PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Softverski procesi

Lekcija 02

PRIRUČNIK ZA STUDENTE

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Lekcija 02

SOFTVERSKI PROCESI

UVOD

Ključna pitanja koja se raymatraiju u ovoj nastavnoj jedinici:

- Objasniti koncept softverskih proces i modela softverskih proces.
- Koja su tri opšta modela softverskog procesa i kada se oni koriste?
- Koje su osnovne aktivnosti inženjerstva utvrđivanja zahteva za softverom, za testiranjem i evolucijom?
- Zašto procesi treba da budu tako organizovani da mogu da se prilagođavaju promenama zahteva i projektnog rešenja softvera?
- Kako Rational Unified Proces integriše dobru inženjersku praksu radi kreiranja prilagodljivih softverskih procesa

Poglavlje 1

Softverski proces

ŠTA JE SOFTVERSKI PROCES?

Softverski proces je skup povezanih aktivnosti koji vodi proizvodnji softverskog proizvoda.

Softverski proces je skup povezanih aktivnosti koji vodi proizvodnju softverskog proizvoda. Ove aktivnosti mogu da dovedu do razvoja potpuno novog softvera u nekom od standardnih programskih jezika (npr. Java ili C), ili do usavršavanja nekog postojećeg softvera, što je najčešći slučaj kod poslovnih aplikacija.

Najčešće se softverski procesi sadrže sledeće četiri aktivnosti koje čine osnovu softverskog inženjerstva:

1. *Specifikacija softvera: Definiše funkcionalnost softvera i ograničenja na rad softvera.*
2. *Projektovanje (dizajn) i razvoj (implementacija) softvera: Definiše kako softver ostvaruje svoju specifikaciju i kako se on pravi.*
3. *Provera (validacija) softvera: Softver se proverava da bi se utvrdilo da li zadovoljava potrebe i očekivanja korisnika.*
4. *Evolucija softvera: Softver mora da ima svoju evoluciju (stalni razvoj i prilagođavanje) da bi zadovoljio nove zahteve korisnika.*

Ove osnovne aktivnosti mogu da imaju svoje pod-aktivnosti (npr. provera zahteva, dizajn arhitekture, jedinični testovi i dr.). Postoje i aktivnosti koje podržavaju osnovne aktivnosti procesa, kao što je izrada tehničke dokumentacije, i definisanje konfiguracije softvera.

Pored navođenja aktivnosti, jedan softverski proces se opisuje i navođenjem dugih faktora, kao što su:

1. *Proizvodi koji su rezultat aktivnosti procesa.*
2. *Uloge (eng. Roles) koje definišu odgovornosti ljudi koji učestvuju u procesu.*
3. *Uslovi koje neka aktivnost mora da zadovolji pre nego što počne sa radom (preduslovi), a da bi završila sa radom (izlazni uslovi).*

OSNOVNE AKTIVNOSTI SOFTVERSKIH PROCESA

Četiri osnovne aktivnosti softverskih procesa su : specifikacija, razvoj, validacija i evolucija

Stvarni softverski procesi su mešavina sekvenci tehničkih, kolaborativnih i upravljačkih aktivnosti sa ukupnim ciljem da specificiraju, projektuju, implemetiraju i testiraju neki softverski sistem. Softverski inženjeri u svom radu koriste različite softverske alate radi rada sa različitim tipovima dokumenata i uređivanja velike količine detaljnih informacija koji se stvara u projektu razvoja nekog velikog softverskog sistema.

Četiri osnovne aktivnosti softverskih procesa su :

1. Specifikacija softvera
2. Razvoj softvera
3. Validacija softvera i
4. Evolucija softvera

.

One se različito organizuju u različitim procesima razvoja softvera. U modelu vodopada, aktivnosti su organizovane redno (jedna posle druge), dok su pri inkrementalnom razvoju one izmešane. Kako se te aktivnosti realizuju, to zavisi od tipa projekta, ljudi, i organizacione strukture. Pri primeni agilne metodologije poznate kao Ekstremno programiranje, specifikacija se piše na karticama. Testovi se izvršavaju i razvijaju pre samog programa. Evolucija softvera može da obuhvati značajnu promenu strukture softvera

OSNOVNI TIPOVI SOFTVERSKOG PROCESA

Softverski procesi se mogu podeliti na 1) procese vođene planom, i na 2) agilne procese.

Softverski procesi su složeni procesi, kao i svi intelektualni i kreativni procesi u kojima učestvuju ljudi koji donose odluke. Zbog toga, oni zavise od ljudi koji rade u organizacijama i odražavaju specifične karakteristike tih organizacija. Zato se softverski procesi razliku od organizacije do organizacije.

Na procese utiče i tip softvera. Ako se radi o softveru koji upravlja tzv. kritičnim sistemima, onda on mora da sledi vrlo formalizovanu proceduru. S druge strane, kako se poslovne aplikacije razvijaju u uslovima promenljivih zahteva korisnika, to su procesi za njihov razvoj manje formalizovani, te se lakše prilagođavaju promenama da bi bili efektivniji. Zbog toga, softverski procesi mogu podeliti i na:

1. *procese vođene planom, i na*
2. *agilne procese.*

Procesi vođeni planom su procesi kod kojih su sve aktivnosti unapred planirane i napredak u razvoju softvera se određuje stepenom ostvarivanja tog plana.

