

Projektiranje informacijskih sustava

SDLC faza dizajna – dizajn programa

Ak. god. 2011/2012

Dizajn programa

- Funkcionalnost informacijskog sustava implementirana je u programu(\ima) koji će sa napisati tijekom faze implementacije informacijskog sustava.
- Dizajna programa određuje koji se programi trebaju napisati, na koji način se pojedini napisani dijelovi kôda međusobno povezuju kako bismo dobili integralno funkcionalnu aplikaciju te na koji način programeri trebaju napisati te programe.

Dizajn programa

- Tijekom dizajna programa donose se važne implementacijske odluke, poput koji će se programski jezik (jezici) koristiti za razvoj programa i sl.
- Koraci prilikom dizajna programa su:
 1. Definiranje fizičkog dijagrama toka podataka (*physical data flow diagram*) koji se izvodi iz dijagram toka podataka dobivenih u fazi analize programa.
 2. Definiranje dijagrama strukture (*structure chart*) povezivanjem svih procesa koji se izvode u sustavu.
 3. Izrada detaljnih instrukcija za pisanje programa, programske specifikacije (*program specifications*).

Fizički dijagrama toka podataka

- Fizički DFD ima iste komponente i za njega vrijede ista pravila kao i kod logičkog DFD (dekompozicija i uravnoteženje).
- Razlika između fizičkog i logičkog DFD je u tome što fizički DFD sadrži implementacijske tehničke detalje o izradi sustava (npr. spremište podataka je UTF8 tekstualna datoteka na disku).

Fizički dijagrama toka podataka

- Za tranziciju logičkog DFD-a u fizički DFD potrebno je provesti četiri koraka:
 1. dodavanje implementacijskih referenci
 2. postavljanje granice između čovjeka i uređaja
 3. dodavanje sistemskih procesa, podataka i tokova
 4. ažuriranje podataka u tokovima

Dodavanje implementacijskih referenci

- Prvi korak u transformiranju logičkog DFD-a u fizički DFD je dodavanje implementacijskih referenci elementima logičkog DFD-a.
- Implementacijske reference se definiraju za:
 - spremišta podataka (npr. način pohrane podataka (Oracle baza - tablica studenta, ASCII datoteka na disku,...)),
 - procese (npr. programski jezik u kojem će određeni proces biti isprogramiran (C# program, PHP skripta, ...)),
 - tokove podataka (npr. fizički medij za tok podataka (papirnati izvještaj, forma za unos podataka, ...)).
- Reference se smještaju uz ime DFD elementa.
- Vanjski entiteti nisu dio sustava i oni se ne mijenjaju.

Postavljanje granice između čovjeka i računala

- Drugi korak je postavljanje granice između čovjeka i uređaja (računalo, printer, skener,...).
- Granica se označava iscrtanom crtom koja omeđuje elemente (proces, podatke) koje radi čovjek od procesa koje radi uređaj jer u sustavu mogu postojati procesi koje nije moguće automatizirati ili nije isplativo automatizirati.

Dodavanje sistemskih procesa, podataka i tokova

- Dodavanje sistemskih procesa, skladišta podataka i tokova podataka je korak u kojem se u postojeći logički DFD dijagram dodaju oni procesi, skladišta podataka i tokovi podataka koji nisu vezani uz definiranu funkcionalnost sustava.
- Ovi dodaci mogu biti uključeni zbog tehničkih nedostataka sustava, zbog potrebe za kontrolom ili zbog rukovanja greškama.

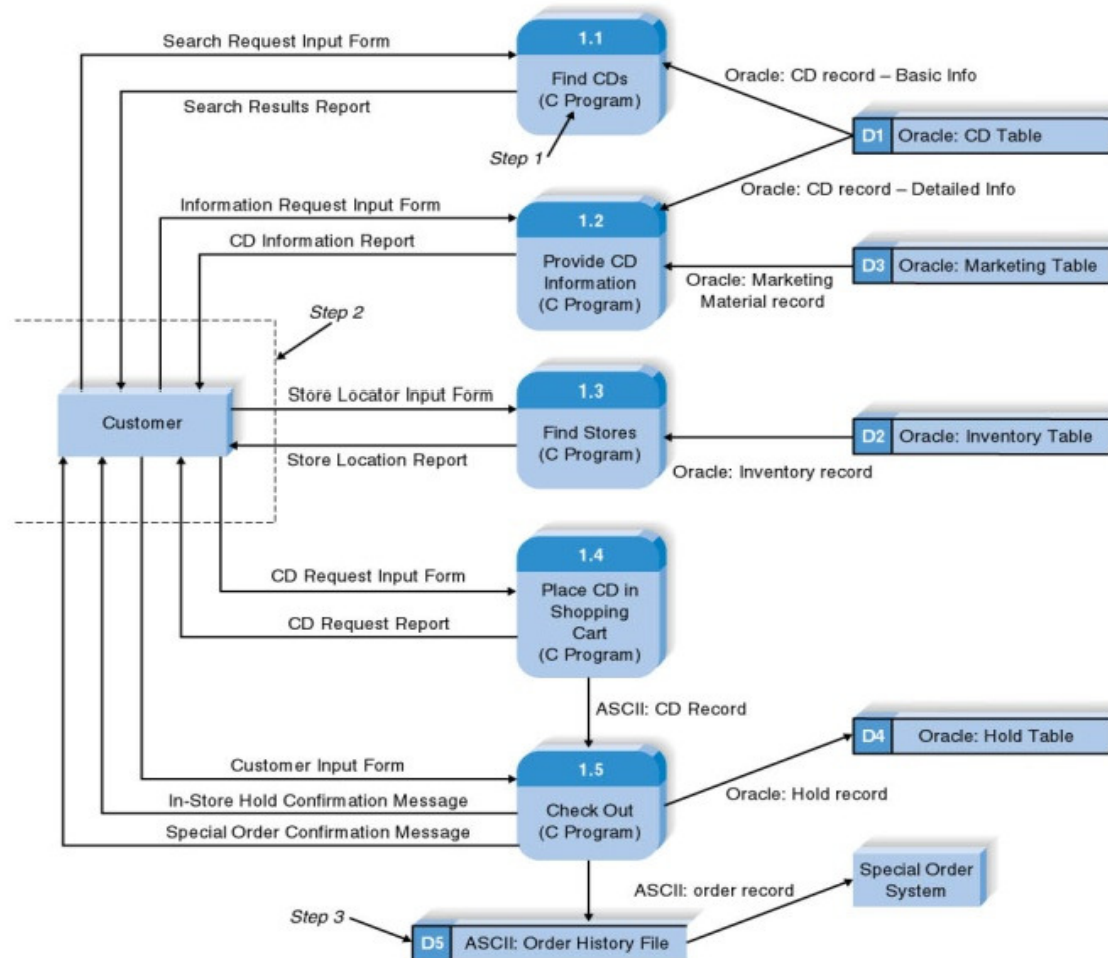
Dodavanje sistemskih procesa, podataka i tokova

- Npr. funkcionalnost sustava je takva da sustav treba poslati obavijest korisniku elektroničkim putem (e-mailom) bez potrebe da pamti tu poslanu obavijest u skladištu podataka. Međutim tehnički nedostatak sustava je da ne može osigurati 24-satnu vezu prema Internetu za slanje elektroničke pošte. Zbog toga je potrebno uvesti skladište podataka za obavijesti. Ovo skladište nije potrebno u funkcionalnim zahtjevima sustava nego je posljedica tehničkih svojstava sustava.

