Evolucija i održavanje softvera

Sadržaj

- ✧ Osnovni pojmovi
- ✧ Vrste održavanja softvera
- ✧ Cena održavanja softvera
- ✧ Tehnike održavanja softvera

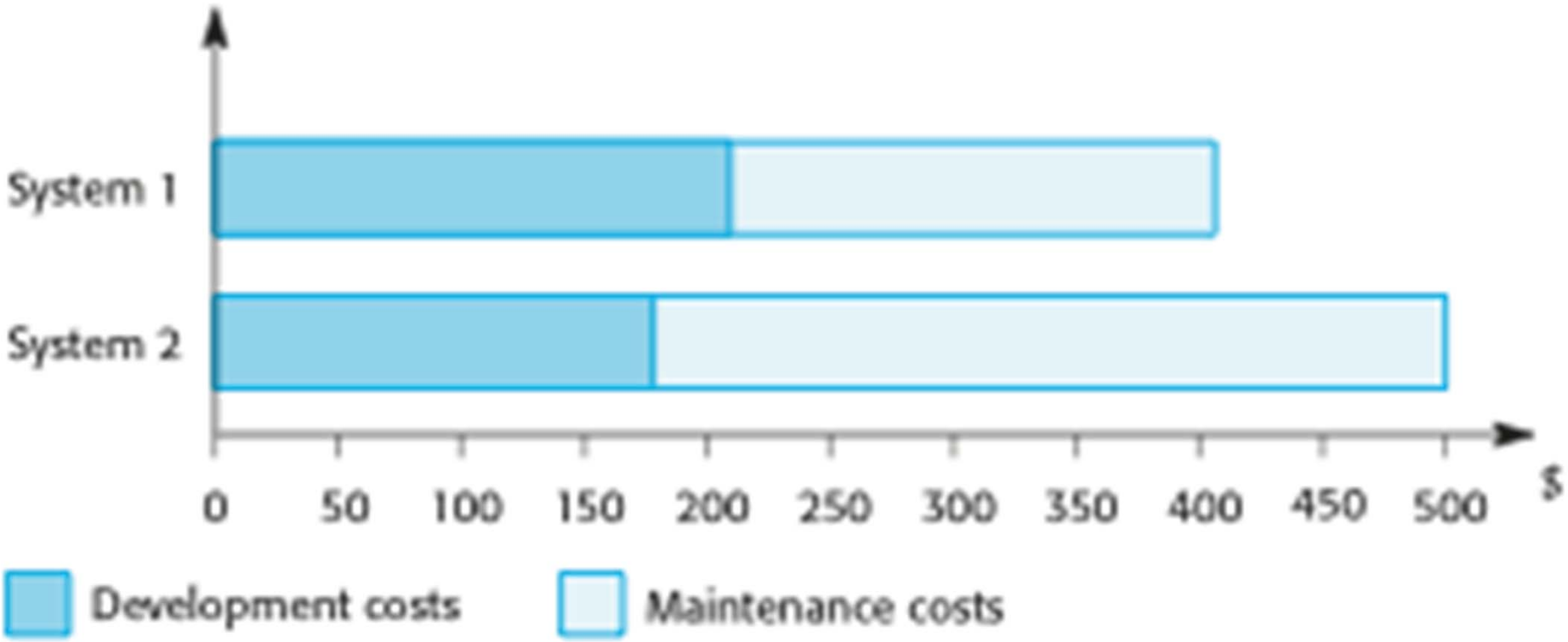
Proces razvoja sistema

Faza	Aktivnost	Izlaz
Započinjanje	Utvrđivanje poslovnih potreba	Biznis dokumenta
Analiza	Intervjuisanje stejkholdera, istraživanje sistemskog okruženja	Organizovana dokumentacija
Specifikacija	Analiza inženjerskih aspekata sistema, definisanje koncepata sistema	Logički model sistema
Implementacija	Programiranje, testiranje jedinica, integrisanje, dokumentovanje	Proverljiv sistem
Testiranje & Integracija	Integrisanje svih komponenti, verifikacija, validacija, instalacija, obuka	Rezultati testiranja, funkcionalan sistem
Održavanje	Popravljanje bagova, modifikacije, adaptacija	Verzije sistema