Agilni procesi su procesi kod koji se planirane radi postupno (inkrementalno) kako bi se proces lakše prilagodili promenljivim zahtevima korisnika.

Obe vrste procesa se koriste, zavisno od vrste softvera,a mogu se i kombinovati. Izazov za softverske inženjere je da primene procese koji najbolje odgovaraju uslovima njihove organizacije, jer idealnih (i opšte prihvaćenih procesa) nema.

Procesi se mogu poboljšati primenom odgovarajućih *standarda, pri čemu se smanjuju specifične razlike između organizacija. Ovo dovodi da smanjivanja vremena obuke, bolje komunikacije i povećanje ekonomičnosti automatizacije procesa. Standardizacija je važan prvi korak uvođenja novih metoda softverskog inženjerstva i tehnika, i dobra praksa softverskog inženjerstva.*

1.1 Validacija softvera

ŠTA JE VALIDACIJA SOFTVERA?

Validacija (potvrđivanje) softvera, ili tačnije verifikacija i validacija, treba da pokaže da li sistem zadovoljava svoju specifikaciju i očekivanje korisnika sistema.

Validacija (potvrđivanje) softvera treba da pokaže da li sistem zadovoljava svoju specifikaciju i očekivanje korisnika sistema. Za to, koristi se testiranje sistema sa simuliranim test podacima. Validacija može da obuhvati i procese provere (inspekcija i recenzija) svake faze softverskog procesa, počev od zahteva korisnika, pa do razvoja programa. Zbog dominantne aktivnosti testiranja, najveći troškovi validacije se vrše za vreme i posle implementacije sistema.

Slika 1.1.1 Proces testiranja

Ako se i moduli i pod-sistema smatraju jedinicama (komponentama softvera), onda se proces validacije može podeliti u tri osnovne faze:

1. *Razvojno testiranje:* Komponente sistema se posebno testiraju od strane onih koji su ih i razvili, u toku procesa njihovog razvoja. Svaka komponenta se posebno testira.

Komponenta može biti vrlo jednostavna jedinica, koja obezbeđuje samo jednu funkciju, ali može biti i grupa ovakvih entiteta (npr. paket klasa). U ovoj fazi koriste se alati za automatsko testiranje, kao što je JUnit.

1. *Sistemsko testiranje*: Komponente sistema su integrisane i čine jedinstveni sistem. Testiranje sistema se vrši da bi se našle greške usled neodgovarajuće interakcije između komponenata i problema sa njihovim interfejsima. Testiranje sistema takođe

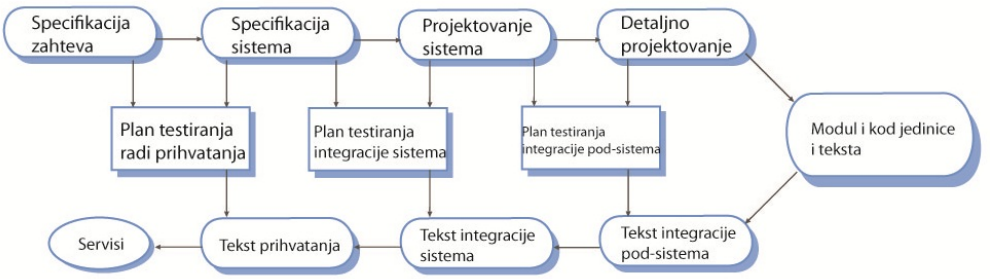
treba da pokaže da li sistem ostvaruje sve funkcionalne i nefunkcionalne zahteve (npr. performanse). Kod velikih sistema (kao što je ovaj prikazan na slici), ovaj proces testiranja sistema se može izvršavati u fazama, jer se prvo testiraju integrisan jedinice u modeli, moduli u podsisteme, a na kraju, u ceo sistem.

2. *Test prihvatanja:* Ovo je krajnja faza procesa testiranja, pre njegovog prihvatanja za operativnu upotrebu. Sistem se testira korišćenjem podataka koje je obezbedio korisnik (kupac). Test prihvatanja može da ukaže greške koje potiču od neadekvatnih zahteva na sistemskom nivou, što testiranje sa simuliranim podacima nisu mogla da pokažu. Testiranje prihvatanja treba da pokaže da sistem zadovoljava sve zahteve korisnika i ostvaruje zahtevane performanse.

PLAN TESTIRANJA

U stvarnosti, razvoj komponenata i proces testiranja su izmešani. Plan testiranja povezuje aktivnosti testiranja i razvoja.

