Informatika
Predavanje
Aplikativni softver

dr Ana Kovačević, vandr. prof.

kana@rcub.bg.ac.rs

Aplikativni (korisnički) programi

- Programi za:
  - obradu teksta
  - tabelarna izračunavanja
  - projektovanje i crtanje
  - za obradu multimedijalnih podatka
- Sistemi za upravljanje bazama podataka
- Video i kompjuterske igre
- Zlonamerni programi....

Aplikativni programi

- Programiranje je neophodno kada ne postoje gotov softver za određeni zadatak.
- Neki zadaci su pogodni za automatsku obradu:
  - Ponavljanje
  - Rad sa digitalnim podacima
  - Prate niz koraka
- Nije pogodno za automatsku obradu:
  - kreativnost i
  - “dodir ljudskosti”.

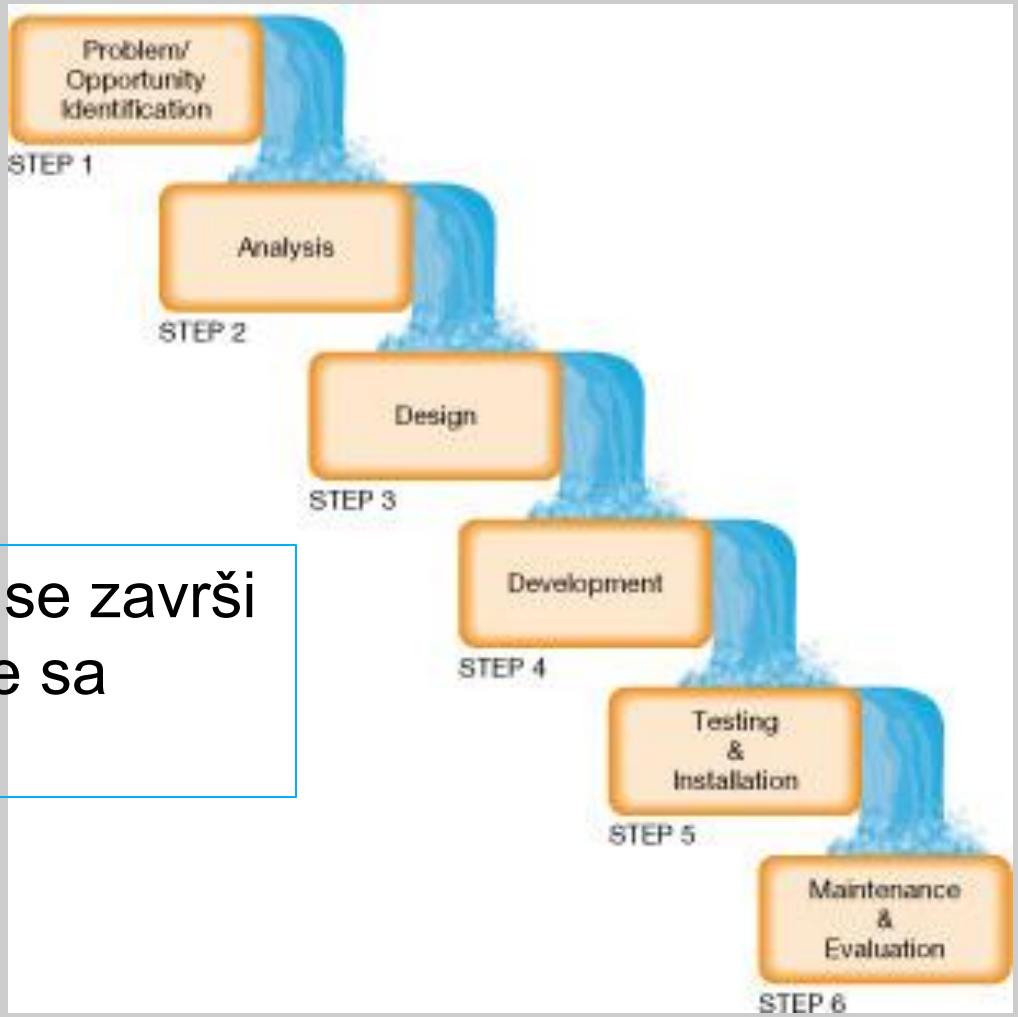
Životni ciklus Informacionog sistema

- Sistem: skup delova koji rade zajedno da bi se ostvario zajednički cilj.
- Informacioni sistem pomaže u planiranju i odlučivanju i obuhvata:
  - Podatke
  - Ljude
  - Procedure
  - Hardver
  - Softver

Životni ciklus razvoja sistema

- Za razvoj modernog softvera, potreban je ceo tim.
- Potrebno je da bude organizovan proces.
- Potrebno je da radi na više operativnih sistema i preko mreže.
- Potrebno je da bude “bez grešaka” i dobro podržano.

Šest koraka u životnom ciklusu razvoja **SISTEMA**



Svaki korak mora da se završi pre nego što se krene sa sledećim

1. Indentifikovanje problema i mogućnosti

PRIMER:

- Korporacije pokušavaju da uđu na nova tržišta i kreiraju nove proizvode.
- Korporacije opslužuju postojeće kupce
  - Veća efikasnost
  - Odgovaraju na problem
- Odlučuje se koji projekti da napreduju zasnovano na dostupnim resurima

2.Analiza

- Istražuje se problem koji treba rešiti
- Razvoj specifikacija programa (ciljevi i ishodi)
- Procena izvodljivosti
- Korisnički zahtevi se izvršavanju
- Analitičari preporučuju plan akcije.

3. Dizajn

- Generiše se detaljan plan za programere
- Dijagram toka (flowchart) i dijagram toka podataka se koriste za predstavljeni sistem
  - Dijagram toka je vizualni dijagram procesa
  - Dijagram toka podataka prati podatke od tačke ulaska u sistem do krajnjeg odredišta.
- Kada je sistem dizajniran, evaluira se postojeći softver i odlučuje da li je potrebno da se razvije novi deo softvera ili može da kupi/preuzme neki već dostupan i prilagodi potrebama.

4. Razvoj

- Programiranje
- **Prvi deo razvoja životnog ciklusa programa**
- Dokumentacija

5. Testiranje i instalacija

- Programi se testiraju da se obezbedi da ispravno rade
- Programi su instalirani za poslovnu primenu.

6. Održavanje i evaluacija

- Performanse sistema se prate
- Korekcije i modifikacije programa se prave
- Dodatni zahtevi koje korisnici prave se procenjuju.
- Odgovarajuće modfikacije programa se prave, i novi zahtevi za dodatne opcije.