Održavanje softvera

- Održavanje softvera je
  - Proces modifikovanja softverskog sistema ili komponente nakon isporuke radi otklanjanja grešaka, poboljšanja performansi ili drugih atributa, ili prilagođavanja promenljivom okruženju.
  - Softverski proizvod se podvrgava modifikaciji koda ili dokumentacije usled problema ili potrebe za unapređenjem. Cilj je modifikovanje postojećeg softverskog proizvoda dok se istovremeno očuvava njegov integritet
- Održavanje programa nakon što je stavljen u upotrebu
- Održavanje obično ne obuhvata značajne izmene na arhitekturi sistema
  - Izmene se implementiraju modifikovanjem postojećih komponenti i dodavanjem novih komponenti sistemu

Odnos cene razvoja i održavanja



Vrste održavanja softvera

- Adaptivno održavanje
  - Modifikacija softverskog proizvoda izvršena nakon isporuke da bi se očuvala upotrebljivost softverskog proizvoda u promenjenom ili promenljivom okruženju.
- Korektivno održavanje
  - Ispravljaju se uočene greške. Može se raditi o greškama u kodiranju, u oblikovanju, odnosno u specifikaciji.
- Perfekcijsko održavanje
  - Modifikacija softverskog proizvoda nakon isporuke radi unapređenja performansi ili dodavanja novih karakteristika.
- Preventivno održavanje
  - Modifikacija softverskog proizvoda nakon isporuke radi utvrđivanja i korigovanja latentnih grešaka u softverskom proizvodu pre nego što one nanesu štetu.

Evolucija softvera

✧ Evolucija softvera je

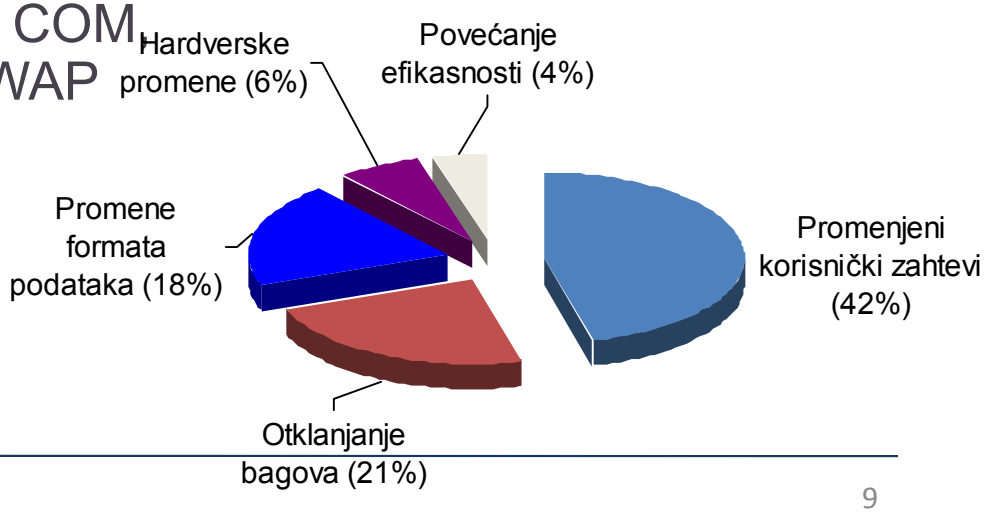
- Skup aktivnosti, tehničkih i upravnih, koje obezbeđuju da softver nastavi da ispunjava organizacione i poslovne ciljeve na isplativ način
- Sve programerske aktivnosti čiji je cilj generisanje nove verzije softvera na osnovu starije operacione verzije
- Primena aktivnosti i procesa održavanja softvera kojima se generiše nova operativna verzija softvera sa novim funkcionalnostima ili karakteristikama u odnosu na prethodnu operativnu verziju pri čemu su obuhvaćene i aktivnosti obezbeđivanja kvaliteta

Značaj evolucije

- Najveći deo budžeta za razvoj softvera u velikim kompanijama je u mnogo većoj meri namenjen adaptaciji (evoluciji) postojećeg softvera, a u manjoj razvoju novog softvera.