U stvarnosti, razvoj komponenata i proces testiranja su izmešani. Programeri rade sopstvene testove sa svojim podacima i inkrementalno testiraju kod u toku njegovog razvoja. To je sa ekonomskog stanovišta ispravan pristup, jer programer najbolje poznaje komponentu koju je razvio, te je najbolja osoba i da generiše slučajeve (scenarije) za testiranja. U ekstremnom programiranju, testovi se razvijaju zajedno sa zahtevima, i pre nego što razvoj i počne. Ovo omogućava testerima i programerima da bolje razumeju zahteve i da obezbede da nema kašnjenja pri testiranju. Pri primeni softverskih procesa koji su vođeni planom (slika 2), testiranje se vrši u skladu sa planovima testiranja. Nezavistan tim testera pripreme te planove, a na osnovu dokumentacije sa specifikacijom sistema i projektnom rešenju. Slika 2 pokazuje kako planovi testiranja povezuju aktivnosti testiranja i razvoja



Slika 1.1.2 Faze testiranja u procesu razvoja vođenim planom

ALFA I BETA TESTIRANJE

Alfa testiranje vrši razvojni tim, sam, ili sa naručiocem softvera (kao test prihvatanja). Beta testiranje se vrši kod grupe odabranih korisnika sistema, tj. u njihovom radnom okruženju

Testovi prihvatanja se često nazivaju i „alfa testovima“. Alfa testiranje traje sve dok se naručilac softvera i razvojni tim sistema ne slože da razvijeni sistem zadovoljava postavljene zahteve.

U slučaju softverskih proizvoda (koji se prodaju na tržištu), koristi se i proces testiranja koji se naziva „beta testiranje“. Beta testiranje sistema se vrši kod grupe odabranih korisnika sistema, tj. u njihovom radnom okruženju. Oni šalju izveštaj razvojnom timu sistema. Na ovaj način se utvrđuju greške koje se javljaju u stvarnim uslovima rada sistema. Na osnovu ovih izveštaja, razvojni tim vrši promene u sistemu i proizvodi novu verziju ili za novo beta testiranje, ili za prodaju na tržištu.

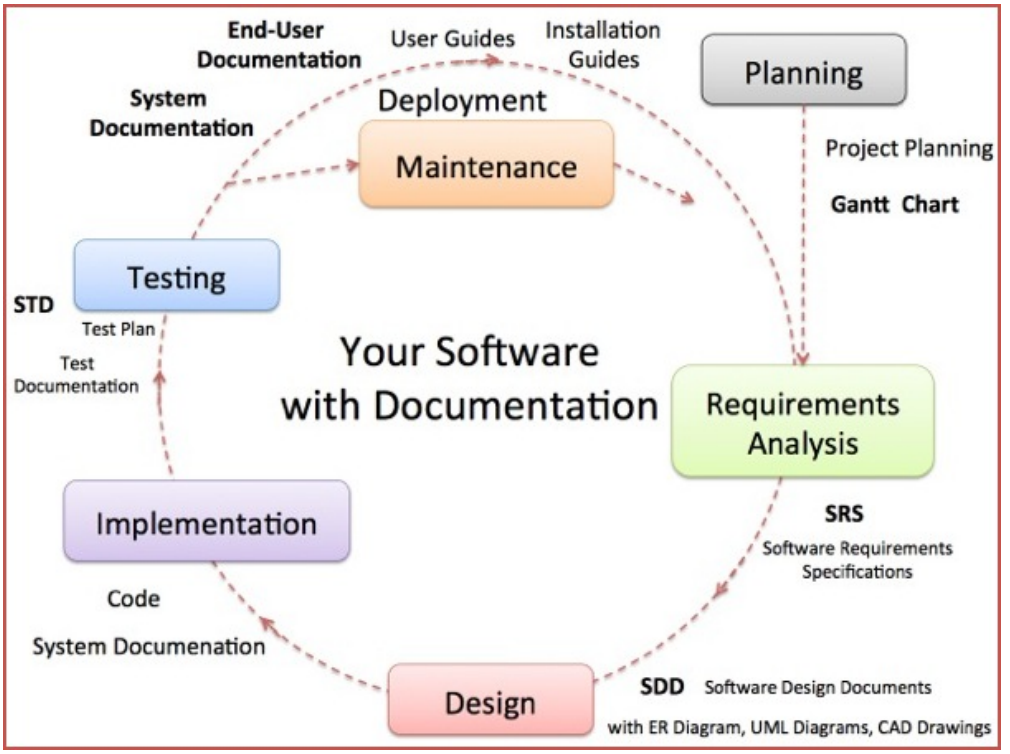
VERIFIKACIJA I VALIDACIJA SOFTVERA (VIDEO)

Validicija softvera - video klip 1

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER

Osnovne aktivnosti softverskih procesa



Slika 1.1.3

ZADATAK

Odgovorite na sledeća pitanja

1. Skicirati faze softverskog procesa poštujući redosled realizacije.
2. Šta predstavlja validacija softvera?

3. Definisati proces alfa i beta testiranja sistema za poslovanje banke.

Poglavlje 2

Model softverskog procesa

TRI OPŠTA MODELA SOFTVERSKIH PROCESA

Model softverskog procesa je uprošćeno predstavljanje nekog softverskog procesa.