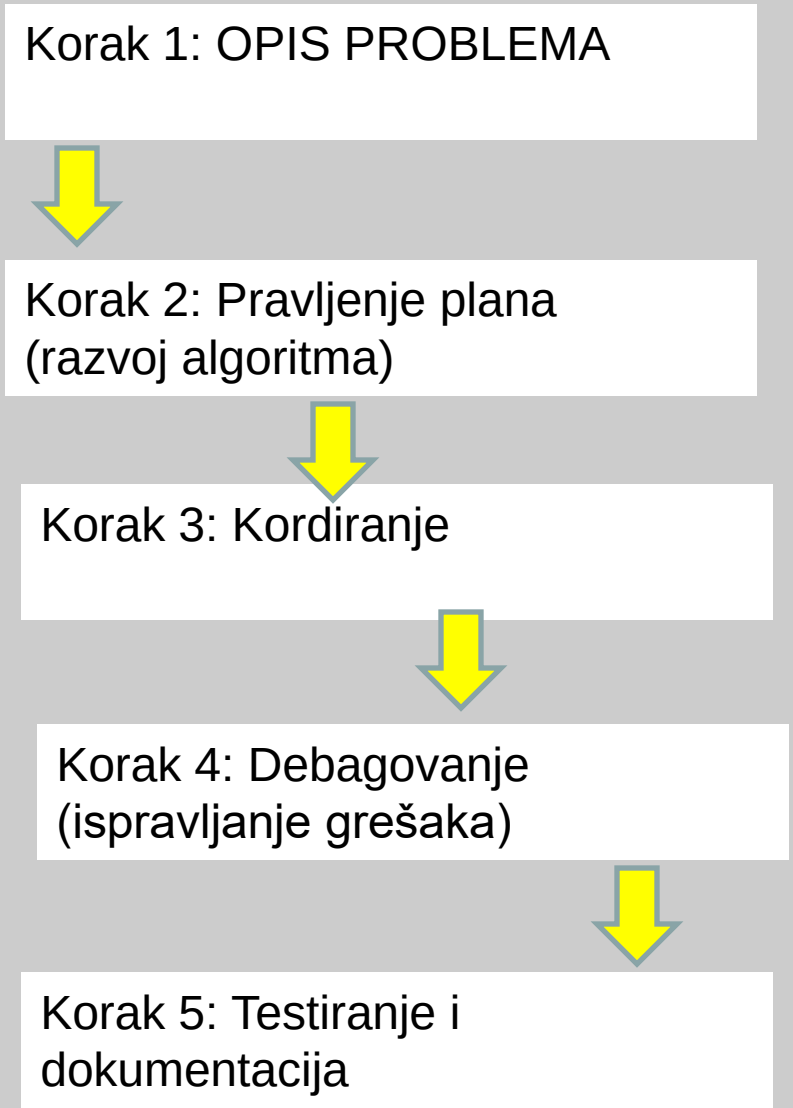
ŽIVOTNI CIKLUS PROGRAMIRANJA

autor: dr Ana Kovačević, FB

Životni ciklu programiranja

- Programiranje
  - Proces prevođenja zadataka u niz komandi tako da računar može da izvrši te zadatke
  - Uključuje identifikovanje koje delove zadatka kompjuter će izvoditi
  - Opisuje zadatke potpuno i kompletnim postupkom.
  - Prevodi ovaj opis u jezik koji razume kompjuterski CPU.

Životni ciklus razvoja **programa**



Korak 1: Opis problema

- Programeri prvo moraju da u potpunosti opišu problem
- Identifikovanje zadataka za automatizaciju
- Opis “ponašanja” sofvera.

Korak 2: Pravljenje plana (algoritma)

- Problem je preveden u niz specifičnih, sekvencijalnih koraka poznatih kao algoritam.
- Opisuju šta kompjuter mora da uradi da bi obavio posao (zadatak).
- Algoritam je pisan u prirodnom (natural) jeziku, kao npr. srpskom ili engleskom.

Korak 3: Kodiranje

- Algoritam se prevodi u programerske kodove
- Kod programa je razumljiviji od jezika CPU (0 i 1) ali je visoko struktuiran.
- Prilikom kodiranja, mora se voditi računa o operacijama koje samo CPU može da izvodi.

Korak 4: Debagovanje

- U procesu debagovanja ispravljaju se sve greške nađene u kodu.

Korak 5:
Testiranje i dokumentacija

- Softver testira:
  - Programerski tim
  - Korisnici programa
- Rezultat celokupnog projekta je dokumentovan za:
  - Korisnike
  - Razvojni tim.
- Korisnici se obučavaju da koriste program efikasno.

Opis problema

- Opis problema je neophodan:
  - Jer predstavlja početnu tačku programerskog posla.
  - Jasno opisuje zadatke koje kompjuterski program mora da završi.
  - Opisuje kako program će izvršiti zadatke i odgovoriti na neuobičajene situacije.
  - Pomaže da se bolje razumeju ciljevi programerskih napora.

Opis problema: (nastavak)

- Zadaci koje ponavljaju, sa elektronskim informacijama, i imaju niz jasnih koraka su dobri kandidati za kreiranje programa.
- Računari pored toga mogu da pomognu kod sofisticiranih problema koji je mogu podeliti u serije lako kompjuterizovanih zadataka.

Opis problema (nastavak)

- Kompjuterski programi NE MOGU:
  - Da se ponašaju INTUITIVNO
  - Budu spontano kreativni
  - “misle” kao ljudi
- **Kompjuteri samo prate instrukcije i algoritme.**

Opis problema (nastavak)

- Osnovni elementi koji se definišu prilikom opisa problema su:
  - **Podaci**: “Sirov” ulaz koji korisnici zadaju
  - **Informacije (izlaz)** Rezultati koje su potrebni korisnicima
  - **Metode:** proces kako program konvertuje *ulaz* u korektan *izlaz*.

Opis problema (nastavak)

- Programeri rešavaju problem lošeg *ulaza* preko obrade grešaka u procesu testiranja:
  - Definisati specifične ulazne brojeve očekivane od korisnika.
  - Izlistati određene izlazne brojeve koje će program vratiti.
  - Koristi ove liste da se oceni da li će program raditi.

Opis problema (nastavak)

- Plan testiranja ne pokriva svaki ulaz koji program može da dobije, umesto toga:
  - Raditi sa korisnicima da se identifikuju kategorije ulaza
  - Specifikovati vrste izlaza koje se generišu
  - Opisati kako će se upravljati (obrađivati) greške za svaku ulaznu kategoriju.

Opis problema (nastavak)

- Većina kompanija ima svoj format za dokumentovanje problema. Ipak, svi problemi uključuju iste osnovne komponente:
  - Ulaz: podaci koji se očekuju da se obezbede
  - Izlaz: informacije koje se očekuju da se dobiju
  - Obrada: pravila za transformisanje ulaza u izlaz,
  - Obrada grešaka (kako će program reagovati ako korisnik unese pogrešne podatke)
  - Plan testiranja

PRIMER: opis problema

Cilj programa	Obračun zarade		
Ulaz	Broja sati rada (≥0)		
Izlaz	Zarada(≥0)		
Proces	Zarada se obračunava: za prvih 8sati u kontinuitetu po 800 din, a svaki sledeći sat po 1100 din.		
Obrada grešaka	Ulazni podatak mora biti pozitivan realan broj. Ako je negativan broj ili neprihvatljiv karakter program će se zastaviti na tom polju dok se ne unese ponovo ispravan podatak.		
Plan testiranja	Ulaz	Izlaz	Napomene
	8	8*800	Testiranje pozitivne vrednosti
	3	3*800	Testiranje pozitivne vrednosti
	12	8*800+4*1100	Testiranje pozitivne vrednosti
	-5	Greška/Potrebno je da korisnik ponovo unese vrednost	Obrada greške
	M	Greška/Potrebno je da korisnik	Obrada greške