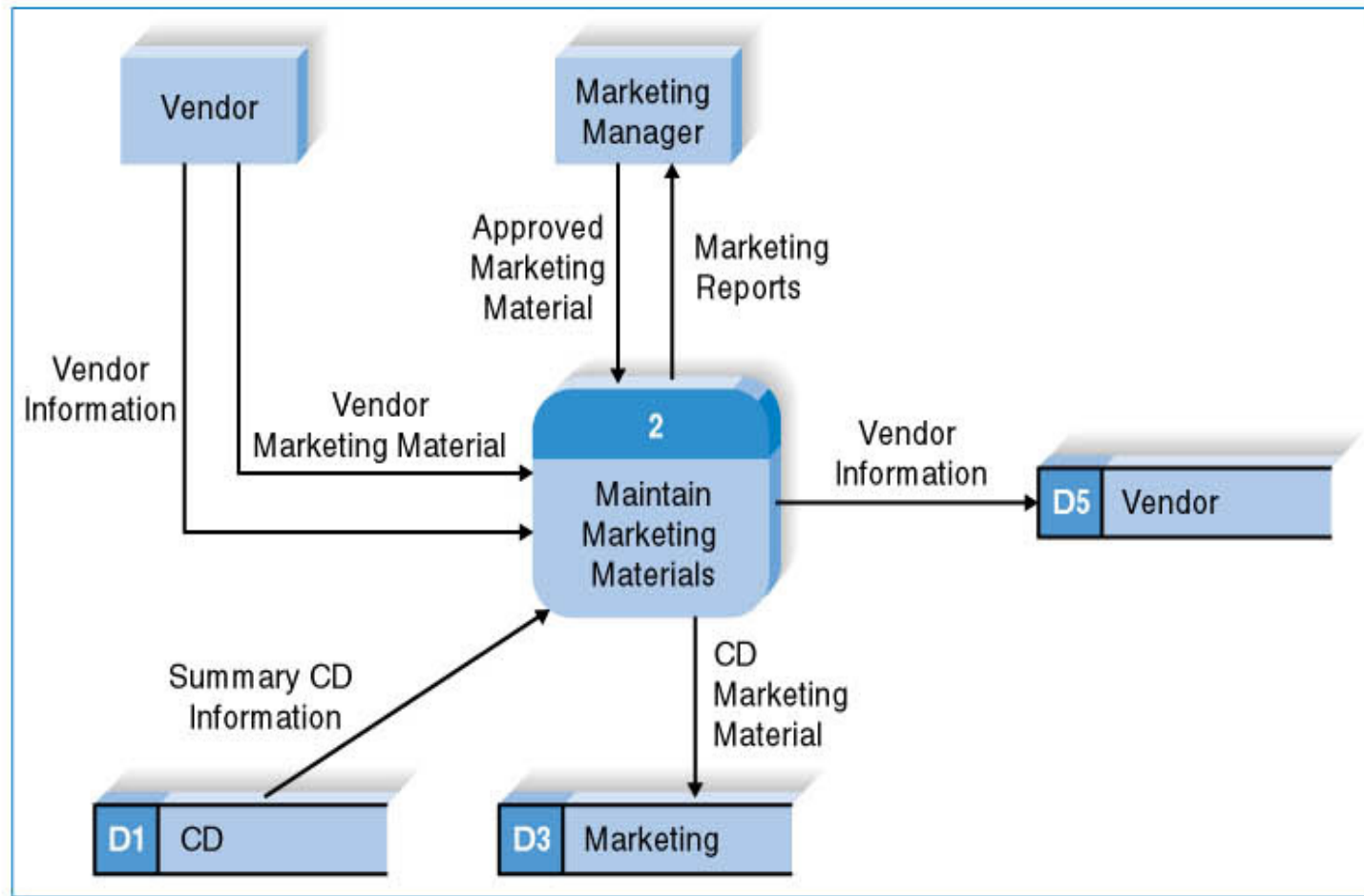
Ažuriranje podataka u tokovima

- Tokovi podataka bi trebali biti identični u logičkom i fizičkom DFD dijagramu.
- Ali fizički model procesa može zahtijevati dodatne podatke koji ne postoje u logičkom modelu.
- Npr. jedinstveni identifikator sloga u bazi, datum zadnjeg ažuriranje sloga u bazi,...

