili UML da biste je čitali. Posle svega što sam rekao o prioritetu učenja koncepata, zašto u ovoj knjizi ima toliko koda i dijagrama klasa?

Prvo, kôd i dijagrami klasa su odlični za ilustrovanje OO koncepata. Drugo, oni su sastavni deo OO procesa i treba im se posvetiti na uvodnom nivou. Cilj nije usredsređivanje na jezik Java, C# i tako dalje, već da se pomoću njih olakša shvatanje koncepata u osnovi.

Imajte na umu da se meni zaista sviđa da koristim UML dijagrame klasa kao vizuelnu pomoć za ilustrovanje klasa, njihovih atributa i metoda. U stvari, dijagrami klasa su jedina komponenta UML-a koja se koristi u ovoj knjizi. Verujem da UML dijagrami klasa nude odličan način za modelovanje konceptualne prirode modela objekata. Stalno koristim modele objekata kao nastavno sredstvo za ilustrovanje dizajna klasa i odnosa među klasama.

Primeri koda u knjizi ilustruju koncepte kao što su petlje i funkcije; međutim, razumevanje samog koda nije preduslov za razumevanje koncepata. Moglo bi da bude korisno imati pri ruci knjigu koja opisuje sintaksu određenih jezika ako želite više detalja.

Ne mogu previše da naglasim da ova knjiga nije udžbenik jezika Java, C# .NET, VB .NET, Objective-C, Swift ili UML, za njih bi bili potrebni zasebni tomovi.Takođe je važno razumeti da je ovo knjiga o konceptima i da svrha primera u ovoj knjizi nije nužno opisivanje optimalnog načina da se projektuju klase; oni su nastavne vežbe namenjene da podstaknu razmišljanje o OO konceptima. Na primer, očigledno je da na poslu nećete praviti mnogo modela u kojima se koriste pingvini ili psi koji ne laju, ali njihovo korišćenje je zabavan način da se objasne koncepti. Imajući sve to u vidu, ja se samo nadam da će ova knjiga kod vas otvoriti apetit za ostale OO teme, poput OO analize, OO projektovanja i OO programiranja.

## Izvorni kôd koji se koristi u knjizi

Kôd primera opisanih u celoj knjizi raspoloživ je na veb sajtu izdavača. Idite na informit.com/register i registrujte svoju knjigu da biste dobili pristup preuzimanju.

# 1

# Uvod u objektno orijentisane koncepte

Mada mnogi programeri nisu toga svesni, razvijanje objektno orientisanog (OO) softvera se pojavilo početkom šezdesetih godina dvadesetog veka. Tek sredinom i krajem devedesetih je objektno orijentisana paradigma počela da se zahuktava, uprkos činjenici da su popularni objektno orijentisani programski jezici, kao što su Smalltalk i C++, već bili u širokoj upotrebi.

Uspon OO metodologija se poklapa sa pojavom interneta kao platforme za poslovanje i zabavu. Ukratko, objekti dobro funkcionišu preko mreže. Pošto je postalo očigledno da internet nije prolazna pojava, objektno orijentisane tehnologije su već bile dobro pozicionirane za razvoj novih tehnologija zasnovanih na mreži.

Važno je imati na umu da je naslov ovog prvog poglavlja „Uvod u objektno orijentisane koncepte". Osnovna reč je ovde 'koncepti' a ne 'tehnologije'. Tehnologije se u softverskoj industriji veoma brzo menjaju, dok se koncepti razvijaju. Ja koristim reč 'razvijaju' zato što se oni, mada ostaju relativno stabilni, ipak menjaju. To je ono što je zaista privlačno kada se fokusirate na koncepte. Uprkos njihovoj doslednosti, oni su uvek podložni reinterpretacijama, pa to omogućava neke vrlo zanimljive rasprave.

Ovaj razvoj može lako da se prati preko proteklih 25 godina praćenjem napredovanja različitih industrijskih tehnologija od prvih primitivnih pretraživača iz sredine i kraja devedsetih pa do mobilnih/telefonskih/veb aplikacija koje danas preovlađuju. Kao i uvek, novi razvoj je stalno na vidiku dok istražujemo hibridne aplikacije i tako dalje. Tokom ovog puta, OO koncepti su bili na svakom koraku. Zbog toga su teme u ovom poglavlju toliko važne. Ovi koncepti su danas jednako relevantni kao što su bili pre 25 godina.

## Osnovni koncepti

Prvenstveni cilj ove knjige jeste da vas navede na razmišljanje o tome kako se koncepti koriste u projektovanju objektno orijentisanih sistema. Istorijski posmatrano, objektno orijentisane jezike definišu: enkapsulacija, nasleđivanje i polimorfizam (ono što zovem 'klasični' OO). Prema tome, ako jezik ne implementira sve to, on se ne smatra potpuno

objektno orijentisanim. Pored ova tri pojma, ja uvek uključujem i kompoziciju; pa moja lista objektno orijentisanih koncepata glasi:

- Enkapsulacija
- Nasleđivanje
- Polimorfizam
- Kompozicija

Sve ćemo detaljno opisati u ostatku ove knjige.