Opis problema: razvoj algoritma

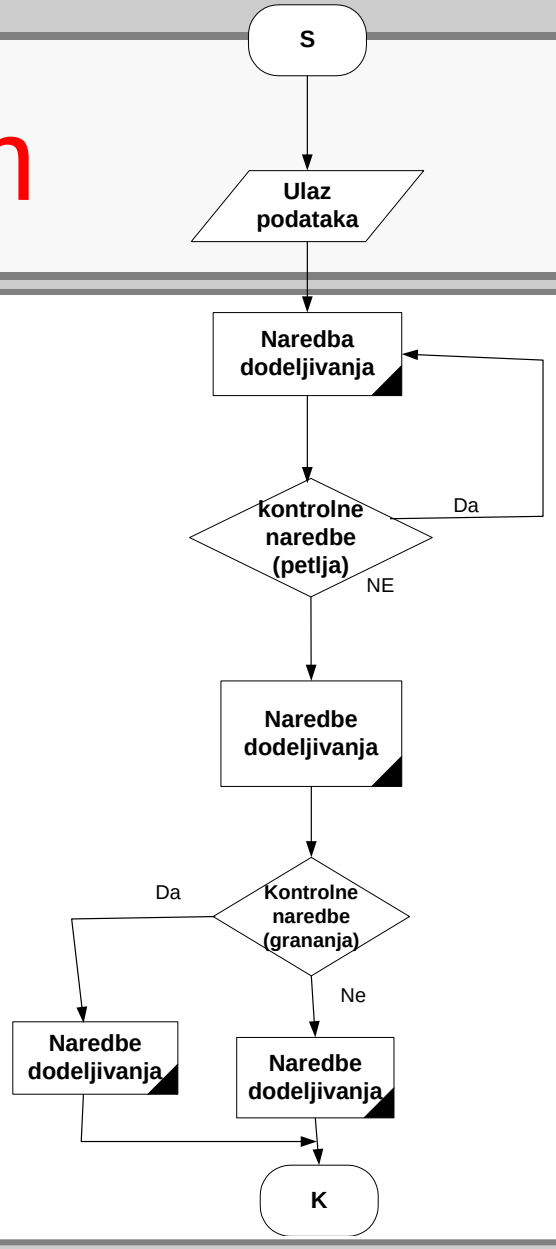
- Proces započinje razvojem detaljnog algoritma:
  - Skup specifičnih, sekvencijalnih koraka
  - Opis pomoću prirodnog jezika šta program mora da uradi.

Opis problema: razvoj algoritma

- Algoritmi:
  - Koriste se u svakodnevnom životu
  - Predstavlja specifični plan koji obuhvata sva moguća rešenja

Algoritam

- Generalni slučaj



Algoritam

- Primer:

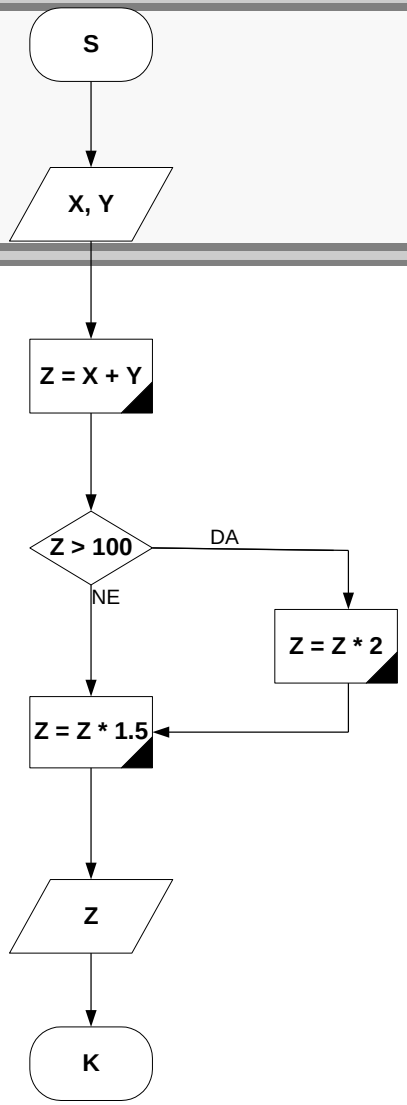
Sabrati brojeve X i Y.

Ako je zbir veći od 100

rezultat pomnožiti sa 2,

a ako je manji od 100

rezultat pomnožiti sa 1.5.

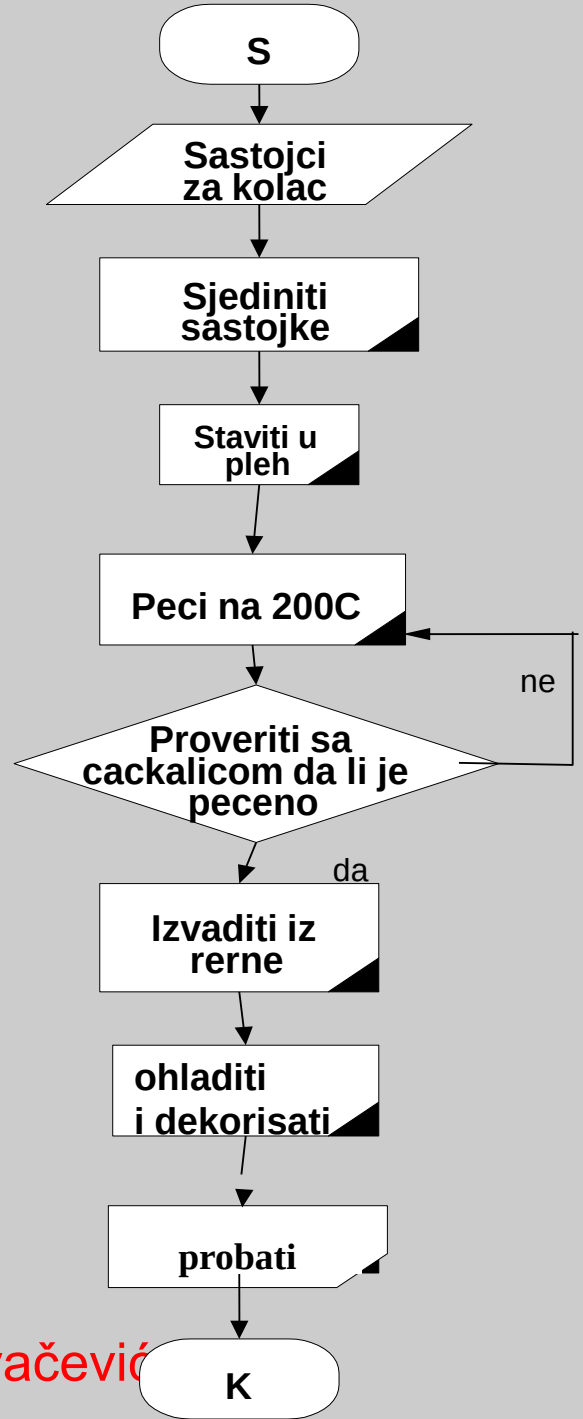


Primer Čokoladni kolač

- Sastojci: 4 štangle čokolade, 125 gr butera, 1 šoljica šećera, 3 jaja , 1 vanilin šećer, 1 kašika brašna
- Postupak: Pomešati čokoladu i puter. Dodati šećer u rastopljenu čokoladu, a potom jaja i vanilin šećer. Izmiksirati sa brašnom. Staviti u pleh i peći u zagrajanoj rerni na 200C oko 40 min. Proveriti da li je pečeno sa čačkalicom. Pošto se ohladi jesti.
- Deklarisane varijable: čokolada, jaja, puter, šećer…

