

# RAZVOJ SOFTVERA

SOFTWARE EVOLUTION

# UVOD U RAZVOJ SOFTVERA

Ovde ćemo govoriti zašto je razvoj softvera bitan deo softverskog inženjerstva i objasniti procese razvoja softvera

Bitno je napomenuti da razvoj softvera ne prestaje nakon njegove isporuke već se on nastavlja tokom čitavog životnog ciklusa softvera

Razvoj softvera je veoma bitan jer kompanije ulažu velike količine novca u njiž s obzirom da uglavnom dosta zavise od tog softvera

Takodje mnogo više novca se ulaže u održavanje tog softvera nego u sam razvoj softvera

Razvoj softvera može biti pokrenut usled izmene poslovnih zahteva, prilikom pojave grešaka u radu softvera ili promenom sistema na kojem se radi

Kvalitetni i korisni softveri uglavnom imaju dug život. Sistemi obicno koštaju dosta novca pa kompanije teže ka tome da ih što duže koriste kako bi se ulaganja isplatila



O razvoju softvera treba da razmišljamo kao spiralnom procesu koji podrazumeva zahteve, dizajn, implementacije i testiranja koji se dešavaju tokom životnog veka sistema.

Počinjemo tako što pravimo prvo izdanje sistema.

Kada je jednom isporučen, izmene isporučenog izdanja kao i razvoj drugog izdanja kreću istovremeno.

Potreba za razvojem može da se javi i pre nego što se sistem isporuči što može dovesti do toga da neka kasnija izdanja softvera budu u fazi razvoja i pre nego što je tekuća verzija objavljena.



Ovaj model razvoja softvera podrazumeva da je jedna organizacija odgovorna i za početni razvoj softvera kao i za kasniji razvoj

Kupac softvera može napraviti dogovor sa nekom drugom kompanijom koja će kasnije vršiti održavanje i razvoj sistema, u ovom slučaju oni bi prekinuli taj spiralni proces

Proces menjanja softvera nakon isporuke se često naziva “održavanje softvera”



Rajlich i Bannett (2000) uvode novi pogled na odvijanje razvoja softvera. U ovom modelu, oni prave razliku između razvoja i servisiranja.

*Razvoj* je faza koja podrazumeva značajne promene koje mogu da se obave na softveru.

*Servisiranje* je faza u kojoj se vrše neke male, suštinske promene.

Prilikom razvoja, softver se uspešno koristi i postoji konstantan tok izvršavanja promena na njemu



# PROCES RAZVOJA SOFTVERA

Proces razvoja softvera dosta zavisi od vrste softvera koji se održava, procesa koji je korišćen za razvoj tog softvera i od veštine ljudi koji su uključeni u proces razvijanja

Predlozi promena sistema su putokaz u razvoju sistema u svim organizacijama

Proces uočavanja promena i razvoja sistema je cikličan i nastavlja se kroz životni vek sistema

Arthur (1988) – proces razvoja softvera



Velika razlika je ta da prvi stadijum implementacije promena na softveru može da dovede do proučavanja programa, pogotovu ako prvobitni razvijaoac sistema nije odgovoran za implementaciju promena

Razumevanje programa je neophodno da bi bili sigurni da implementirana promena neće prouzrokovati nove probleme kada se uključi u sistem

Tokom procesa razvoja, zahtevi se analiziraju do detalja i nekada je potrebno razgovarati sa klijentom pre nego sto se promene implementiraju