Osnovni razlozi za evoluciju softvera

- Promene korisničkih zahteva
  - Proširenja ili modifikacije koje je zahtevao korisnik
- Otklanjanja bagova
  - Planirane aktivnosti otklanjanja
  - Nužno otklanjanje (uglavnom skupo zbog velikog pritiska)
- Promene formata podataka
  - Y2K, Euro, poreske rate, poštanski kodovi, telefonski brojevi, ...
  - Novi standardi: UML, XML, COM, DCOM, CORBA, ActiveX, WAP
- Hardverske promene
- Povećanje efikasnosti



Starenje softvera

- Neophodno je da naučimo kako da sprečimo efekte starenja
  - Programi, kao i ljudi stare. Starenje ne možemo sprečiti, ali možemo razumeti njegove uzroke, preduzeti korake da ograničimo njegove efekte, privremeno otklonimo štetu koju je izazvalo i pripremimo se za dan kada softver više neće biti upotrebljiv.

Starenje softvera

- Razlozi za starenje softvera
  - Održavanje
  - Nefleksibilnost od početka projekta
  - Nedovoljna ili nekonzistentna dokumentacija
  - Pritisak krajnjih rokova
  - Dupliranje funkcionalnosti (dupliranje koda)
  - Nedostatak modularnosti
  - ...
- Moguće rešenje: reinženjerstvo

Legacy (nasleđeni) sistemi - definicija

- Bilo kakvi informacioni sistem koji se opire promenama
- Postojeći kompjuterski sistem ili aplikacioni program koji se nastavlja koristiti jer korisnik (najčešće organizacija) ne želi (ne može) da ga zameni ili redizajnira. Mnogi ljudi koriste ovaj pojam da ukažu na “zastarele” sisteme.

Legacy (nasleđeni) sistemi - definicija

- Sa tehnološkog aspekta – čak i kompletno funkcionalan i održiv sistem se može smatrati zastarelim ukoliko koristi prevaziđenu tehnologiju.
- Sa ekonomskog aspekta – sistem se može smatrati zastarelim ukoliko ne može da prati tempo promena u poslovnom domenu.

Problemi sa legacy sistemima

- Često se izvršavaju na zastarelom hardveru
- Teško se održava, unapređuje i proširuje
- Opšti nedostatak razumevanja sistema:
  - Nema osoba koje mogu objasniti kako funkcioniše
  - Dokumentacija ili uputstva su se izgubili tokom godina
- Teška je integracija sa novim sistemima

Razlozi za korišćenje legacy sistema (uprkos problemima)

- Troškovi redizajna sistema su preterano visoki jer je sistem veliki, monolitski i/ili kompleksan.
- Sistem zahteva 100% dostupnost, pa ne može biti stavljen van upotrebe.
- Način na koji sistem funkcioniše nije dobro shvaćen.
- Korisnik očekuje da sistem može biti jednostavno zamenjen kada to bude neophodno.
- Sistem radi zadovoljavajuće, i vlasnik ne vidi razlog za njegovu promenu.

Legacy sistemi – moguće rešenje

- Reinženjerstvo je sistematična transformacija postojećeg sistema u novu formu da bi se
  - Realizovala kvalitativna unapređenja u operativnosti, mogućnostima sistema, funkcionalnosti i performansama,
  - ili evoluiralo ka manjim troškovima, uštedi vremena ili smanjenju rizika korisnika.

Dinamika održavanja softvera

✧ Lehman-ovi zakoni:

- Nužnost menjanja – softver koji se zaista koristi u stvarnom svetu nužno se mora menjati jer u protivnom ubrzo postaje neupotrebljiv.
- Povećanje složenosti – dok se softver menja, njegova struktura teži tome da postane sve složenija. Da bi se očuvala jednostavnost strukture, potrebno je uložiti dodatni trud i resurse.
- Ograničena brzina unapređivanja – količina "novosti" koju pojedino izdanje softvera može doneti otprilike je konstantna i karakteristična za taj softver.