Model softverskog procesa je uprošćeno predstavljanje nekog softverskog procesa. Svaki model procesa predstavlja neki proces iz neke posebne perspektive, te zbog toga, obezbeđuje samo delimične informacije o procesu, tj. informacije važne za tu perspektivu (npr. funkciju softvera). Zato se najpre daju vrlo uopšteni modeli softverskih procesa (tzv. „paradigme procesa“) koji se prikazuju iz perspektive arhitekture procesa. Oni prikazuju tok procesa sa aktivnostima, ali ne prikazuju i detalje o svakoj aktivnosti. Zbog toga, ovi opšti modeli predstavljaju samo apstrakcije procesa (uprošćeni prikaz) razvoja softvera. Njihova namena je da ukažu na različite pristupe u razvoju softvera. Oni su samo osnova za razvoj detaljnijih modela procesa.

Najčešće se navode sledeća tri opšta modela softevrskih procesa:

1. Model vodopada *(engl., the whaterfall model)*: On prikazuje osnovne aktivnosti procesa : specifikacija, razvoj, provera (validacija) i evolucija. Ove aktivnosti su prikazane kao posebne faze procesa, kao što su: specifikacija zahteva, projektovanje (dizajn) softvera, primena (implementacija), testiranje i dr. koje se jedna za drugom realizuju.

1. Inkrementalni (postepeni) razvoj*:* On povezuje aktivnosti specifikacije, razvoja, i provere, kao niz serija verzija (inkremenata) softvera, pri čemu svaka verzija dodaje određenu funkcionalnost na prethodnu verziju.
2. Softversko inženjerstvo zasnovano na višestrukoj upotrebljivosti *(eng.* Reuse-oriented software engineering*)*: Ovaj pristup se oslanja na korišćenje komponenata softevera koje se mogu višestruko koristiti. Ovim procesom se integrišu postojeće softverske komponente u sistem, umesto da se razvijaju nove.

Ovi modeli se često kombinuju jer se mogu međusobno dopunjavati, što je važno naročito kod razvoja velikih softverskih sistema. Za velike sisteme, moraju biti poznati zahtevi korisnika, definisani na formalni način i na osnovu toga, da se postavi arhitektura softvera (primena modela vodopada). Međutim, manji, podsistemi, se mogu razvijati i na drugačiji način. Neki, dobro definisani delovi, mogu se razvijati primenom modela vodopada. Drugi, koje je teško unapred precizno definisati, kao što je na primer korisnički interfejs, razvijaju se na inkrementalni način.

ZADATAK

Odgovorite na sledeća pitanja

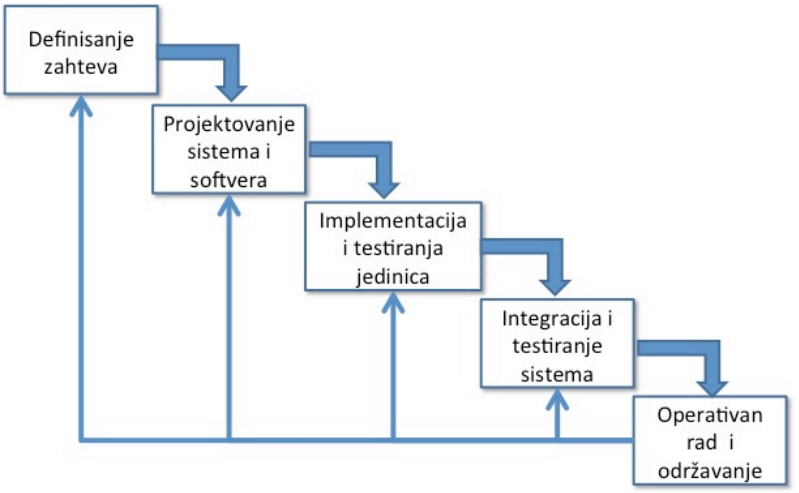
1. Šta predstavlja model softverskog procesa?
2. Koji modeli softverskog procesa postoje?

2.1 Model vodopada

FAZE RAZVOJA U MODELU VODOPADA

Model vodopada je planski vođeni proces jer se ceo proces mora planirati i odrediti termini za sve aktivnosti procesa, pre nego što počne njegovo izvršenje.

Model vodopada je primer planski vođenog procesa jer se ceo proces mora planirati i odrediti termini za sve aktivnosti procesa, pre nego što počne njegovo izvršenje (slika 1). Glavne faze modela vodopada su direktno povezane sa osnovnim aktivnostima razvoja softvera:



Slika 2.1.1 Model vodopada

1. Analiza i definisanje zahteva: Definišu se servisi sistema, ograničenja i ciljevi, definisani uz konsultaciju sa korisnicima sistema. Detaljnije se opisuju kao sistemska specifikacija.

2. Projektovanje sistema i softvera: Proces projektovanja sistema raspoređuje zahteve svim komponentama sistema i uspostavljanje celokupne arhitekture sistema, kao i utvrđivanje i opisivanje osnovnih apstrakcija softverskog sistema i njihove međuzavisnosti (relacije).