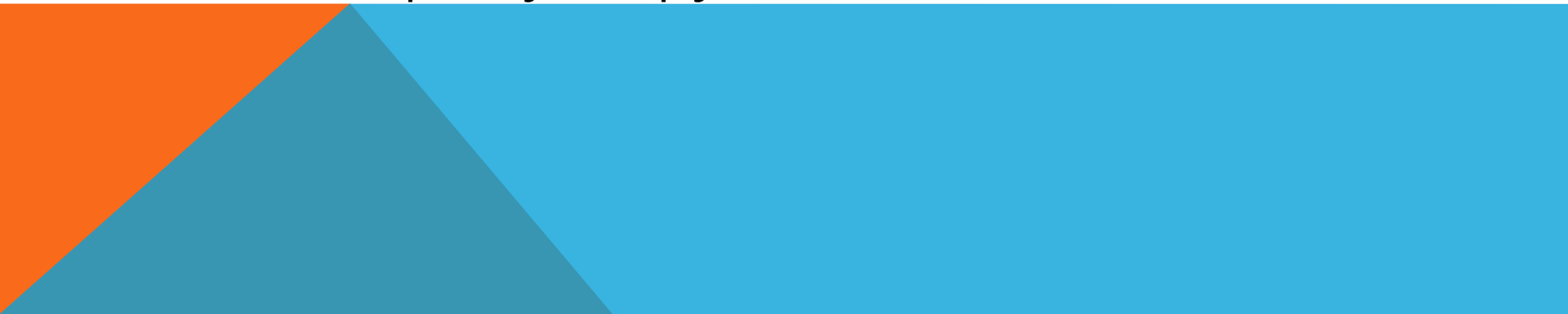


Zahtevi za promenama se nekad odnose na sistemske probleme koje je potrebno hitno resavati. U ovim slučajevima možda nećemo biti u mogućnosti da pratimo proces formalne analize promena

Opasnost je ta što zahtevi, dizajn softvera i kod ne ostaju u skladu

Ovde se uvek bira brže i efektivnije rešenje nego najbolje moguće sve dok je sistem u kritičnoj fazi

Ovo ubrazava proces starenja softvera tako da kasnije promene postaju sve komplikovanije i održavanje sistema postaje skuplje



Agilne metode i procesi se mogu koristiti za razvoj programa

Problemi se mogu javiti u situacijama kada se vrši predaja softvera od tima koji napravio softver timu koji učestvuje u kasnijem razvoju tog softvera

- stvaralački tim koristi agilni pristup, a razvojni tim ne
- za izradu softvera se koristi pristup zasnovan na planiranju, a za kasniji razvoj agilne metode



# DINAMIKA RAZVOJA PROGRAMA

Dinamika razvoja programa je studija o promenama sistema

Lehman i Belady su sproveli nekoliko empirijski studija o promeni sistema sa ciljem da što više uzmu u obzir karakteristike razvoja softvera

Iz ovih studija potekli su „Lehmanovi zakoni“



- *Prvi zakon* kaže da je održavanje sistema neizbežan proces
- *Drugi zakon* kaže da ako je sistem promenjen, njegova struktura je degradirana
- *Treći zakon* je najzanimljiviji. Predlaže da veliki sistemi imaju sopstvenu dinamiku koja je osnovana u ranoj fazi razvoja
- *Lehmanov četvrti zakon* govori o tome da promena resursa ili promena osoblja ima minimalan efekat na dugoročan razvoj sistema
- *Lehmanov peti zakon* govori da što se više funkcionalnosti dodaje više je i problema koji nastaju i koji se moraju rešiti
- *Šesti i sedmi zakon* su slični i u suštini kažu da će korisnici softvera biti jako nesrećni sve dok je softver u fazi razvoja i dok se dodaju nove funkcionalnosti

# ODRŽAVANJE SOFTVERA

Održavanje softvera podrazumeva promene sistema nakon njegovog isporučivanja.

Promene na softveru mogu biti promene koje su se pojavile u kodu prilikom kodiranja, malo veća promena bi bila promena u dizajnu, promene radi ispravljanja grešaka u specifikaciji ili promene usled ubacivanja novih zahteva

Promene se implementiraju menjajući postojeći sistem i gde je to potrebno, dodavanje novih komponenata u sistem



Postoje tri različita tipa održavanja sistema:

- *Popravke grešaka* - jeftine za korekciju
- *Adaptacija na okruženje* - kada ne postoji sklad sa sistemskim okruženjem kao što je hardver
- *Dodavanje funkcionalnosti* - kada dođe do promena u sistemskim zahtevima
- *Softverske greške* su česta pojava jer korisnici često koriste sofver na neki neočekivani način. Promena sistema kako bi se olakšao rad na njima je najbolji način za rešavanje ovih problema.



Rad tokom razvoja softvera na tome da softver učinimo lakšim za razumevanje i menjanje najverovatnije dovodi do kasnijeg smanjenja troškova prilikom razvoja


To je razlog za refaktorisanje u agilnom razvoju.