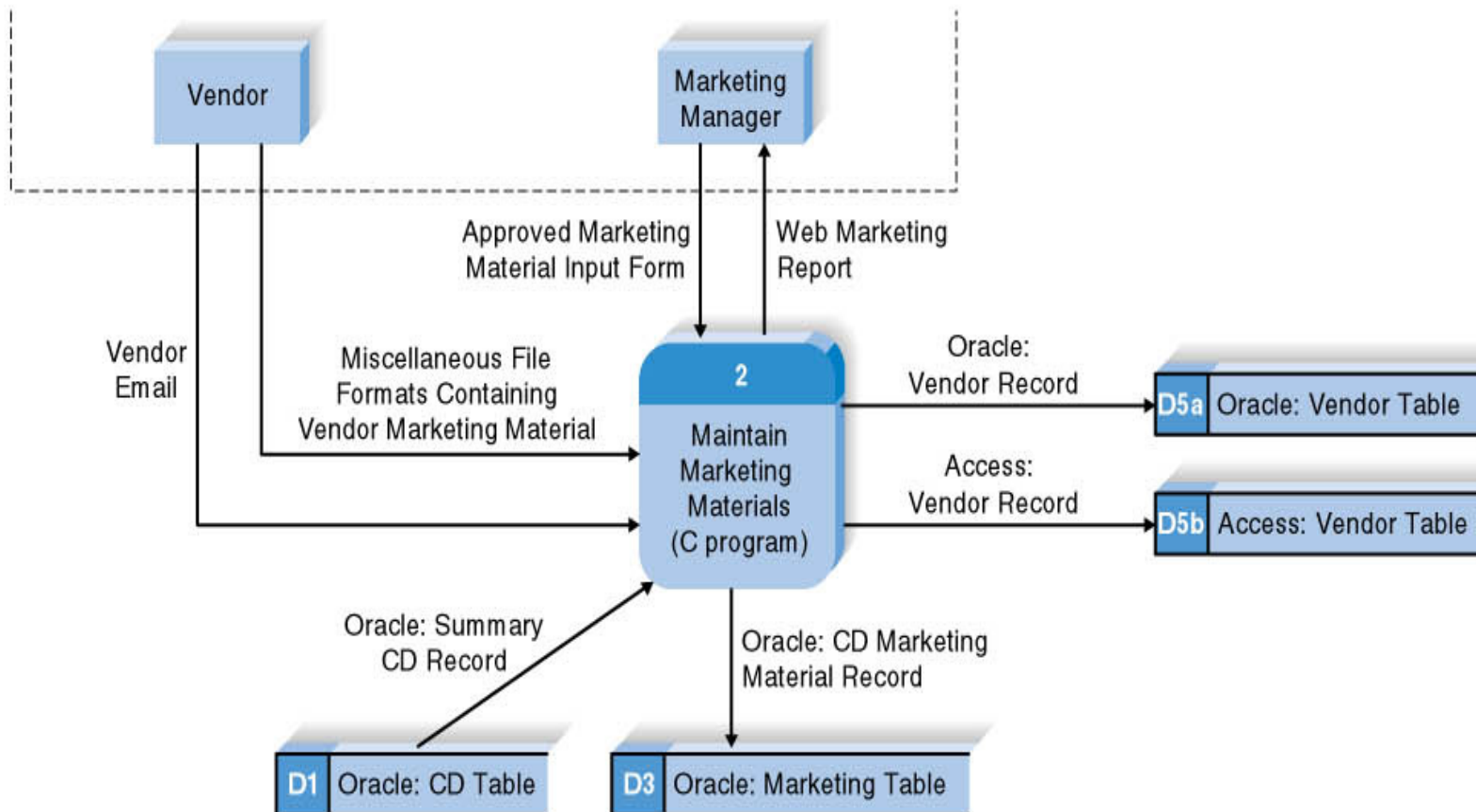
Fizički dijagram toka podataka



Logički dijagram toka podataka



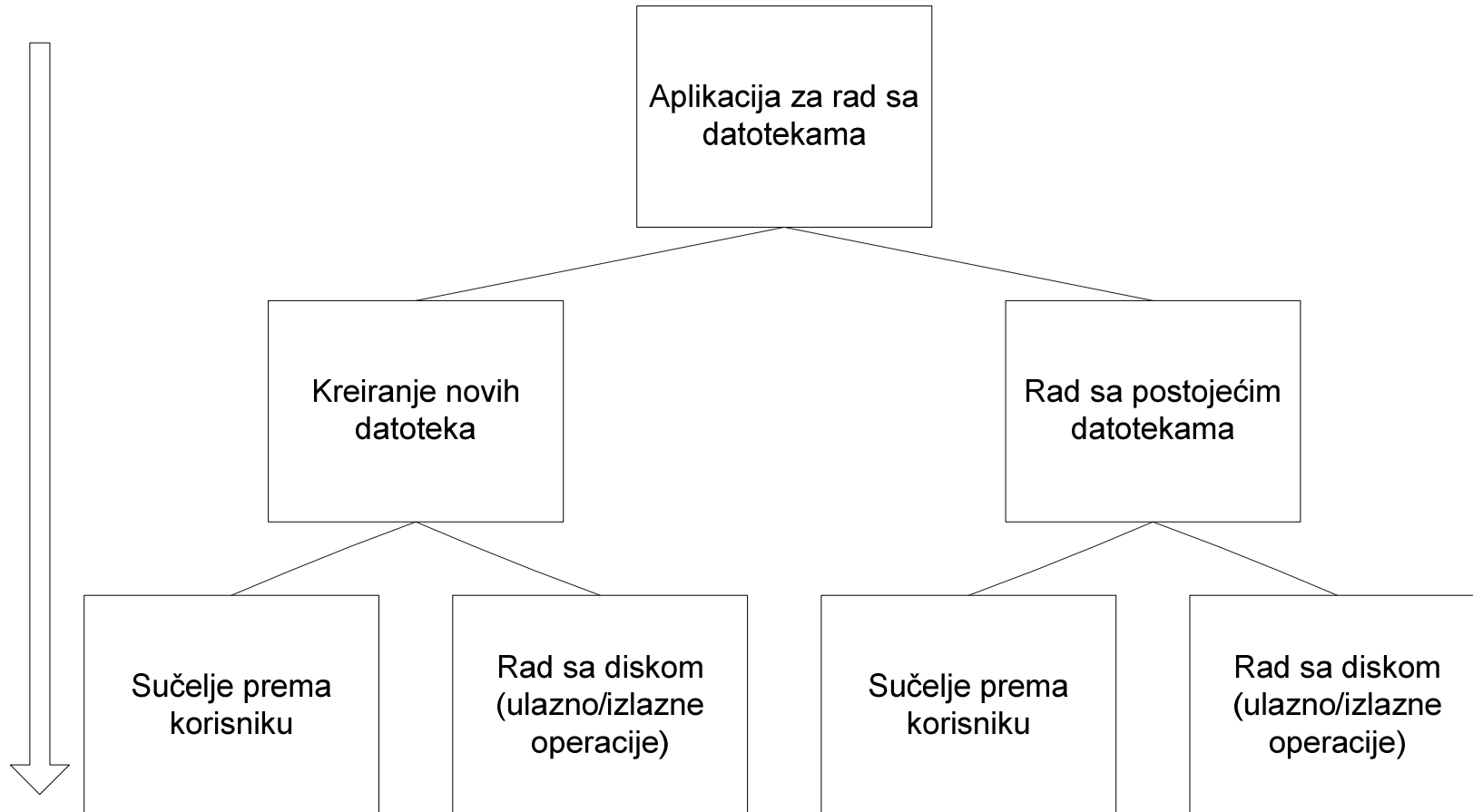
Fizički dijagram toka podataka



Dizajn programa

- Često korišteni način dizajna složenih programa je tzv. top-down (odozgo prema dole) pristup.
- Problem se razbija na manje probleme u svakom koraku sve dok se ne dobiju dovoljno jednostavni problemi čije rješavanje rješava početni složeni problem.
- Rezultat ovog pristupa je hijerarhijski sustav koji se može preslikati u modularnu strukturu kompatibilnu s imperativnom programskom paradigmom (rješenje problema je funkcija, procedura).

Top-down pristup



Bottom-up pristup

- Drugi pristup je *bottom-up* (odozdo prema gore) pristup kod kojeg se na početku identificiraju pojedinačni zadaci u sustavu i na koji način se rješenje pojedinačnih zadataka može koristiti u rješavanju složenijih problema u sustavu.
- Nekada je prevladavao *top-down* pristup, a danas se više koristi *bottom-up* pristup jer su pouzdaniji sustavi u kojima su moduli ravnopravniji od strogo hijerarhijskih sustava.
- Ovaj pristup je i bliži izradi softvera od gotovih, postojećih komponenti (*off-the-shelf components*) (npr. dizajnerski predlošci).

Moduli

- Modul se definira kao kôd (kolekcija programskih linija) koji izvršava određenu funkcionalnost.
- Moduli se mogu identificirati kroz programsku paradigmu tj. obrasce ili uzorke programiranja; to je samo jedan od načina definiranja modula softverskog sustava
- 4 osnovne programske paradigme:
 - imperativna (ili proceduralna) (C, Fortran, Pascal, ...) – npr. modularnost softverskog sustava se identificira na osnovu različitih zadataka koje sustav treba napraviti → tj. funkcija koje se trebaju programirati
 - objektno-orijentirana (C++, Java, ...) – npr. moduli se promatraju kao klase (ili objekti) određene funkcionalnosti sustava
 - funkcionalna (Lisp, Haskell,)
 - deklarativna (Prolog,....)

Dijagram strukture programa

- Dijagram strukture (*structure chart*) programa prikazuje sve module koji trebaju biti uključeni u program u hijerarhijskom formatu toka izvršavanja programa (gornji modul poziva module na nižoj razini).
- Hijerarhijski format uključuje:
 - sekvencu – redoslijed kojim se moduli pozivaju
 - selekciju – uvjeti pod kojima se modul poziva
 - iteraciju – koliko puta se modul ponovljeno poziva

Dijagram strukture programa

- Elementi dijagrama strukture su prikazani na slici.