Jedno od pitanja sa kojima sam se borio još od prvog izdanja je kako se ovi koncepti direktno odnose na trenutnu dizajnersku praksu, koja se stalno menja. Na primer, uvek je postojala rasprava o korišćenju nasleđivanja u OO dizajnu. Da li nasleđivanje zapravo remeti enkapsulaciju? (Ova tema će biti obrađena u kasnijim poglavljima.) Čak i sada, mnogi programeri pokušavaju da izbegnu nasleđivanje koliko god mogu. Dakle, postavlja se pitanje: da li nasleđivanje uopšte treba koristiti?

Kao i uvek, moj stav jeste da se držim koncepata. Bez obzira na to da li koristite nasleđivanje, barem morate da shvatite šta je to kako biste bili u stanju da odlučujete o dizajnu na bazi znanja. Važno je da ne zaboravite da ćete na nasleđivanje skoro sigurno naići prilikom održavanja koda, pa morate u svakom slučaju da ga naučite.

Kao što je pomenuto u uvodu, knjiga je namenjena onima koji žele opšte upoznavanje sa osnovnim OO konceptima. Imajući to u vidu, u ovom poglavlju predstavljam osnovne objektno orijentisane koncepte u nadi da će vam to obezbediti solidnu osnovu za donošenje važnih dizajnerskih odluka. Koncepti koji su ovde obuhvaćeni tiču se većine, ako ne i svih tema obrađenih u narednim poglavljima, u kojima se ta pitanja mnogo detaljnije istražuju.

## Objekti i prevaziđeni sistemi

Kako je OO počeo da preovlađuje, jedno od pitanja sa kojima se suočavaju programeri je spajanje novih OO tehnologija sa postojećim sistemima. Postavljale su se granice između OO i strukturnog (ili proceduralnog) programiranja, što je u to vreme bila preovlađujuća razvojna paradigma. Meni je to uvek bilo čudno jer, po mojem mišljenju, objektno orijentisano i strukturno programiranje se međusobno ne takmiče. Ona su komplementarna, pošto se objekti dobro integrišu sa strukturnim kodom. Čak i sada često čujem sledeće pitanje: jeste li vi strukturni programer ili objektno orijentisani programer? Bez oklevanja bih odgovorio: i jedno i drugo.

Na isti način, objektno orijentisani kôd nije namenjen da bude zamena za strukturni kôd. *Mnogi prevaziđeni sistemi* (to jest, stariji sistemi koji se već koriste) koji nisu OO sasvim dobro obavljaju posao, pa zašto njihovim menjanjem ili zamenom rizikovati potencijalne katastrofe? U većini slučajeva nema potrebe da ih menjate, bar ne samo promene radi. Sistemima koji su pisani bez OO ništa suštinski ne nedostaje. Međutim, potpuno novi razvoj svakako opravdava da se razmotri upotreba OO tehnologija (u nekim slučajevima, nema drugog izbora).

Mada je u poslednjih 25 godina došlo do stalnog i značajnog porasta u OO razvoju, zavisnost globalne zajednice od mreža kao što su internet i mobilne infrastrukture pomogla je da on još

više preovlada. Bujice transakcija koje se vrše u pretraživačima i mobilnim aplikacijama otvorile su potpuno nova tržišta, gde je veliki deo softverskog razvoja nov i neopterećen brigom o nasleđenom prevaziđenom softveru. Čak i kada postoje problemi sa prevaziđenim softverom, postoji trend da se prevaziđeni sistemi spakuju u objekte omotače.

> **Objekti omotači**
>
> Objekti omotači su objektno orijentisani kôd koji u sebi sadrži drugi kôd. Na primer, možete da uzmete strukturni kôd (kao što su petlje i uslovi) i omotate ga u objekat da bi izgledao kao objekat. Možete takođe u objekte omotače da omotate funkcionalnosti, kao što su bezbednosni elementi, neprenosivi hardverski elementi i tako dalje. Omotavanje strukturiranog koda je detaljno obrađeno u poglavlju 6 „Projektovanje korišćenjem objekata".

Jedno od najzanimljivijih područja softverskog razvoja je spajanje prevaziđenog koda sa mobilnim i veb sistemima. U mnogim slučajevima, mobilni veb pristup se na kraju povezuje sa podacima koji se nalaze na mejnfrejm računaru. Postoji potražnja za programerima koji su u stanju da kombinuju veštinu mejnfrejm programiranja i mobilnog veb programiranja.

Verovatno u svakodnevnom životu srećete objekte a da toga niste ni svesni. To može da bude u vašim kolima, kada razgovarate mobilnim telefonom, koristite svoj sistem kućne zabave, igrate igrice na računaru i u mnogim drugim situacijama. Elektronski autoput je, u suštini, postao zasnovan na objektima. Kako preduzeća teže ka mobilnom vebu, ona teže ka objektima, pošto su tehnologije koje se koriste za elektronsku trgovinu pretežno po prirodi OO.