3. Implementacija i testiranje jedinica: U ovoj fazi je projektno rešenje softvera realizovano skupom programa ili programskih jedinica. Testiranje jedinica utvrđuje da li svaka jedinica ostvaruje svoju planiranu funkciju..

4. Integracija i testiranje sistema: Sve programske jedinice u ovoj fazi se integrišu u sistem i testiraju se kao kompletan sistem radi provere da li softver zadovoljava postavljene zahteve, tj. ostvaruje svoje funkcije i performanse. Posle ovog testiranja, softverski sistem se isporučuje kupcu.

5. Operativni rad i održavanje: Ovo je obično najduža faza životnog ciklusa softvera. Sistem je instalisan i pušten u operativni rad, tj. u upotrebu. Održavanje obuhvata ispravljanje grešaka koje nisu otkrivene u ranijim fazama životnog ciklusa, poboljšavanja primene programskih jedinica i poboljšanje usluga softverskog sistema u skladu sa novopostavljenim zahtevima.

KADA KORISTITI MODEL VODOPADA?

Model vodopada se koristi samo u slučajevima kada su svi zahtevi dobro definisani, jasni i stabilni.

Na kraju svake faze dobija se jedan ili više dokumenata koje neko mora da zvanično odobri, što je uslov za početak sledeće faze životnog ciklusa softverskog sistema. U praksi, radi ubrzanja procesa (što je uvek važan zahtev), faze se delimično preklapaju, te se specifikacija vrši i u toku kasnijih faza. Često se u fazi kodiranja vrše dorade projektnog rešenja softvera. Softverski proces nije linearan, već ima povratne sprege između faza procesa. Zbog toga, se vrši i naknadna dorada već urađenih dokumenata.

Kako proizvodnja i odobravanje dokumenata košta, to i svaka iteracija (vraćanje posla na neku od prethodnih faza) košta, jer zahteva ponekad, i značajan dodatni rad. Zbog toga, obično se posle nekoliko iteracija vrši „zamrzavanje“ urađenog posla na razvoju, kao na primer, izrada specifikacije, i kreće se na narednu fazu procesa razvoja softvera. Preostali problemi se ostavljaju za kasnije rešavanje, ili se ignorišu ili se programski naknadno rešavaju. Prevremeno zamrzavanje zahteva može dovesti do toga da sistem ne radi ono što korisnik od njega očekuje. Takođe, to može da dovede do loše struktuisanog sistema, jer se problemi rešavaju raznim ad-hoc programerskim korekcijama.

Model vodopada je u saglasnosti sa drugim inženjerskim modelima procesa i sa praksom da se posle svake faze dobija odgovarajuća dokumentacija. Ovo čini proces vidljivim od strane menadžera, jer oni mogu da ga nadziru i kontrolišu da li je u skladu sa planom. Međutim, nedostatak ovog pristupa je u nefleksibilosti procesa, jer se teško prilagođava promenama zahteva korisnika, što se u praksi često dešava.

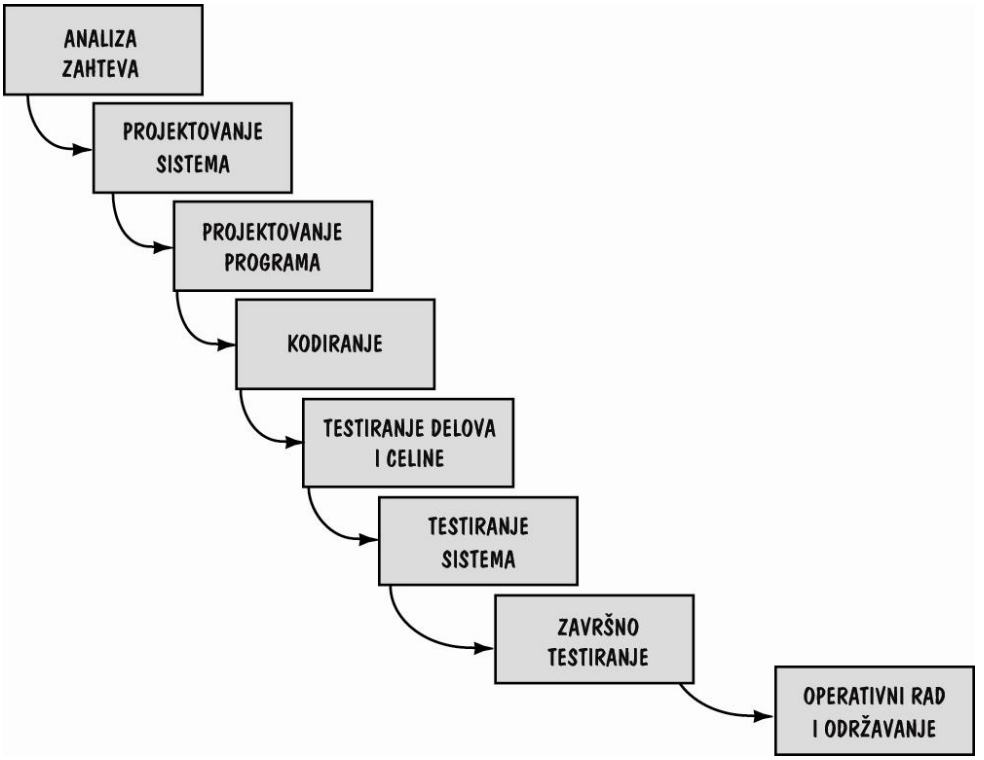
Po pravilu, model vodopada se koristi samo u slučajevima kada su svi zahtevi dobro definisani, jasni i stabilni (ne menjaju se za vreme razvoja softvera). I pored nedostataka, model vodopada je i dalje u čestoj upotrebi, jer se dobro uklapa modele upravljanja projektima.