Structure Chart Element	Purpose	Symbol
Every module: <ul style="list-style-type: none"> • Has a number • Has a name • Is a control module if it calls other modules below it • Is a subordinate module if it is controlled by a module at a higher level 	Denotes a logical piece of the program	
Every library module has: <ul style="list-style-type: none"> • A number • A name • Multiple instances within a diagram 	Denotes a logical piece of the program that is repeated within the structure chart	
A loop: <ul style="list-style-type: none"> • Is drawn using a curved arrow • Is placed around lines of one or more modules that are repeated 	Communicates that a module(s) is repeated	
A conditional line: <ul style="list-style-type: none"> • Is drawn using a diamond • Includes modules that are invoked based on some condition 	Communicates that subordinate modules are invoked by the control module based on some condition	
A data couple: <ul style="list-style-type: none"> • Contains an arrow • Contains an empty circle • Names the type of data that is being passed • Can be passed up or down • Has a direction that is denoted by the arrow 	Communicates that data is being passed from one module to another	
A control couple: <ul style="list-style-type: none"> • Contains an arrow • Contains a filled-in circle • Names the message or flag that is being passed • Should be passed up, not down • Has a direction that is denoted by the arrow 	Communicates that a message or a system flag is being passed from one module to another	
An off-page connector: <ul style="list-style-type: none"> • Is denoted by the hexagon • Has a title • Is used when the diagram is too large to fit everything on the same page 	Identifies when parts of the diagram are continued on another page of the structure chart	
An on-page connector: <ul style="list-style-type: none"> • Is denoted by the circle • Has a title • Is used when the diagram is too large to fit everything in the same spot on a page 	Identifies when parts of the diagram are continued somewhere else on the same page of the structure chart	

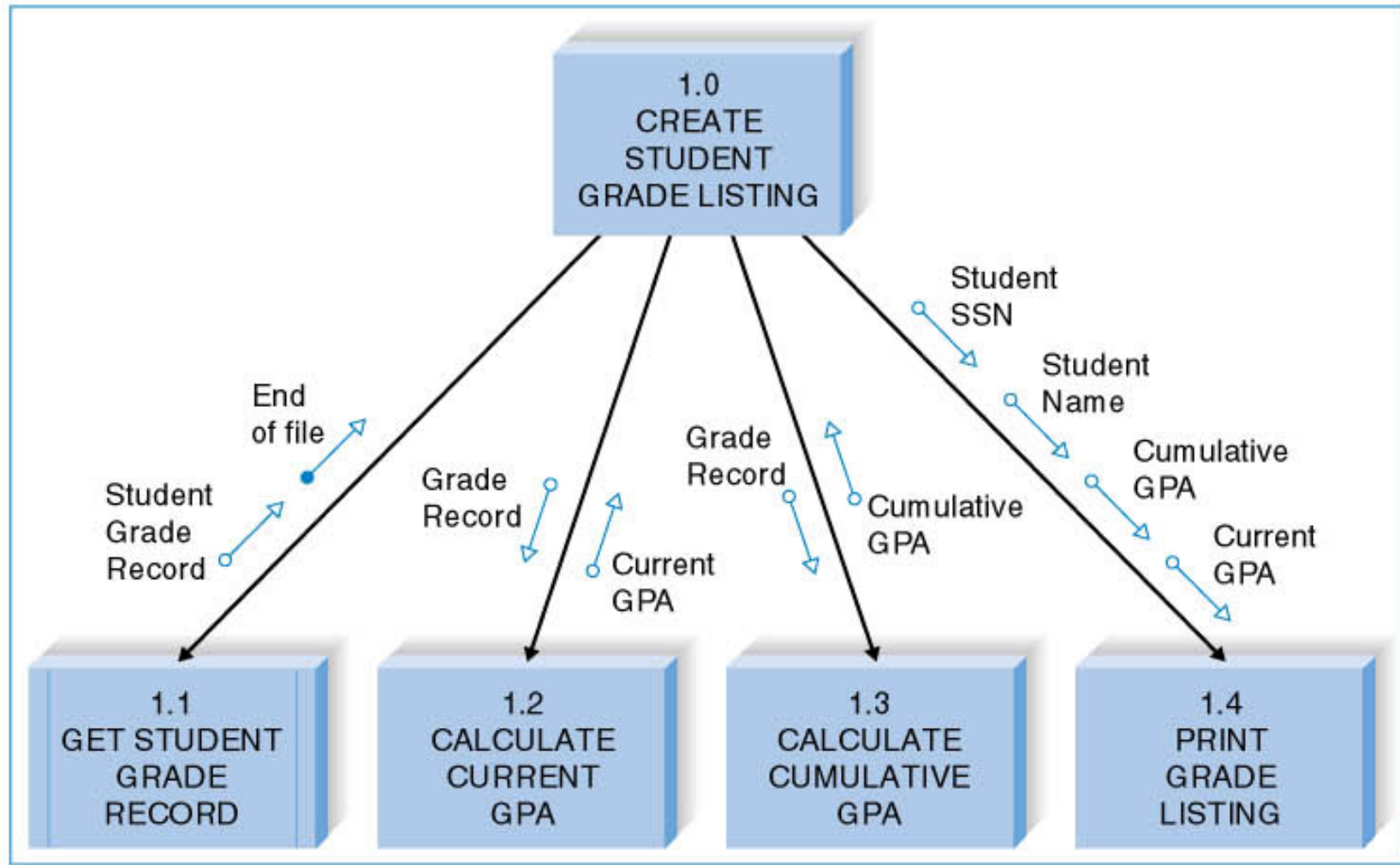
Dijagram strukture programa

- Modul se označava pravokutnikom.
- Linije koje povezuju module označavaju prenošenje kontrole izvršavanja aplikacije od jednog modula do drugog.
- Kontrolni modul (*control module*) je modul više razine koji sadrži logiku izvršavanja modula niže razine koji se nazivaju podređeni (*subordinate*) u odnosu na promatrani modul.
- Iteracija se označava zakrivljenom strelicom dok se selekcija označava sa rombom.

Dijagram strukture programa

- Modul koji se ponovno koristi (biblioteka) se označava sa dodatnim linijama na stranama pravokutnika.
- Konektor omogućava povezivanje više dijagrama strukture (*off-page* i *on-page*).
- Sprega modula definira povezanost modula. Kontrolna sprega (*control couple*) je kada jedan modul prenosi kontrolu odnosno slijed izvršavanja zadatka na drugi modul. Podatkovna sprega (*data couple*) je kada moduli prenose podatake.

Dijagram strukture programa



Izgradnja dijagrama strukture

- Izgradnja strukturnog dijagrama se odvija u četiri koraka:
 - Identifikacija modula i razina
 - Identifikacija selekcija i iteracija
 - Dodavanje sprege
 - Provjera strukturnog dijagrama
- Izgradnja dijagrama strukture počinje od DFD-a jer obično svaki proces u DFD-u odgovara jednom modulu u dijagramu strukture.

Izgradnja dijagrama strukture

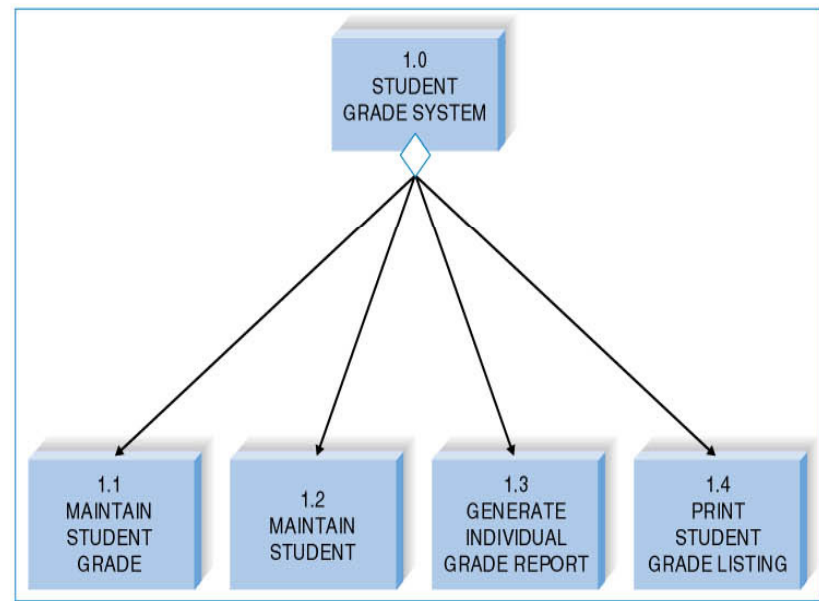
- Razinski DFD-ovi (kontekstni, razine 0,1...) se podudaraju s hijerarhijom dijagrama strukture pa tako glavni proces kontekstnog dijagrama odgovara glavnom kontrolnom modulu.
- Iz DFD dijagrama možemo lako identificirati module, kao i razinu na kojoj se nalaze u strukturalnom modelu, ali identifikacija hijerarhijskog toka izvršavanja programa (sekvence, iteracije i selekcije) nije eksplicitno definirana u modelu procesa. Ona se izvlači iz slučajeva korištenja i zahtjeva prema sustavu.