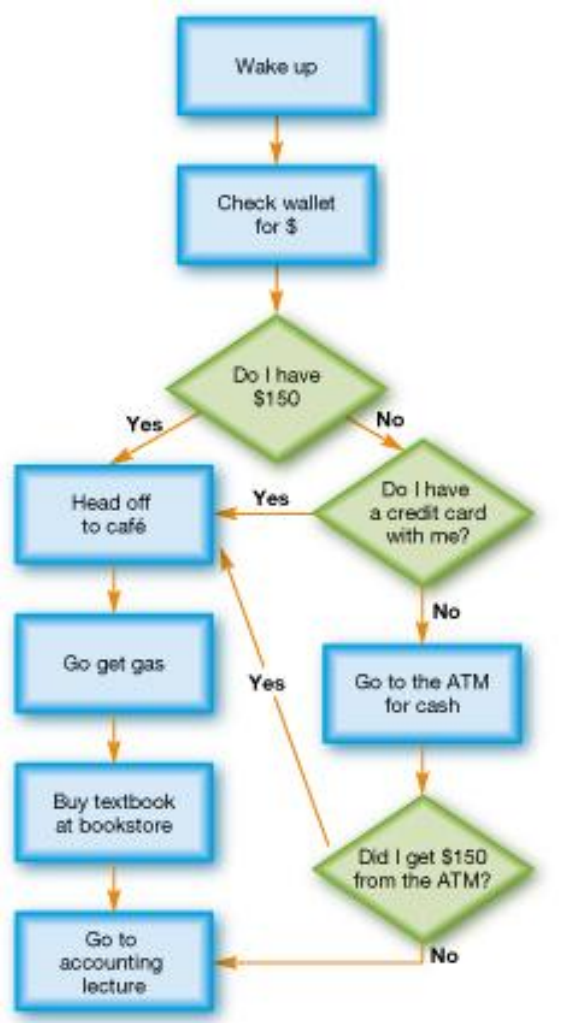
Algoritam:

čokoladni kolač



Prijatno!

Dijagram toka



Dijagram toka je vizualni dijagram procesa
Dijagram toka podataka prati podatka od tačke ulaska u sistem do krajnjeg odredišta.

Opis problema: razvoj algoritma

- Algoritmi imaju ograničenja
  - Ne mogu se svi problemi opisati kao fiksne sekvence predodređenih koraka
  - Neki uključuju slučajne, nepredvidljive događaje, ili ima puno uticaja koji nisu obuhvaćeni (predviđanje obveznica).

Opis problema: razvoj algoritma

- Programeri predstavljaju algoritam preko:
  - *Flowcharts (dijagram toka)* vizuelno predstavljanje paterna:
    - Specifični oblici simbola prikazuju ponašanje programa
    - MS Visio je popularan program za dijagram toka
  - *Pseudocode:* pristup baziran na tekstu
    - Rečima su opisani koraci (akcije) koje se izvršavaju u algoritmu.
    - Organizovan je kao struktura (različiti nivoi)

Razvoj algoritma: Odlučivanje i dizajn

- Programeri upravljaju kompleksnim algoritmima:
  - Korišćenjem liste izbora
  - Na osnovu tačaka odluke koje zavise od ulaznih vrednosti.

Razvoj algoritma: Odlučivanje i dizajn

- Postoji nekoliko vrsta tačaka odluke:
  - Binarna odluka:
  - Petlja
    - Postavlja se pitanje, ako je odgovor DA, izvodi se niz akacija, ako je odgovor NE ide se na korak koji sledi petlju.

Razvoj algoritma: Odlučivanje i dizajn

- Postoje tri elementa petlje:
  - Početna ili inicijalna vrednost (podrazumevana vrednost, npr. ukupna vrednost plate ima inicijalnu vrednost 0)
  - Skup akcija koje se izvode
  - Test uslov koji proverava da li je petlja završena.