Jedna posebna varijanta model vodopada je *formalan sistem razvoja.* U njemu, postoji matematički model specifikacije sistema, koji se matematičkim transformacijama, prebacuje iz faze u fazu, zadržavajući konzistentnost sistema sve do dobijanja izvršnog koda. Pod pretpostavkom da je matematički model ispravno postavljen, primena formalnog procesa razvoja, dolazi se do softvera koje 100% konzistentan sa definisanim zahtevima. Na ovaj način se obično razvija softver koji mora da garantuje bezbednost, pouzdanost i zadovoljenje zahteva sigurnosti.

PRIMER

Izbor pravog softverskog procesa

Sistem protiv blokiranja kočnica u automobilima
Ovo je sigurnosno kritičan sistem, tako da zahteva naprednu analizu pre implementacije. Svakako je potreban planski usmereni pristup razvoju sa pažljivo analiziranim zahtevima koji nisu podložni promenama. Model vodopada je stoga najprikladniji pristup za korišćenje, uz dodatak formalnih transformacija između različitih faza razvoja.



Slika 2.1.2

ZADATAK

Navesti primer korišćenja modela vodopada

Navesti jedan primer korišćenja modela vodopada i na izabranom primeru objasniti svaku od faza kao i aktivnosti svake faze.

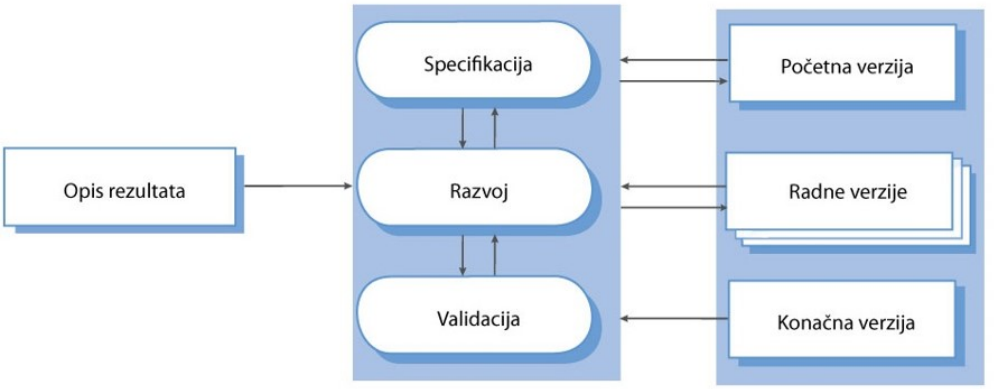
2.2 Model inkrementalnog razvoja

KONCEPT INKREMENTALNOG RAZVOJA SOFTVERA

Inkrementalni razvoj se zasniva na postupku postepenog razvoja novih inkremenata, tj. verzija softvera, i zamenom prethodno razvijenih verzija.

Inkrementalni razvoj se zasniva na postupku po kome se razvije neka početna implementacija softvera, koja se pokazuje korisniku. Na osnovu njegovih primedbi, napravi sa nova verzija softvera, i tako preko više verzija, dolazi se do konačne verzije, koja predstavlja razvijeni softver.

Inkrementalni razvoj, koji čini osnov agilnih pristupa u razvoju softvera, je bolji od modela vodopada za najveći broj poslovnih aplikacija, e-poslovanje i personalnih sistema. Inkrementalni razvoj odražava način kako mi često rešavamo probleme. Mi retko pravimo detaljan plan za budućnost, već idemo u budućnost pristupom korak-po-korak. Pri tome se često vraćamo nazad, da bi ispravili uočene greške. Inkrementalni razvoj softvera je jeftiniji i lakši za rad kada se često traže promene u softveru tokom njegovog razvoja.



Slika 2.2.1 Model inkrementalnog razvoja softvera

Svaki inkrement, tj. nova verzija softvera, sadrži neke od potrebnih funkcija (tj. funkcionalnost) koje traži korisnik. Prve verzije obično sadrže najvažnije funkcije, ili najhitnije zahtevane funkcije. Ovo omogućava da korisnik softvera može da u ranim fazama razvoja softvera oceni da li softver obezbeđuje funkcije koje se od njega zahtevaju. Ako se utvrdi da to softver ne obezbeđuje, trenutna verzija softvera se onda menja, a nova funkcija se ostavlja za neku kasniju verziju, tj. inkrement.