Izgradnja dijagrama strukture

- Razlikujemo tri osnovne vrste procesa u modelu procesa:
 1. Ulazni (*Afferent*) procesi – to su procesi koji pružaju ulazne podatke/informacije u sustav
 2. Centralni procesi – to su procesi koji obavljaju određene kritične funkcionalnosti u sustavu
 3. Izlazni (*Efferent*) procesi – to su procesi koji pružaju izlazne podatke/informacije iz sustava

Izgradnja dijagrama strukture

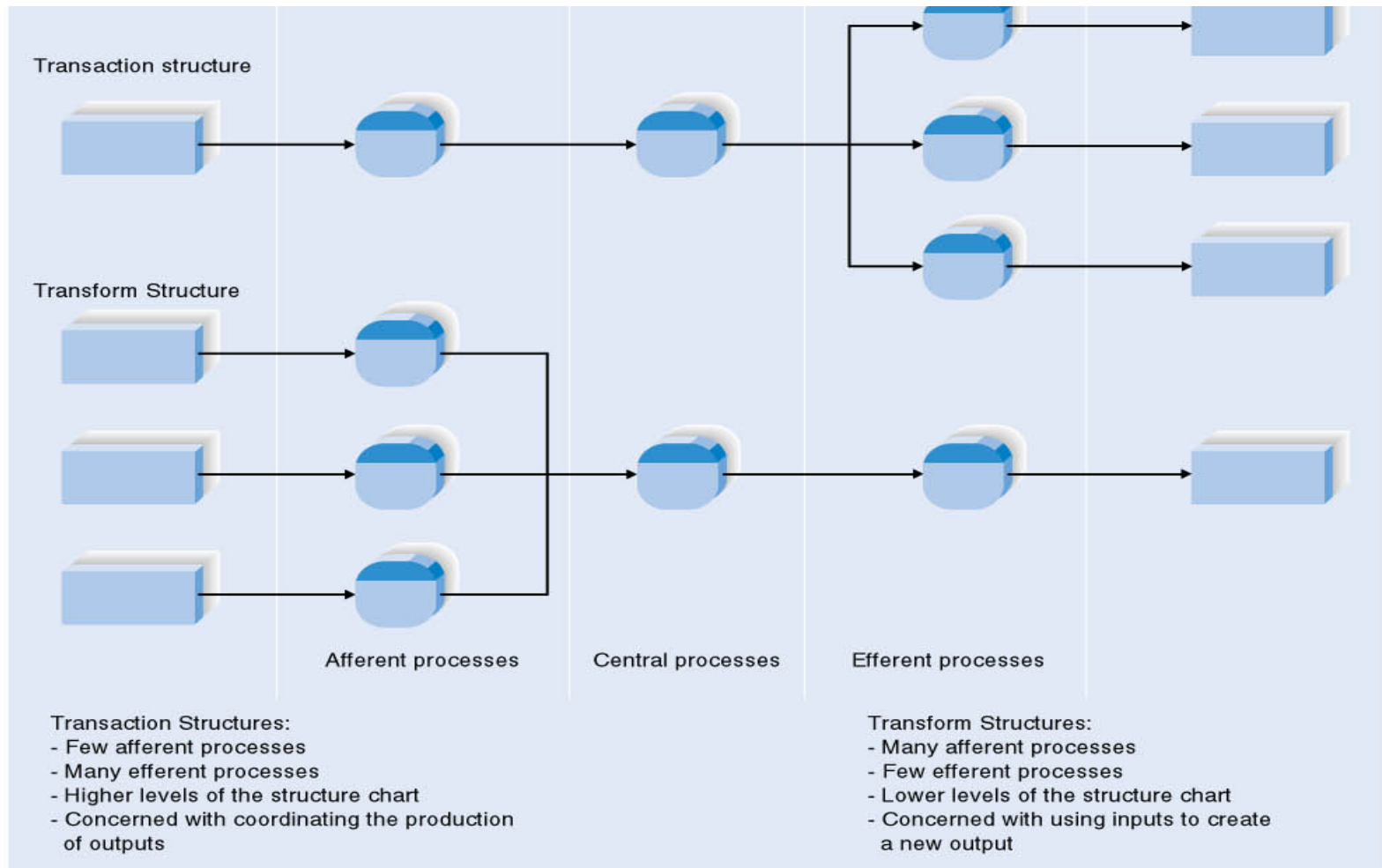
- Postoje dva osnovna tipa strukture u dijagramu struktura:
1. Transakcijska struktura se koristi kada svaki modul izvodi zasebnu transakciju. Obično ima malo ulaznih, a dosta izlaznih modula. Obično se nalazi na višim razinama dijagrama strukture. Npr. imamo modul koji sadrži izbornik i ovisno o korisnikovoj akciji preko izbornika se poziva modul koji ima specifičnu funkcionalnost.