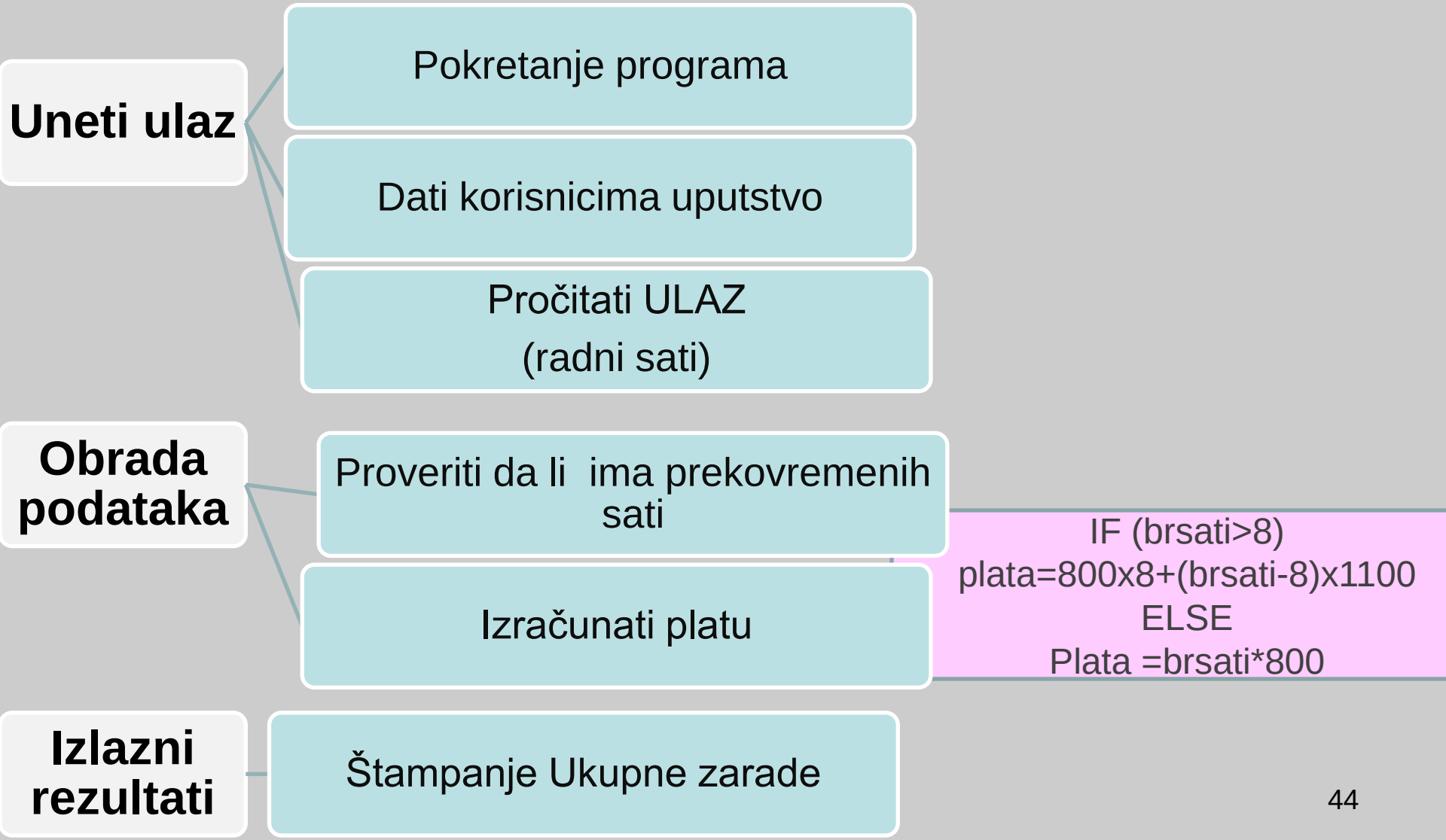
Razvoj algoritma: Odlučivanje i dizajn

- Kolmplikovano je ljudima da rešavaju probleme visoko stuktuirano, detaljnim algoritmima koji su potrebni za računare. Zato postoji nekoliko metodologija koje podržavaju programere korišćenjem:
  - Top-down dizajna
  - Objektno-orijentisane analize

Top-Down dizajn

- Top-down pristup je u kome su problemi podeljeni u niz zadataka visokog-nivoa.
- U ovom pristupu, programeri raščlanjuju zadatke u niz detaljnijih pod zadataka do momenta kada se mogu kodirati programskim jezikom.

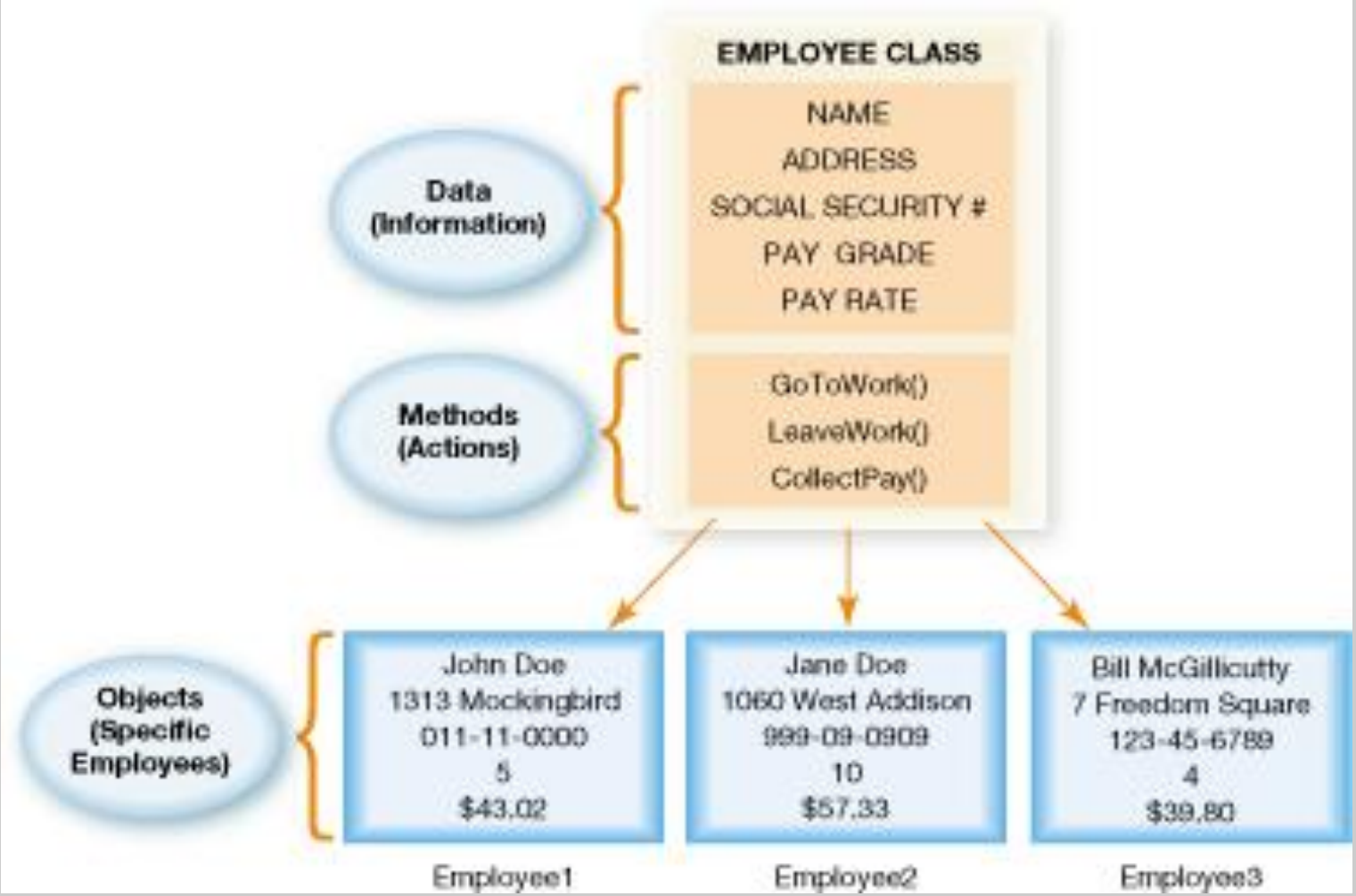
Primer: top-down pristupa



Objektno-orijentisana analiza

- Programeri identifikuju kategorije ulaza:
  - Klase (kategorije ulaza) su identifikovane
  - Klase su definisane pomoću informacija (podatka) i akcija (metoda ili ponašanja)
  - Algoritam omogućava objektima da stupaju u interakciju.

Primer: objektno-orijentisana analiza.



U objektno-orijentisanoj analizi
šta od sledećeg predstavlja “metod” za zaposlenog?

1. JMBG()
2. Izračunajkoeficijent()
3. Izračunajplatu()
4. Oba 2 & 3

Objektno-orijentisana analiza (nastavak)

- Programer koristi prednosti ponovnog korišćenja:
  - Hijerarhije mogu da se kreiraju brzo
  - Nasleđivanje – nova klasa preuzima podatke i metode od postojeće klase
    - Proširuje se i prilagođava sopstvenim potrebama
  - Originalna klasa se naziva osnovna klasa
  - Nova modifikovana klasa, je izvedena klasa.

Kodiranje

- Kodiranje prevođenje na jezik računara
- Ideje se prevode u CPU instrukcije
  - Izbor najboljeg programskog jezika za dati slučaj.
  - Kodiranje: prevođenje algoritma u programski jezik
  - Veoma precizan format korišćenjem ključnih reči, ali konzistentne strukture.

Kodiranje

- Kada programeri imaju algoritam, određuju ključne delove informacija koje koristi algoritam da napravi odluke:
  - Koji koraci su potrebni za proračun nove informacije?
  - Da ima tačaka kada je potrebno odlučivanje_
  - Koje vrste odluka se donose?
  - Da li ima ponavljanja određenih koraka?
- Kada se odrede potrebne informacije, i tok kako će se menjati svaki korak algoritma, moguće je konvertovati algoritam u specifični programski jezik.