Dinamika održavanja softvera

- Kontinuirani rast – funkcionalnosti koje softver nudi moraju se konstantno povećavati kako bi se očuvalo zadovoljstvo korisnika.
- Opadanje kvaliteta softvera – kvalitet softvera će opadati ukoliko se ne modifikuje u skladu sa promenama u operacionom okruženju.

Cena održavanja softvera

✧ Osnovni faktori koji utiču na cenu održavanja:

- Celovitost polazne specifikacije – ukoliko odmah uključimo sve zahteve, kasnije će biti manje perfekcijskog održavanja.
- Kvalitet dizajna – dobar dizajn je jeftiniji za održavanje. Smatra se da su sa stanovišta održavanja najbolji objektno-orjentisani sistemi, koji se sastoje od malih modula sa jakom unutrašnjom kohezijom i labavim vezama prema spolja.
- Način implementacije – Kod u "strožem" programskom jeziku poput Jave lakše se održava nego kod u jeziku poput C-a. Struktirirani kod (if, while) sa smisleno imenovanim varijablama razumljiviji je od kompaktnog koda s mnogo goto naredbi.
- Stepen verifikovanosti – dobro verifikovani softver ima manje grešaka pa će zahtevati manje korekcijskog održavanja.

Cena održavanja softvera

- Stepen dokumentovanosti – uredna, dobro struktuirana i celovita dokumentacija olakšava razumevanje softvera, pa na taj način pojeftinjuje održavanje.
- Starost softvera – što je softver stariji, to je skuplji za održavanje, budući da mu se građa degradirala, zavistan je od zastarelih razvojnih alata, a dokumentacija mu je postala neažurna.
- Svojstva domena aplikacije – ako je reč o stabilnom domenu gde se poslovna pravila retko menjaju, tada će se retko pojavljivati potreba za perfekcijskim održavanjem u svrhu usklađivanja s novim pravilima.
- Stabilnost razvojnog tima – održavanje je jeftinije ako se njime bave originalni implementatori softvera, jer oni ne moraju trošiti vreme na upoznavanje sa softverom.
- Stabilnost platforme – ako smo softver implementirali na platformi koja će još dugo biti savremena, tada neće trebati adaptacijsko održavanje.