Izgradnja dijagrama strukture

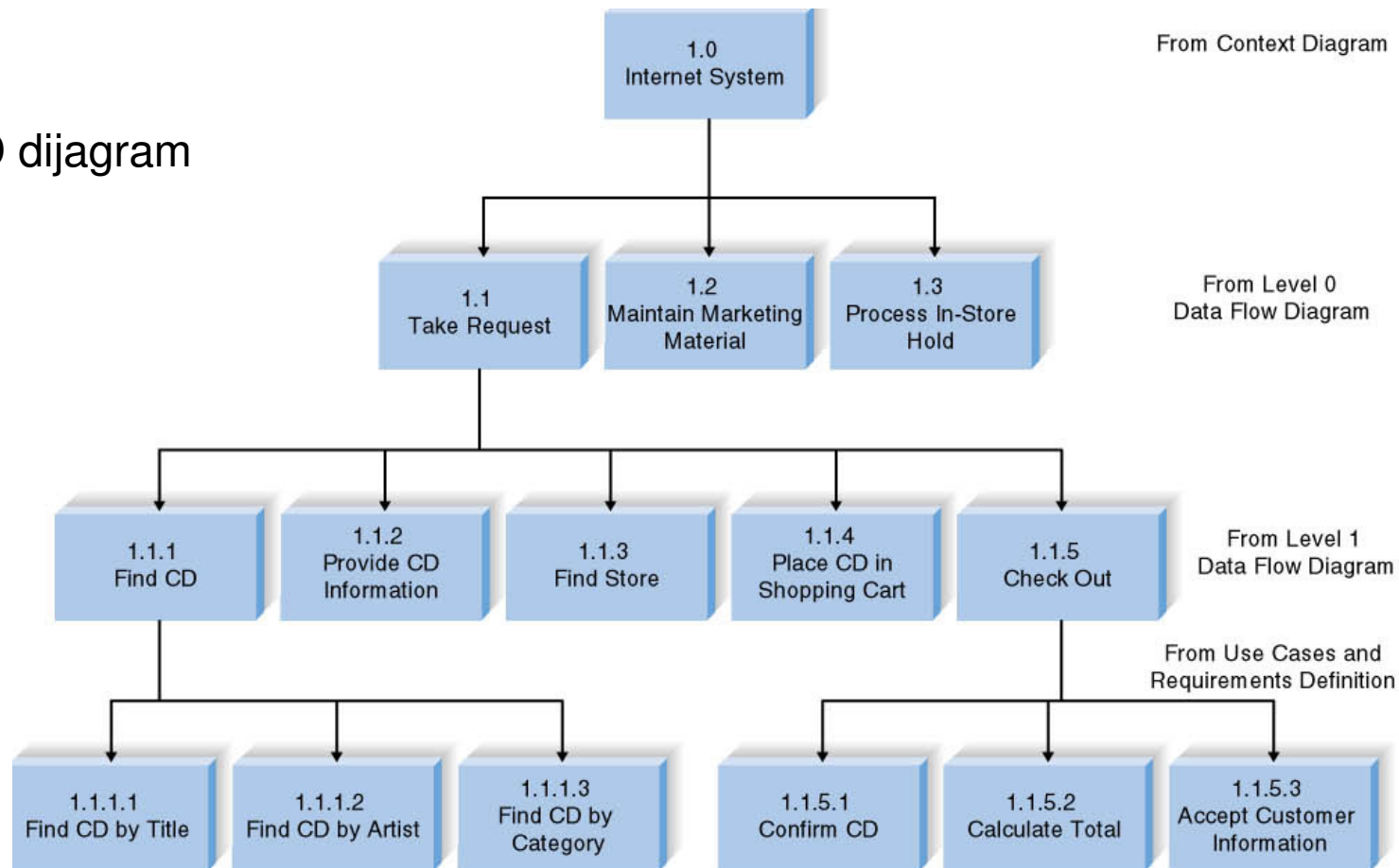
2. Transformacijska struktura se koristi kada su moduli povezani i zajedno čine neki proces koji transformira ulaz u izlaz. Obično ima dosta ulaznih procesa, a malo izlaznih. Obično se nalazi na nižim razinama dijagrama strukture. Npr. imamo slijed web formi koje moramo popuniti jednu za drugom, svaka web forma je razvijena kao jedan ASP.NET ili PHP modul.