Kodiranje

- Programeri transformišu od algoritma do koda pomoću:
  - Identifikovanja ključnih delova informacije
  - Identifikovanja toka svakog koraka
  - Konvertovanja algoritma u kompjuterski kod

Kodiranje

- Postoje razne vrste programskih jezika:
  - Klasifikovani u glavne grupe nazivaju se generacije.
  - Prvi jezici (mašinski i asembler) su zahtevali da programer zna kako je konstruisan kompjuter i kako su sačuvani podaci.
  - Programiranje je jednostavnije sada kad jezici su bliži ljudskom razmišljanju.

Kodiranje

- Evolucija morednih programskih jezika:
  - Prve generacije (1GL): Aktuelni mašinski jezik – sekvence bitova CPU koje razume.
  - Druge generacije (2GL): Poznat kao Asembler.
  - Treće generacije (3GL): Koristi simbole i komande da olakša programerima da zadaju komandu računarima, primeri BASIC; FORTRAN, COBOL, C/C++, Java.

Kodiranje

- Četvrte generacije (4GL): jezici za upite baze podataka i generisanje izveštaja. Npr. SQL
- Pete generacije (5GL): Najprirodniji, problemi se predstavljaju kao serija činjenica, primer PROLOG.

Mašinski jezik

- Programi na mašinskom nivou – instrukcije direktno kodirane.
- Skup mogućih instrukcija zavisi od unutrašnje arhitekture svakog procesora.
- Primer: procesor Intel 8086, kod operacije sabiranja – 00010011.
- Pisanje programa je težak i spor posao, podložan greškama.

Simboličko programiranje

- Simboličko programiranje – svaka kodirana instrukcija se zamenjuje mnemoničkim tekstualnim kodom koji predstavlja skraćeno ime instrukcije
- Primer: Sabiranje dva broja Z = X + Y;
  1. MOV A, X; <A> ← <X>
  2. MOV B,Y; <B> ← <Y>
  3. ADD A,B; <A> ← <A> + <B>
  4. MOV Z,A; <Z> ← <A>

Z = X + Y

NA MAŠINSKOM jeziku:

	kod operacije	operand (adresa u memoriji)
1.	1010 0001	1000 0001
2.	1010 0010	1000 0010
3.	0001 0011	
4.	1010 0100	1000 0011

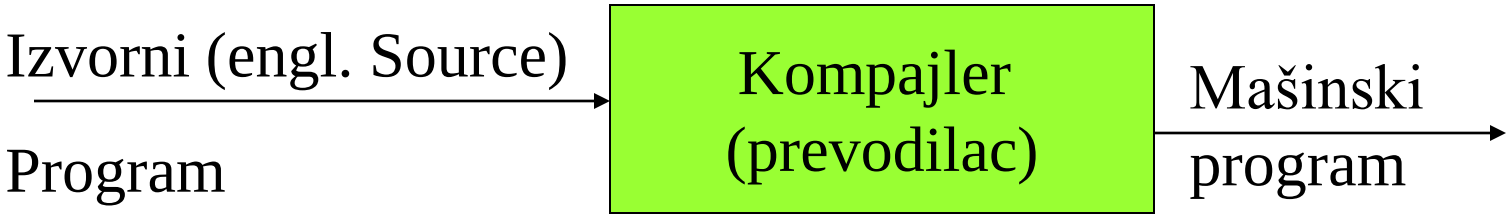
NA simboličkom jeziku:

```
1. MOV   A,X    ; <A> ← <X>
2. MOV   B,Y    ; <B> ← <Y>
3. ADD   A,B    ; <A> ← <A> + <B>
4. MOV   Z,A    ; <Z> ← <A>
```

Programski jezici višeg nivoa

- Nedostatak mašinskog i simboličkog jezika - svaki korak treba specificirati posebnom naredbom.
- Sa napretkom računarske tehnologije pojavili su se i programski jezici višeg nivoa, gde se programska operacija prikazuje jednom naredbom.
- Naredbe mogu biti:
  - Deklaracione
  - Naredbe dodeljivanja
  - Kontrolne naredbe
  - Naredbe ulaza/izlaza

Programski jezici višeg nivoa



- Ideja programskih jezika višeg nivoa:
  - – postići nezavisnost od računara,
  - – računarski program približiti problemu koji se rešava.

Programski jezici višeg nivoa

- Sastoje se obično od:
  - Skupa reči engleskog jezika,
  - Semantičkih pravila – skupa pravila za dodeljivanje simboličkih imena memorijskim lokacijama sa podacima.
  - Sintaksnih pravila – skupa pravila za formiranje ispravne naredbe u programskom jeziku.
  - Jezici visokog nivoa omogućavaju programerima da se fokusiraju na problem koji treba da se reši.
  - Jezici visokog nivoa omogućavaju portabilnost.

Kodiranje (nastavak)

- Programeri ne moraju da koriste programske jezike visokog nivoa:
  - Jezici visokog nivoa omogućavaju programerima da se fokusiraju na problem koji treba da se reši.
  - Jezici visokog nivoa omogućavaju portabilnost.

Kompajliranje

- Kod programskog jezika mora se konvertovati u mašinski jezik (0 i 1) da bi CPU razumeo.
- Kompajler je program koji razume sintaksu programskog jezika i strukturu CPUa i mašinskog jezika.
  - Kompajliranje je proces u kome je kod konvertovan u mašinski jezik
  - Kompajleri shvataju sintaksu jezika i strukturu CPUa
  - Izvorni kod (source)– instrukcije koje su programeri napisali u jezicima visokog nivoa.
  - Izvršni program (binarni kod) je spreman za distribuciju, npr. imaju ekstenziju *exe ili *.com za Windows sisteme.