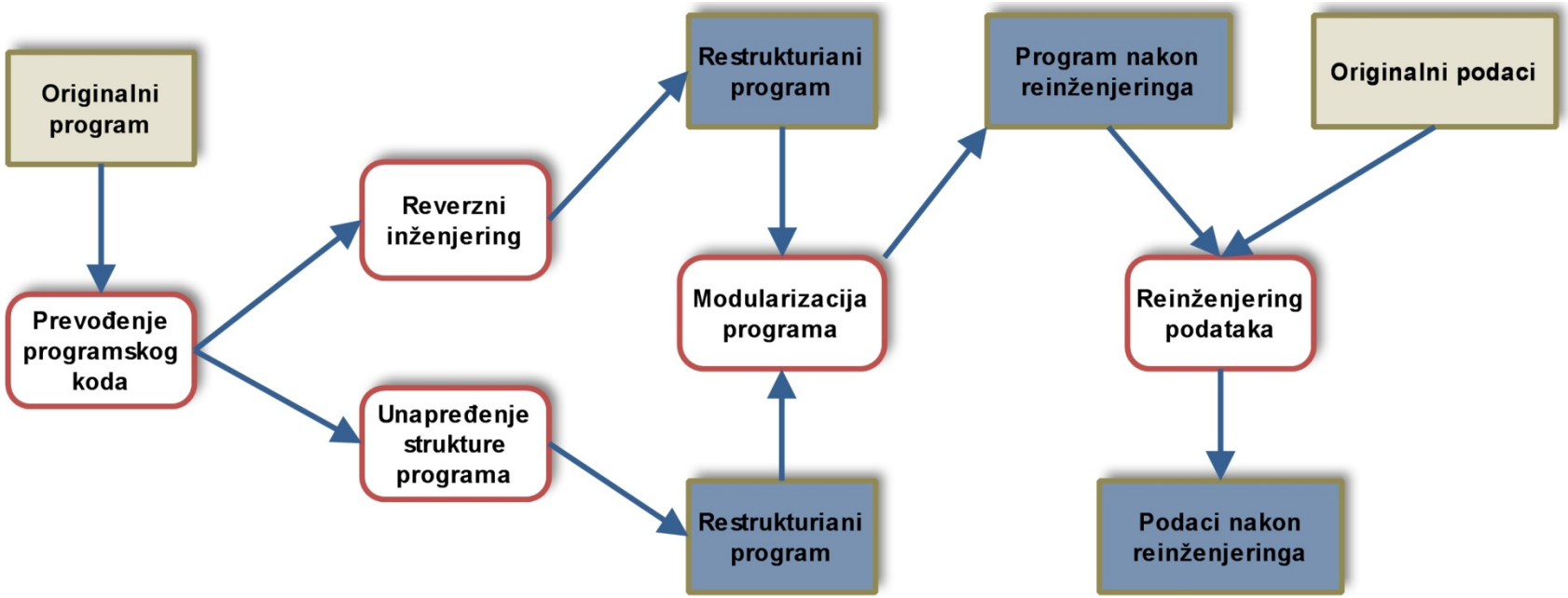
Sistemski reinženjering

- Restruktuiranje ili ponovno pisanje dela ili celog legacy sistema bez promene njegovih funkcionalnosti.
- Pogodan je tamo gde pojedine (ali ne sve) podkomponente većeg sistema zahtevaju često održavanje.
- Obuhvata dodatne napore da bi se softver učinio lakšim za održavanje. Sistem može biti restruktuiran i redokumentovan.

Prednosti reinženjeringa

- ✧ Smanjeni rizik
  - Postoji veliki rizik ukoliko se započne razvoj novog softvera. Mogu postojati problemi u razvoju, u obezbeđivanju ljudstva i speficikaciji.
- ✧ Smanjenje troška
  - Troškovi reinženjeringa su značajno manji od troškova razvoja novog softvera.