Bez refaktorisanja kod postaje sve više i više komplikovan i skup za promene

U razvoju zasnovanom na planiranju realnost je da je dodatno ulaganje u poboljšanje koda vrlo retko prilikom razvoja



Obično je mnogo skuplje dodavati funkcionalnosti kada je sistem već isporučen:

- *Stabilnost tima*
  - *Loša praksa razvoja*
  - *Starost programa i struktura*
  - *Veštine osoblja*
- 



# PROGNOZE ODRŽAVANJA

Što su programi kompleksniji to je skuplje kasnije održavanje sistema

Kako bi se smanjili troškovi održavanja treba se truditi da kompleksne delove programa zamenimo manje kompleksnim alternativama

Nakon što sistem pustimo u korišćenje, možemo iskoristiti to da pogledamo rezultate rada kako bi predvideli proces održavanja sistema



# SOFTVERSKI REINŽENJERING

Mnogi sistemi, pogotovu stariji sistemi, su teški za razumevanje i menjanje

Kako bi učinili da stariji softveri budu lakši za održavanje, možemo se baviti reinženjeringom ovih sistema i na taj način unaprediti njihovu strukturu i razumljivost

Reinženjering može obuhvatiti preuređivanje dokumentacije, refaktorisanje arhitekture sistema, prevođenje programa na moderne programske jezike i modifikaciju i ažuriranje strukture i vrednosti sistemskih podataka



Prednosti reinženjeringa u odnosu na zamenu softvera:

- *Smanjena opasnost*
- *Smanjeni troškovi*

Ulaz u proces uglavnom obuhvata neki stari softver, dok izlaz iz procesa obuhvata isti taj softver samo poboljšan




Osnovni koraci u reinženjeringu:

- *Prevođenje izvornog koda*
- *Prepravljanje softvera*
- *Unapređivanje programske strukture*
- *Rekonstruisanje podataka*

Programski reinženjering ne mora sadržati sve prethodno navedene korake

Problem kod softverskog reinženjeringa je taj što postoje granice u tome koliko možemo da unapredimo sistem koristeći inženjering



# PREVENTIVNO ODRŽAVANJE SISTEMA REFAKTORISANJEM

Refaktorisanje je proces uvođenja poboljšanja u sistem radi usporavanja degradacije prilikom promena. To podrazumeva modifikovanje programa kako bi se unapredila njegova struktura, kako bi se smanjila njegova kompleksnost ili kako bi ga učinili jednostavnijim za razumevanje

Refaktorisanje na neki način smanjuje probleme u budućim izmenama

Refaktorisanje je kontinuiran proces unapređivanja za vreme razvoja softvera

Refaktorisanje je sastavni deo agilnih metoda ali ne zavisi od drugih agilnih metoda i može se primeniti bilo gde

Primeri koda “koji smrdi” a koji se može unaprediti refaktorisanjem:

- *Ponavljajući kod*
- *Dugački metodi*
- *Switch-case naredba*
- *Lanci poruka (data clumping)*
- *Spekulativno uopštavanje*

# UPRAVLJANJE ZASTARELIM SISTEMOM

Mnoge firme imaju još uvek zastarele sisteme za koje imaju ograničene budžete za održavanje i unapređivanje sistema.

Postoje 4 strategijske mogućnosti za unapređenje ovih sistema:

- *Uništiti sistem kompletno*
- *Ostaviti sistem neizmenjen i krenuti sa regularnim održavanjem*
- *Rekonstruisanje sistema kako bi se unapredilo njegovo održavanje*

*Zameniti deo ili ceo stari sistem novim sistemom*

Kada procenjujemo stari sistem moramo ga posmatrati iz poslovne i tehničke perspektive


Tada kombinujemo poslovne vrednosti i kvalitet sistema kako bi lakše odlučili šta da radimo sa zastarelim sistemom

- *Loš kvalitet, niska poslovna vrednost*
- *Loš kvalitet, visoka poslovna vrednost*
- *Visok kvalitet, niska poslovna vrednost*
- *Visok kvalitet, visoka poslovna vrednost*





Kako bi procenili poslovnu vrednost sistema neophodno je odrediti aktere sistema, kao sto su krajnji korisnici i njihovi menadžeri i postaviti im seriju pitanja o sistemu

- *Koliko je sistem koristan*
  - *Koje poslovne procese softver podrzava*
  - *Pouzdanost sistema*
  - *Rezultati rada sistema*
- 

Kako bi procenili tehnički kvalitet aplikativnog sistema moramo proučiti niz faktora koji se prvenstveno odnose na pouzdanost, kompleksnost održavanja sistema kao i na njegovu dokumentaciju

Podaci koji mogu biti korisni u proceni kvaliteta:

- *Broj zahteva za izmenu sistema*
  - *Broj korisnickih interfejsa*
  - *Kolicina podataka koje koristi sistem*
- 