Usporedba

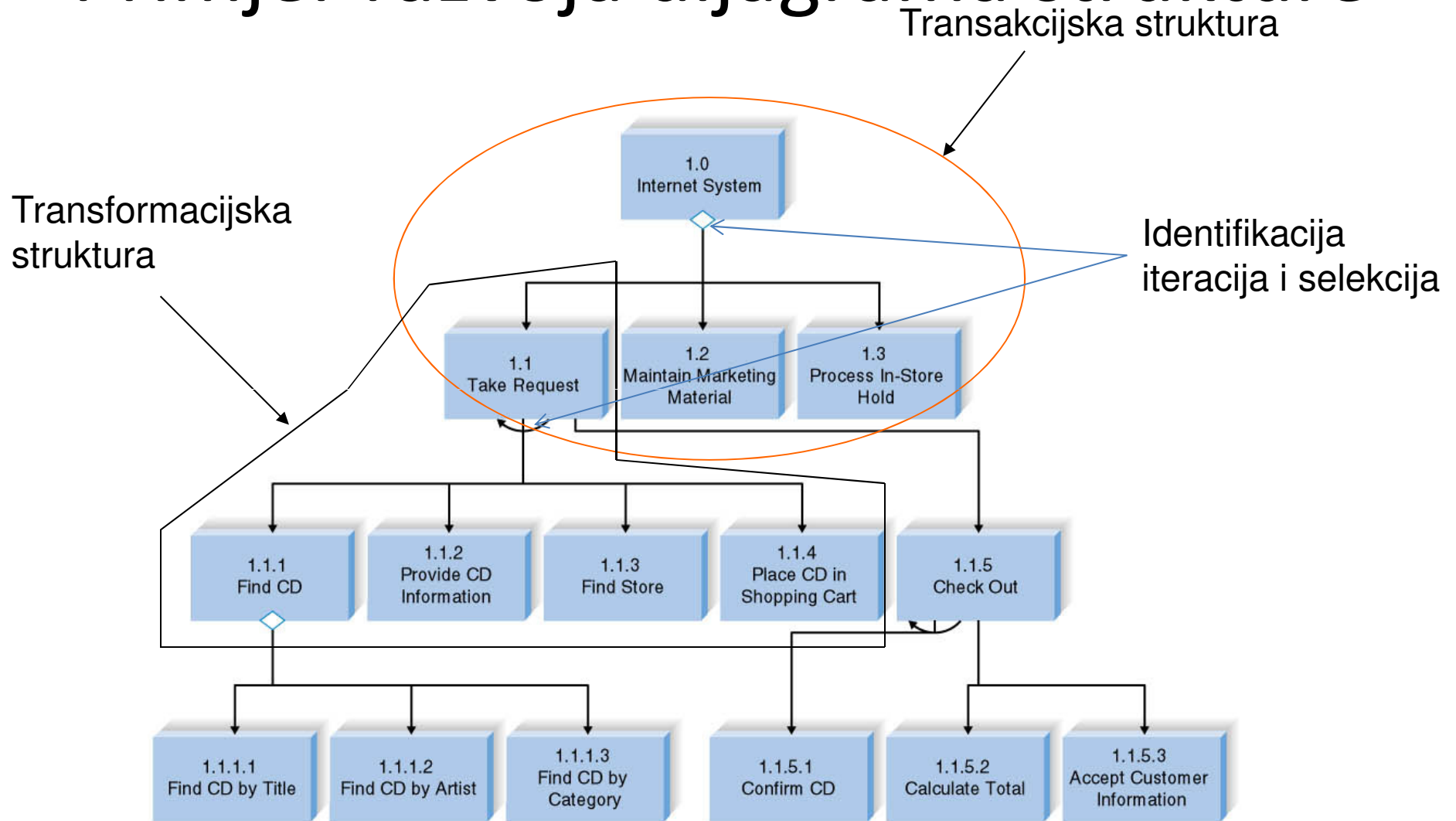


Primjer razvoja dijagrama strukture

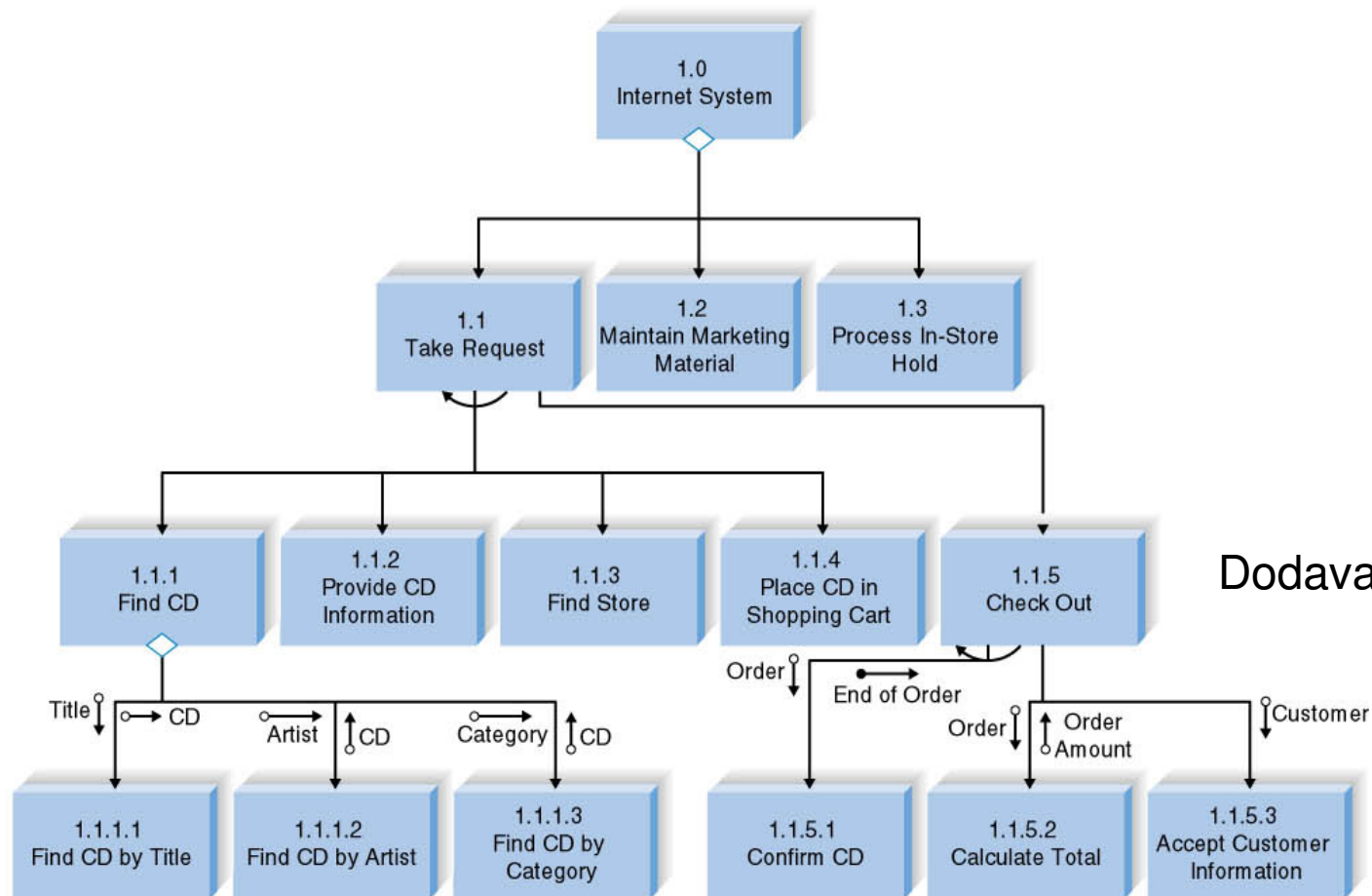
DFD dijagram



Primjer razvoja dijagrama strukture



Primjer razvoja dijagrama strukture



Dodavanje sprege

Smjernice

- Kvalitetni dijagrami strukture rezultiraju modularnim, ponovno upotrebljivim (*reusable*), i lako implementirajućim programima.
- Mjere dobrog dizajna uključuju:
 - Maksimalnu koheziju modula
 - Minimalnu spregu modula
 - Prikladne razine *fan-in* i *fan-out*

Kohezija modula

- Važno je da pojedinačni modul ne uključuje različitu funkcionalnost jer to otežava izmjene tog modula. Npr. da modul za dohvaćanje postavki korisnika ne uključuje obradu narudžbi.
- Slabiji oblik kohezije naziva se logička kohezija (*logical cohesion*) i odnosi se na sličnost aktivnosti koje izvode elementi tj. dijelovi unutar modula.
- Jači oblik kohezije naziva se funkcionalna kohezija (*functional cohesion*) u kojoj su svi dijelovi modula uključeni u istu aktivnost.

Kohezija modula

- Npr. kada dizajnirate klasu dobro je da metode klase imaju funkcionalnu koheziju tj. napraviti ćete posebnu metodu za postavljanje vrijednosti podatkovnog člana, a posebnu za dohvaćanje vrijednosti podatkovnog člana.

```
class primjer
{
    private int broj;
    public void postavi_broj(int pr)
    {
        broj = pr;
    }
    public int dohvati_broj()
    {
        return broj;
    }
}
```

Kohezija modula

- Npr. dijelovi metode `postavi_broj` imaju sličnu funkcionalnost (logička kohezija) jer obje aktivnosti u metodi su postavljanje vrijednosti dva podatkovna člana klase.

```
class primjer
{
    private int broj, drugi_broj;
    public void postavi_broj(int pr, int dr)
    {
        broj = pr;
        drugi_broj = dr;
    }
    public int dohvati_broj()
    {
        return broj;
    }
}
```

Sprega modula

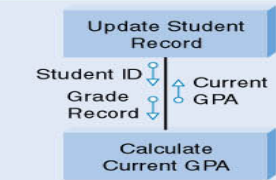
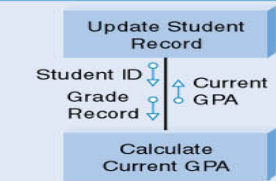

- Cilj dizajna je postići što veću neovisnost modula.
- Naravno moduli trebaju iskazivati određenu međusobnu ovisnost kako bi tvorili cjelinu.
- Sprega modula treba biti što vidljivija (eksplicitna).
- Implicitna sprega (*implicit coupling*) modula nije očigledna, a često uzrokuje greške u softveru.
- Tipičan primjer implicitne sprege su globalni podaci, većina programskih jezika više razine podržava primjenu globalnih podataka. Često se promjene na globalnim podacima dešavaju kao popratna pojava (*side effect*) izvođenja modula. Pojam *side effect* označava akciju koju izvodi softver, a koja nije očigledna iz koda.

Sprega modula

- Pored kontrolne sprege (*control couple*) kada jedan modul prenosi kontrolu odnosno slijed izvršavanja zadatka na drugi modul i podatkovne sprege (*data couple*) kada moduli prenose sadržaj podataka postoje i drugi oblici sprege koji se ne mogu identificirati iz dijagrama strukture, a poželjno ih je izbjegavati.

Sprega modula

- Tipovi sprega modula se dijele na:
- Eksplicitne
 - Podatkovnu spregu
 - Obilježnu (stamp) spregu
 - Kontrolnu spregu
- Implicitne:
 - Uobičajenu spregu
 - Sadržajnu spregu

TYPE		DEFINITION	EXAMPLE
Good ↓ Bad	Data	Modules pass fields of data or messages	 <p>All couples that are passed are used by the receiving module</p>
	Stamp	Modules pass record structures	 <p>The entire student record is not used by the receiving module, only the <i>student ID</i> field</p>
	Control	Module passes a piece of information that intends to control logic	 <p>The receiving module has to determine which GPA to calculate</p>
	Common	Modules refer to the same global data area	Typically, common coupling cannot be shown on the structure chart; it occurs when modules access the same data areas; and errors made in those areas can ripple through all the modules that use the data
	Content	Module refers to the inside of another module	<p>Module A: Update Student</p> <p>If student = new Then go to Module B</p> <p>Module B: Create Student</p> <p>At all costs avoid modules referring to each other in this way</p>