ŠTA SADRŽI JEDAN INKREMENT?

Inkrement sadrži novi deo softvera koji zadovoljava novu grupu zahteva koji su dodeljeni inkrementu, i tako razvijen inkrement se integriše sa prethodno razvijenim softverom.

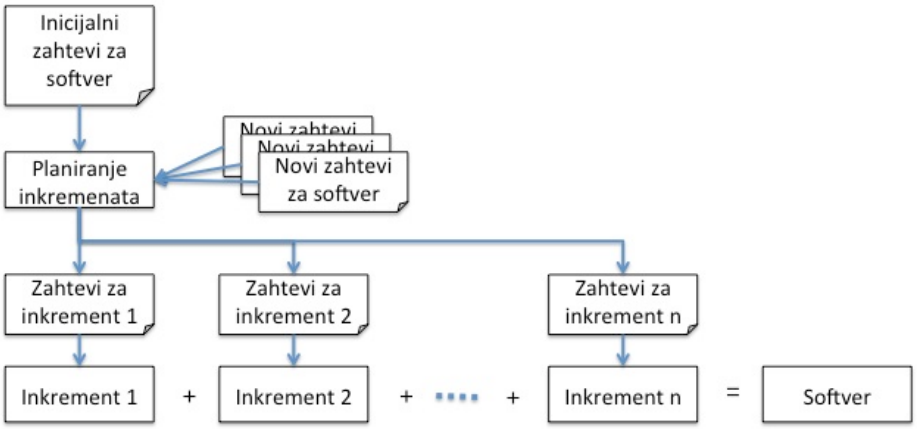
Kao i kod linearnog, tj. sekvencijalnog modela (modela vodopada), i kod inkrementalnog razvoja najpre se napravi lista zahteva koje softverski sistem treba da zadovolji. Razlika je samo u tome, što kod inkrementalnog razvoja, ti zahtevi se ne primenjuju u razvoju svi odjedanput već se podele po fazama razvoja softvera. U svakoj fazi se primenjuje samo skup zahteva dodeljen toj fazi. Svaka faza se realizuje određenim inkrementom (dodatkom) koji predstavlja novi deo softvera koji se dodaje prethodno razvijenom softevru.

Svaki inkrement, znači, sadrži svoj skup zahteva koje treba da primeni, tj. da se razvije softverski inkrement koji primenjuju te zahteve. Zato se svaki inkrement detaljno analizira, i softver koji se razvija u okviru jednog inkrementa prolazi kroz sve aktivnosti sekvencijalnog modela: projektovanje, razvoj (implementacija), testiranje, integracija se prethodno razvijenim inkrementima, tj. softverskim sistemom, njegovo testiranje i evolucija.

Za razvoj svakog inkrementa se planira jedno fiksno vreme (npr. dve nedelje). Ako se za to vreme ne primene svi planirani zahtevi za taj inkrement, oni se onda izostavljaju iz inkrementa razvijenog softvera i takav, reducirani inkrement se integriše sa prethodno razvijenim softverom. Neprimenjeni deo zahteva se ostavlja za naredni inkrement.

Ako se desi, da su svi planirani zahtevi realizovani pre planiranog vremena razvoja inkrementa, onda se dodaje deo zahteva planiranih za naredni inkrement, i tako proširem inkrement se razvija i integriše sa do tada razvijenim softverskim sistemom. Kupac softvera onda može da analizira tako dobijenu novu verziju softvera i da da neke nove primedbe i zahteve, koji se onda ubacuju u listu tahteva za naredne inkremente.

Da rezimiramo. Inkrement obezbeđuje novi deo softvera koji zadovoljava novu grupu zahteva. Sabiranjem tako razvijenih softverskih inkremenata, dobija se konačana softverski sistem (slika 2)