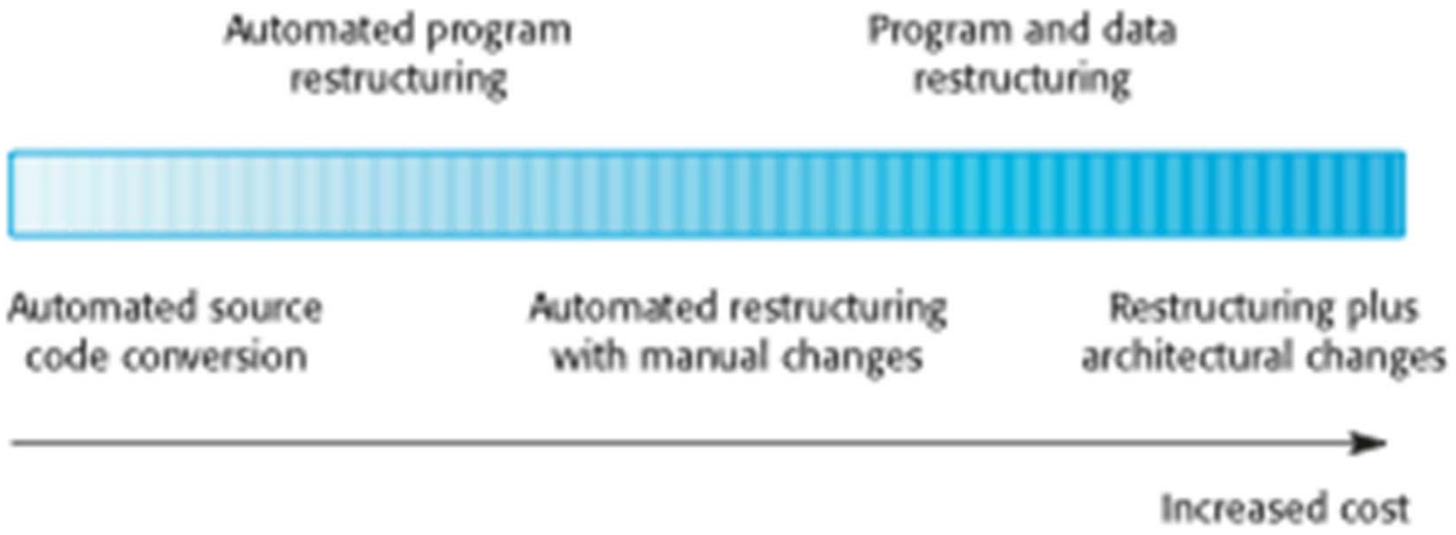
Proces reinženjeringa



Aktivnosti procesa reinženjeringa

- Prevođenje programskog koda
  - Prevođenje koda u drugi programski jezik
- Reverzni inženjering
  - Analiziranje programa kako bi se razumeo
- Unapređenje strukture programa
  - Struktura programa se analizira i modifikuje kako bi se jednostavnije razumela
- Modularizacija programa
  - Grupisanje povezanih celina programa gde god je to moguće kako bi se izbegla redundansa
- Reinženjering podataka
  - Redefinisanje šeme baze podataka i konvertovanje u novu strukturu

Pristupi reinženjeringu



Faktori koji utiču na cenu reinženjeringa

- Kvalitet softvera za koji se vrši reinženjering.
- Dostupnost alata za podršku reinženjeringu.
- Obim podataka za koje je potrebno izvršiti konverziju.
- Dostupnost eksperata za reinženjering.
  - Ovo može biti problem kod starih sistema koji su zasnovani na tehnologiji koja se ne koristi.

Preventivno održavanje korišćenjem refaktoringa

- Refaktoring je proces unapređenja programa kako bi se usporila njegova degradacija u toku promena.
- Refaktoring možemo posmatrati kao preventivno održavanje koje smanjuje probleme u budućim promenama.
- Obuhvata modifikaciju programa kako bi se unapredila njegova struktura, smanjila kompleksnost i jednostavnije razumeo.
- Kada vršite refaktoring programa, ne treba da dodajete nove funkcionalnosti već da se koncentrišete na unapređenje programa.

Refaktoring i reinženjering

- Reinženjering se koristi kada je sistem održavan određeno vreme i kada se troškovi održavanja povećavaju. Koristite automatske alate za obradu i reinženjering legacy sistema kako bi se kreirao novi sistem koji se lakše održava.
- Refaktoring je kontinuirani proces unapređenja kroz razvoj i evoluciju. Cilj mu je da se izbegne degradacija strukture i koda koji bi povećali troškove i otežali održavanje sistema.
-

‘Loši pokazatelji’ u programskom kodu

- ✧ Dupliranje koda
  - Isti ili sličan kod može biti uključen na različitim mestima u programu. Ovo se može ukloniti i implementirati kao jedna metoda koja se poziva kada je potrebno.
- ✧ Dugačke metode
  - Ako je metoda previše dugačka, treba je redizajnirati kao više kraćih metoda.
- ✧ Switch (case) iskazi
  - Ovakvi iskazi često obuhvataju dupliranje, ako switch zavisi od tipa vrednosti. U objektno orjentisanim jezicima može se koristiti polimorfizam da bi se postigla ista stvar.

‘Loši pokazatelji’ u programskom kodu

- ✧ Ponavljanje istih podataka (Data clumping)
  - Dešava se kada se grupa istih podataka (polja u klasama, parametri u metodama) javlja na više mesta u programu. Ovo se uglavnom može zameniti objektom koji enkapsulira sve ove podatke.
- ✧ Generalizacija zbog pretpostavki
  - Dešava se kada programeri generalizuju elemente u programu za slučaj da će im trebati u budućnosti. Ovo se jednostavno može ukloniti.