Fan-in

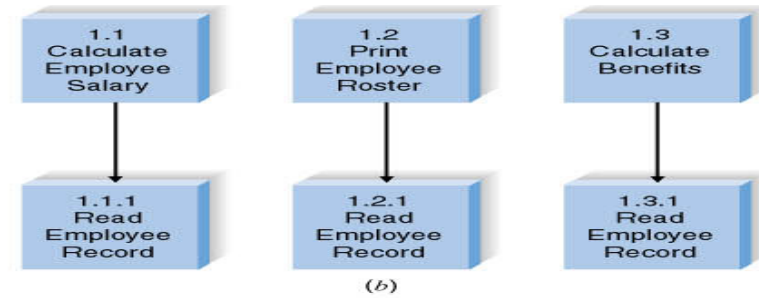
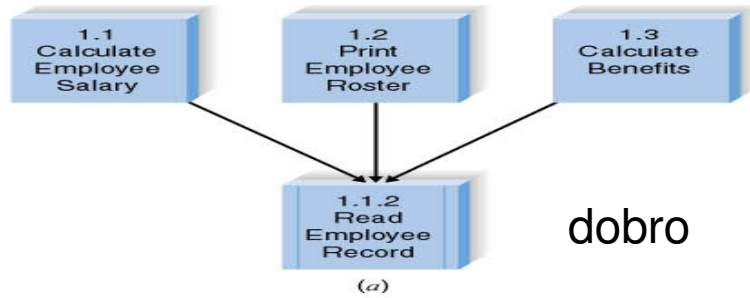
- Fan-in opisuje broj kontrolnih modula koji komuniciraju sa podređenim modulom.
- Modul sa visokim fan-in-om ima veliki broj kontrolnih modula koji ga pozivaju.
- To je poželjno jer znači da je jedan te isti modul korišten na puno mjesta u aplikaciji.
- Lakše je održavanje jer se izmjena na tom modulu odmah propagira u svim dijelovima aplikacije gdje se promatrani modul koristi.

Fan-out

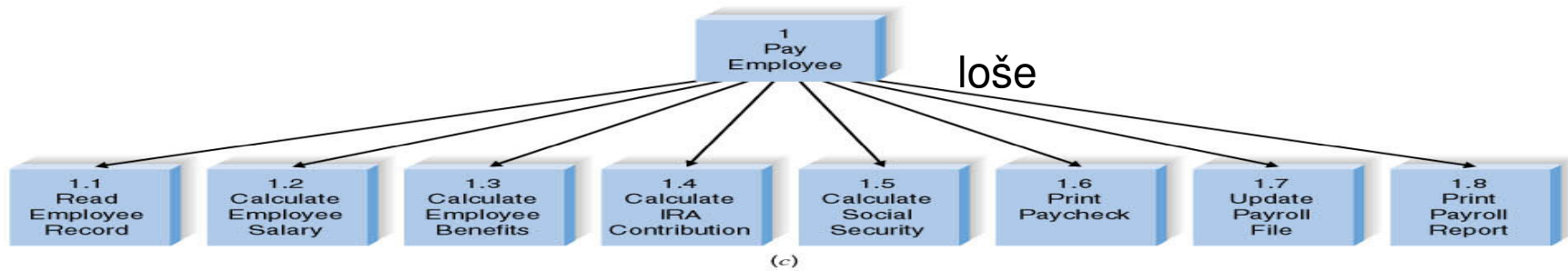
- Fan-out opisuje broj podređenih modula s kojima komunicira kontrolni modul.
- Poželjno je da za svaki modul fan-out bude što manji jer veliki broj podređenih modula ukazuje na to da je kontrolni modul izuzetno složen da bi mogao kontrolirati sve podređene module.
- Preporučeno je broj podređenih modula ograničiti na do 7 (osim kod transakcijske strukture).

Fan-in i Fan-out

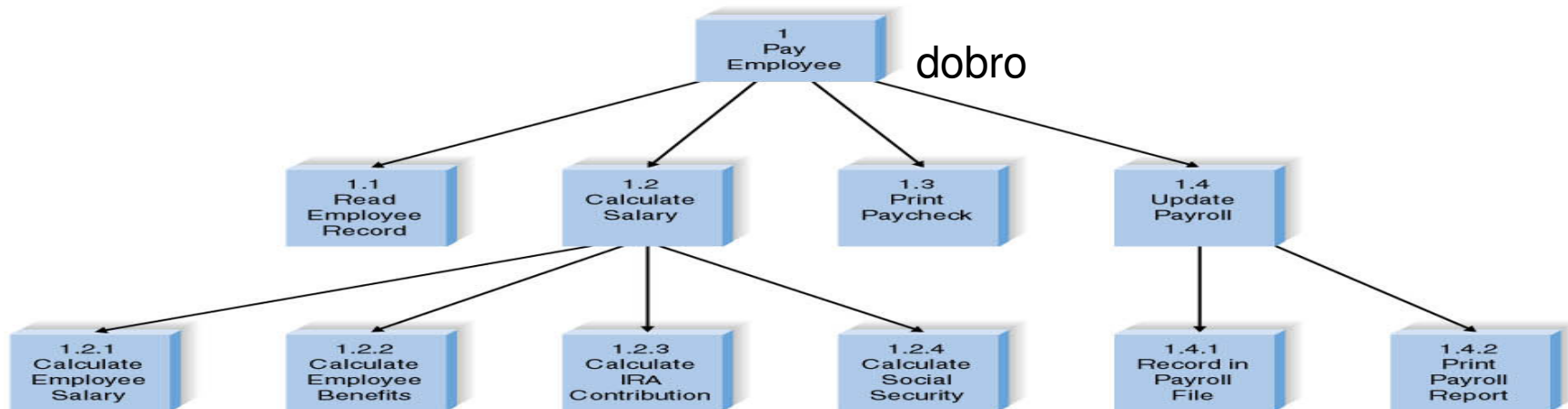
loše



loše



dobro



Programska specifikacija

- Moduli sa dijagrama strukture opisuju se detaljno pomoću programske specifikacije.
- To je pisani dokument koji sadrži eksplicitne instrukcije za pisanje dijelova kôda.
- Ne postoji formalna sintaksa, svaka organizacija koristi vlastiti format programske specifikacije.

Programska specifikacija

- Primjer forme za programsku specifikaciju. Obično sadrži četiri dijela:
 1. Osnovne informacije o modulu (ime, namjenu,...)
 2. Događaje koji triggeriraju promatrani modul.
 3. Ulaze i izlaze iz modula koji se identificiraju iz podatkovne i kontrolne sprege iz dijagrama strukture.
 4. Pseudokôd

Program Specification 1.1 for ABC System

Module _____
Name: _____
Purpose: _____
Programmer: _____
Date due: _____

C PowerScript COBOL Visual Basic

Events _____

Input Name	Type	Used By	Notes

Output Name	Type	Used By	Notes

Pseudocode _____

Other _____

Programska specifikacija

- Ne postoji standardna sintaksa pseudokôda.
- Obično se koristi strukturirani prirodni jezik.
- Pseudokôd je povezan uz strukturu programskog jezika za koji se piše pseudokôd.

```
(Get_CD_Info module)
  Accept (CD.Title) {Required}
  Accept (CD.Artist) {Required}
  Accept (CD.Category) {Required}
  Accept (CD.Length)
Return
```

Programska specifikacija

Program Specification 1.1.1.1 for Internet Sales System

Module _____

Name: Find_CD_by_Title

Purpose: Display basic CD information, using a title input by the user

Programmer: John Smith

Date due: April 26, 2006

☐ C ☒ PowerScript ☐ COBOL ☐ FORTRAN

Events _____

search by title pushbutton is clicked

search by title menu choice is selected

Input Name:	Type:	Provided by:	Notes:
CD title	String (50)	Program 1.1.1	
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

Output Name:	Type:	Used by	Notes:
CD id	String (10)	Program 1.1.1	
Not_found	Logical	Program 1.1.1	Used to communicate when CD is not found
_____	_____	_____	_____

Pseudocode _____

(Find_CD module)

Find CD id via the Title

If no CD is found

Set not_found True

Else

Set not_found False

End_If

Return

Other _____

Business rule: if a CD id is not found, the "CD of the week" will appear to the user

Note: a control couple containing a not_found flag should be added from 1.1.1.1 to 1.1.1 to instruct 1.1.1 to display a not found message to the user and the CD of the week