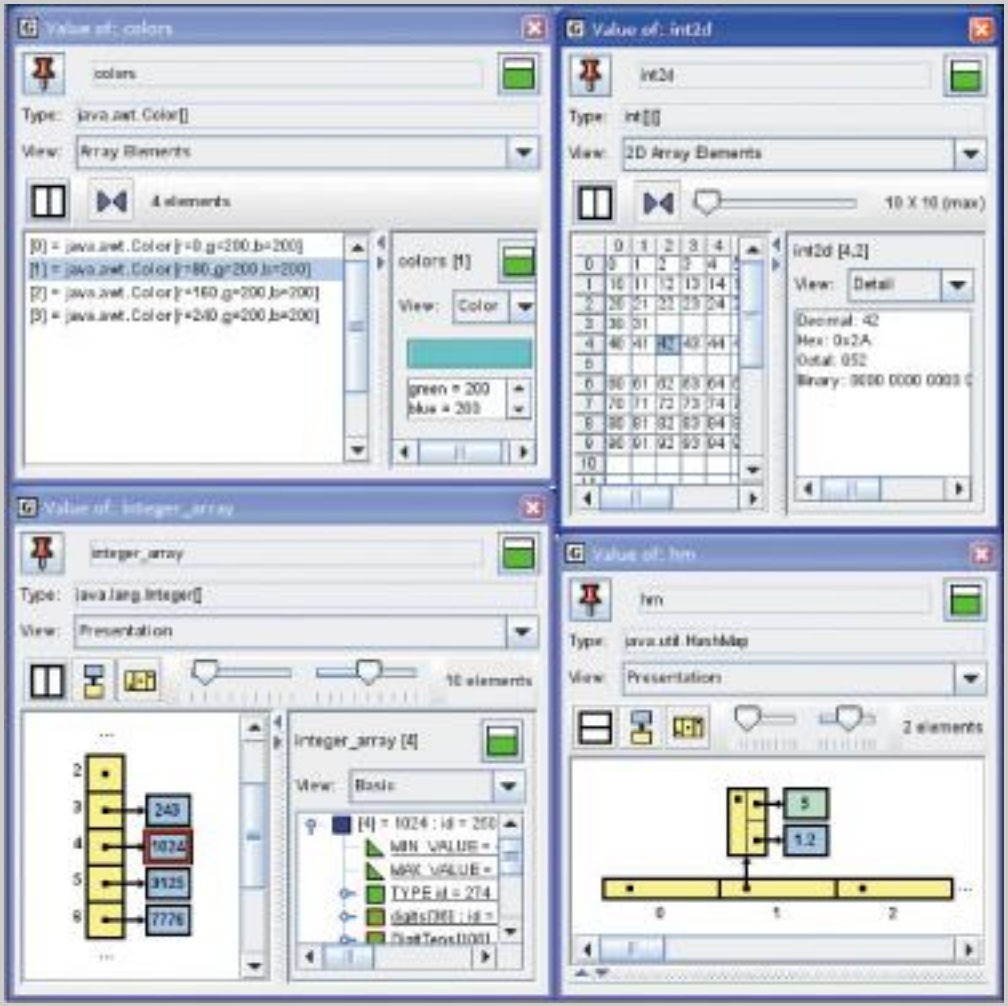
Kompajliranje.

- Nema svaki programski jezik kompajler – neki koriste i interpeter.
  - Iterperter prevodi izvorni kod u red-po-red međuformu.
  - Svaka linija se izvršava pošto se prevede.
  - Programeri ne moraju da čekaju da se ceo program ponovo kompajlira svaki put kad se napravi promena, već mogu odmah da vide rezultate kako naprave promene.

Alati za kodiranje: Integrated Development Environments (IDE)

- Alati koji olakšavaju procesa kodiranja
  - IDE: Integrated development enviroment olakšavaju programerima proces pisanja i testiranja programa.
  - Jedan IDE može se konfigurisati da podržava više jezika.

Alati za kodiranja: Integrated Development Environments (IDE)



jGRASP IDE je besplatan alat koji pomaže programerima da vizualizuju kod i da se isprave logičke greške.

Debagovanje: Ispravljanje grešaka

- Postoje alati koji pomažu programerima da nađu logičke greške:
  - *Debager* zaustavlja program, tako da vrednosti svih varijabli se mogu ispitati.
  - Može izvršavati program *polagano*, sa kontrolnim tačkama.

Testiranje i dokumentacija

- Prva runda testiranja programa:
  - Interno testiranje: Grupa u softverskoj kompaniji koristi program na svaki mogući način –> i daje se izveštaj programerima.
  - Eksterno testiranje: Korisnici rade sa programom da odrede da li zadovoljava originalnu verziju.

Testiranje i dokumentacija

- Dodatno testiranje – Beta verzija:
  - Program se daje besplatno ili po povoljnoj ceni
  - Programeri skupljaju informacije o preostalim greškama.
  - Rezultuje u finalnoj reviziji pre zvaničnog izdavanja programa.
  - Često je dostupno par meseci pre zvaničnog izdavanja.

Testiranje i dokumentacija

- Rešavanje problema nakon beta testirnja
  - Proizvođači će napraviti promene izdavanja drugim proizvođačima, npr. na novim mašinama. (Release to manufactures RTM).
  - Nakom RTMa, proizvod je u opštem slučaju dostupan i može se kupiti (poručiti).
  - Dalje uočene greške (problemi) se rešavaju kroz “service packs”
  - Npr. da se proveri poslednji service pack za Windows OS, videti windows.microsoft.com/en-US/windows/downloads/service-packs

Testiranje i dokumentacija

- Da bi se završio projekat:
  - Tehnički proizvode internu dokumentaciju
    - Razvoj i tehnički detalji softvera
    - Kako radi kod
    - Kako korisnici interaguju sa programom.
  - Pravi se korisnička dokumentacija i obučavaju se korisnici. .

Razni jezici za razne projekte

- Programeri žele rešenja koja zadovoljava nekoliko zahteva:
  - Brzo se izvršava
  - Pouzdan je
  - Lak je za proširivanje kasnije, kada se zahtevi u sistemu promene.
  - Završava se na vreme i sa najmanjom mogućom cenom.