> **Mobilni veb**
>
> Nema sumnje, pojavljivanje interneta pružilo je veliki zamah za prelazak na objektno orijentisane tehnologije. To je zato što su objekti pogodni za korišćenje na mrežama. Mada je internet bio na čelu ove promene u paradigmi, mobilne mreže su se sada priključile na značajan način. U ovoj knjizi, izraz *mobilni veb* će se koristiti u kontekstu koncepata koji se odnose kako na razvoj mobilnih aplikacija, tako i na veb razvoj. Izraz *hibridna* aplikacija se ponekad koristi za aplikacije koje se prikazuju u pretraživačima kako na veb, tako i na mobilnim uređajima.

## Proceduralno naspram OO programiranja

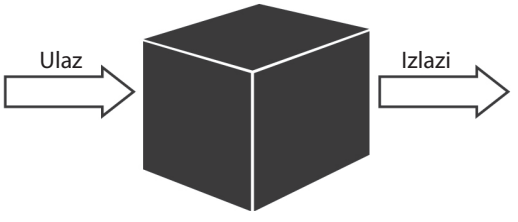
Dok se nismo dublje upustili u prednosti OO programiranja, razmotrićemo jedno fundamentalnije pitanje: šta je, u stvari, objekat? Ovo pitanje je istovremeno i složeno i prosto. Složeno je zato što učenje svakog metoda za razvijanje softvera nije trivijalno. Jednostavno je zato što ljudi već razmišljaju u smislu objekata.

> **SAVET**
>
> U jednom predavanju, u okviru YouTube videa, koje je držao OO guru Robert Martin izneo je mišljenje da su tvrđenje da „ljudi razmišljaju u smislu objekata" skovali ljudi iz marketinga. Prosto, tema za razmišljanje.

Na primer, kada posmatrate neku osobu, vi je vidite kao objekat, a objekat se definiše pomoću dve komponente: atributa i ponašanja. Osoba ima atribute, kao što je boja očiju, starost, visina i tako dalje. Osoba ima i ponašanja, kao što su hodanje, govor, disanje i tako dalje. U svojoj osnovnoj definiciji, *objekat* je entitet koji sadrži *oboje*: podatke i ponašanje. Reč oboje je ključna razlika između OO programiranja i drugih metodologija programiranja. Na primer, kod proceduralnog programiranja se kôd stavlja u potpuno razdvojene funkcije ili procedure. U idealnom slučaju, kao što se vidi na slici 1.1, te procedure potom postaju „crne kutije", gde se unose ulazni podaci i dobijaju izlazni podaci. Podaci se stavljaju u zasebne strukture, a obrađuju se unutar tih funkcija ili procedura.

**Razlika između OO i proceduralnog**

Kod OO dizajna, atributi i ponašanja se nalaze u istom objektu, dok su u proceduralnom ili strukturiranom dizajnu atributi i ponašanja obično razdvojeni.




Slika 1.1 Crna kutija.

Kako je OO dizajn postajao sve popularniji, jedna od činjenica koje su na početku usporavale njegovo prihvatanje bilo je to što je postojalo mnogo sistema koji nisu bili OO, a koji su sasvim dobro funkcionisali. Zato nije bilo poslovnog smisla da se sistemi menjaju samo radi promene. Ko god je upoznat sa bilo kojim računarskim sistemom, zna da svaka promena može da dovede do katastrofe – čak i kada promena izgleda neznatna.

To se desilo sa slabim prihvatanjem OO baza podataka. U jednom trenutku, kod pojave OO razvoja izgledalo je verovatno da će OO baze podataka zameniti relacione baze podataka. Međutim, to se nikada nije desilo. Preduzeća su već imala mnogo novca investiranog u relacione baze podataka i jedan preovlađujući faktor protiv konverzije bio je: one su funkcionisale. Kada su svi troškovi i rizici pretvaranja sistema iz relacionih u OO baze podataka postali očigledni, nije bilo nikakvog uverljivog razloga za prelazak.

U stvari, poslovne snage su sada pronašle srećnu sredinu. Danas veliki deo praksi softverskog razvoja ima više metodologija, kao što su OO i strukturirana.

Kao što se vidi na slici 1.2, u strukturiranom programiranju su podaci često razdvojeni od procedura. Podaci su često globalni, pa je lako promeniti podatke koji su izvan opsega vašeg koda. To znači da pristup podacima nije kontrolisan, pa je nepredvidljiv (to jest, više funkcija može da ima pristup globalnim podacima). Drugo, pošto nemate kontrolu nad time ko ima pristup podacima, testiranje i rešavanje grešaka je mnogo teže. Objekti rešavaju ove probleme tako što spajaju podatke i ponašanja u jedan lep, kompletan paket.