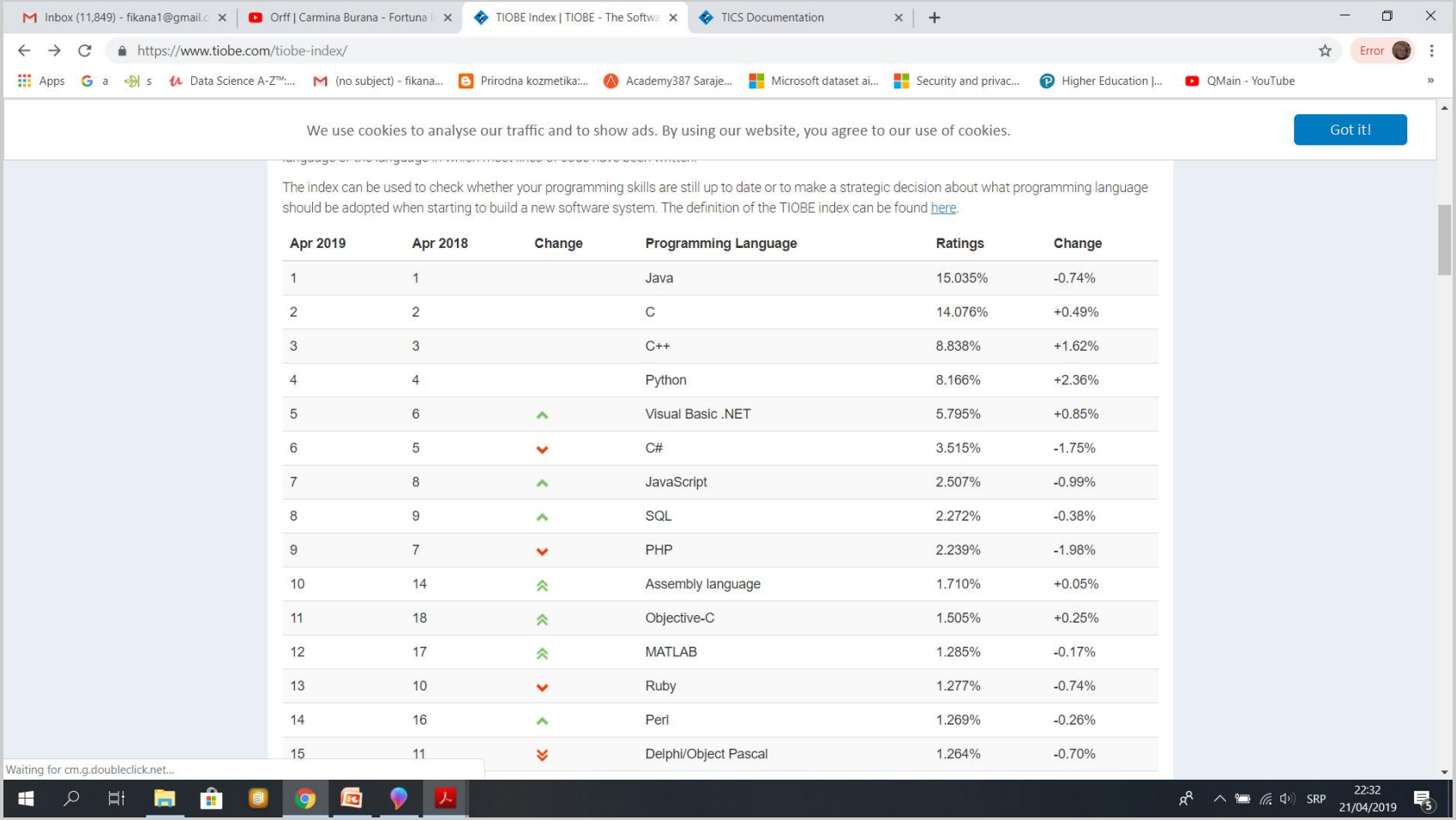
Razni jezici za razne projekte

- Programerski jezici su razvijeni da zadovolje konfliktne ciljeve:
  - Dele slične karakteristike
  - Svaki jezik ima specifične osobine
    - Najpogodniji je za određenu vrstu projekta

Razni jezici za razne projekte

- Popularni programski jezici
  - C/, C++, Java, Python. najveća potržanja
  - COBOL – za bankarstvo i osiguranje
  - Pascal – kreiran kao jezik za učenje; al danas se studenti uče na C++, Javi ili Python.
  - *Tiobe Index* koristi različite tehnike da uoči koji je jezik popularan (www.tiobe.com)

www.tiobe.com



We use cookies to analyse our traffic and to show ads. By using our website, you agree to our use of cookies. Got it!

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Apr 2019	Apr 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.035%	-0.74%
2	2		C	14.076%	+0.49%
3	3		C++	8.838%	+1.62%
4	4		Python	8.166%	+2.36%
5	6	↑	Visual Basic .NET	5.795%	+0.85%
6	5	↓	C#	3.515%	-1.75%
7	8	↑	JavaScript	2.507%	-0.99%
8	9	↑	SQL	2.272%	-0.38%
9	7	↓	PHP	2.239%	-1.98%
10	14	⇈	Assembly language	1.710%	+0.05%
11	18	⇈	Objective-C	1.505%	+0.25%
12	17	⇈	MATLAB	1.285%	-0.17%
13	10	↓	Ruby	1.277%	-0.74%
14	16	↑	Perl	1.269%	-0.26%
15	11	⇊	Delphi/Object Pascal	1.264%	-0.70%

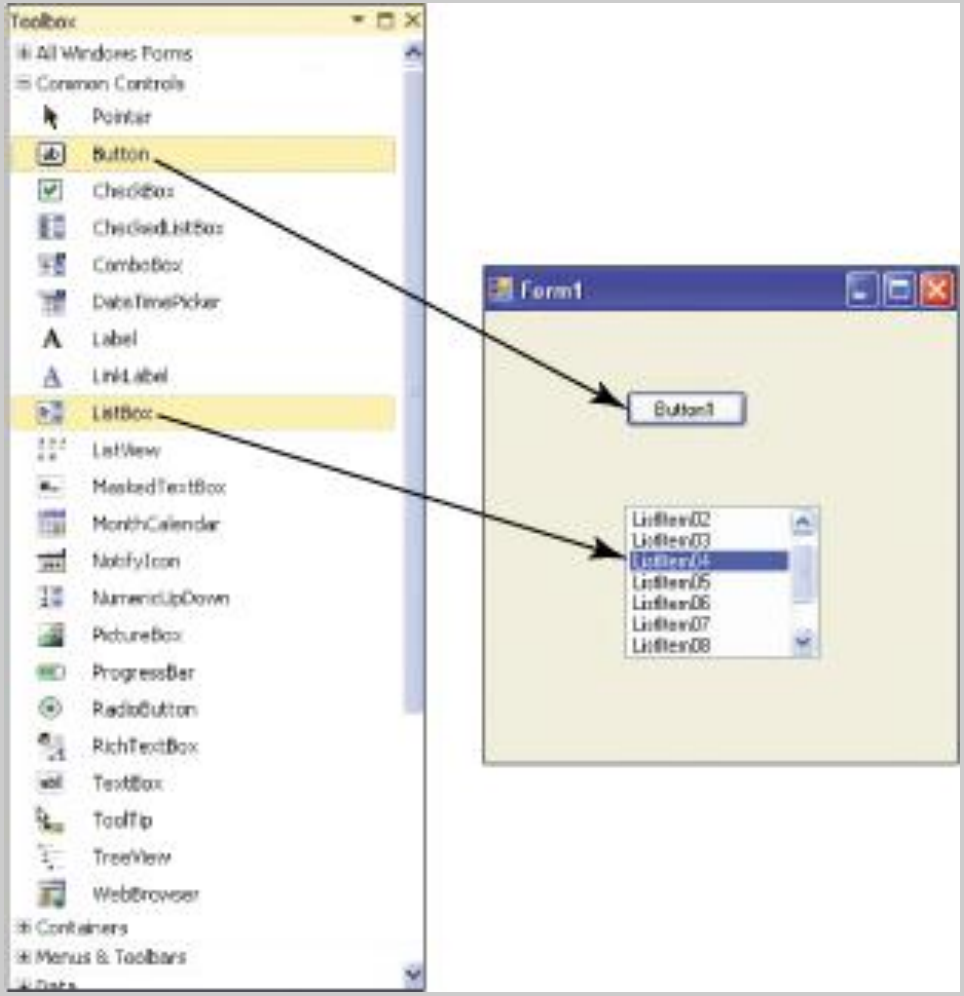
Razni jezici za razne projekte

- Faktori za izbor odgovarajućeg programa:
  - Dostupan prostor– neke aplikacije (mobilni) zahtevaju prostorno-efikasne programe.
  - Potrebna brzina – nekada važnije od veličine koda
  - Tip aplikacije – npr. podržava određeno okruženje (Unix, Windows).
  - Dostupni organizacioni resursi.

Razni jezici za razne projekte.)

- Programi za Windows okruženje:
  - Izuzetno popularni
  - Imaju zajedničke karakteristike (scroll bars, title bars, text boxes, buttons, expanding or collapsing menus)
  - Nekoliko jezika uključuje prilagođene kontrole da olakša programerima

Vizuelno programiranje



Rapid appliation development (RAD)

- Pravljenje prototipa je forma RAD
- Prvo se pravi prototip a potom generiše sistemska dokumentacija
- RAD je alternativa za pristup vodopada.

Domaći 3 (2poena)

- Napisati algoritam polaganja ispita iz Informatike.
  - Algoritam ima 10 koraka
  - Doneti na sledeća predavanja

Litratura

- Alan Evans, Kendall MartinMary, Anne Poatsy: Technology In Action 10th, 2014