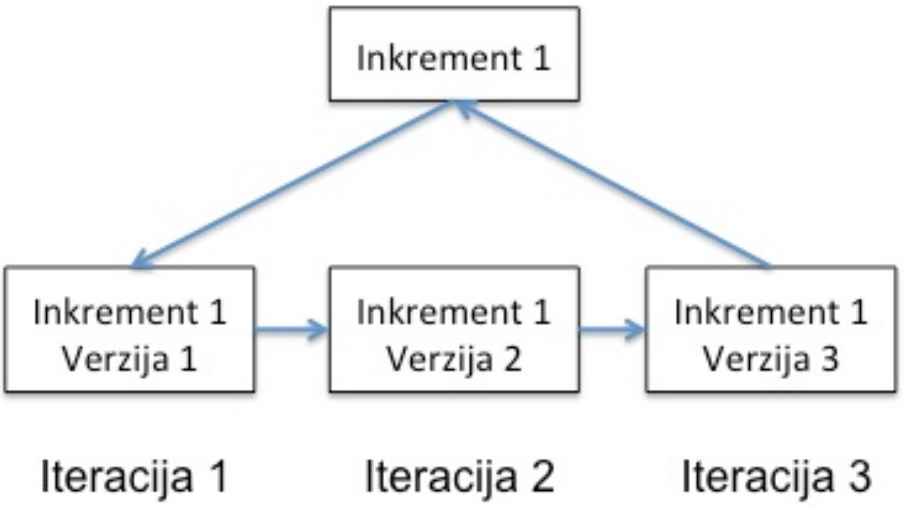


Slika 2.2.2 Inkrementalni razvoj softvera

INKREMENTALNI I ITERATIVNI RAZVOJ SOFTVERA

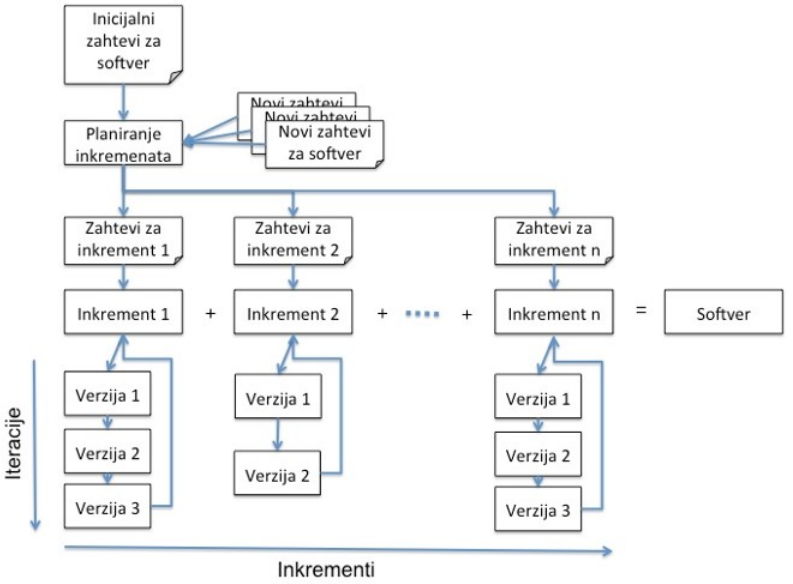
Inkrementalni razvoj preko više verzija, vršio poboljšanje siftvera. Svaki softverski inkrement se može i iterativno razvijati, pri čemu svaka verzija poboljšava prethodnu.

Iteracija znači ponavljanje. Iterativni razvoj znači da se proizvod poboljšava ponavljanjem nekih aktivnosti u procesu razvoja da bi se izvršila njegova poboljšanja. Iterativni razvoj se može primeniti i u slučaju razvoja inkrementa. Svaki inkrement se razvija u više verzija, pre čemu svaka naredna verzija prestavlja poboljšanje prethodne verzije (slika 3).



Slika 2.2.3 Iterativni razvoj jednog softverskog inkrementa

Korišćenje inkremenata i iteracija su bitni delovi više poznatinh inkrementalnih metoda razvoja softvera (npr. Srum, RUP). Slika 4 prikazije princip njihovog kombinovanja.



Slika 2.2.4 Inkrementalni i iterativni razvoj jednog softvera

DOBRE I LOŠE STRANE PRIMENE MODELA INKREMENTALNOG RAZVOJA

Inkrementalni razvoj obezbeđuje niže troškove razvoja, mišljenje korisnika i brži razvoj softvera. Nedostatci su u nevidljivosti procesa i postepeno urušavanje arhitekture sistema.

Inkrementalni razvoj, u odnosu na model vodopada, obezbeđuje sledeće prednosti:

1. *Niži troškovi realizacije zahteva korisnika.* Količina analiza i dokumenata koja treba da se ponovo uradi je znatno manja nego što je u slučaju primene modela vodopada.
2. *Lakše je obezbediti mišljenje korisnika na rezultat razvoja,* jer on može da komentariše demonstraciju radne verzije softvera i da vidi da li su njegovi zahtevi primenjeni. Korisnici se teško snalaze u dokumentaciji, te ne mogu na osnovu nje da ocenjuju da li su njihovi zahtevi zadovoljeni. Tek kada vide kako softver radi, mogu da komentarišu, a to inkrementalni razvoj obezbeđuje.
3. *Brža isporuka i instalacija softvera kod korisnika,* iako nema sve tražene funkcionalnosti (funkcije). Korisnici mogu softver da koriste ranije, iako bez svih traženih funkcija, nego što je to slučaj kod modela vodopada.

Inkrementalni razvoj se danas primenjuje u različitim oblicima pri razvoju aplikacija. Može se primeniti i u obliku planom vođenog procesa, kada se inkrementi (softverske verzije) unapred planiraju..

Kada se primenjuje u obliku agilnog razvoja softvera, rani inkrementi (verzije softvera) su identifikovani, a razvoj kasnijih inkremenata zavisi od napretka u razvoju, kao i od prioriteta koje korisnik nameće.

Iz perspektive upravljanja, inkrementalni pristup razvoju softvera ima dva problema:

1. *Proces nije vidljiv*. Menadžeri žele da imaju regularnost u dobijanju rezultata da bi ocenjivali napredak u radu. Međutim, kako se sistem brzo razvija, štedi sa na izradi dokumentacije za svaku verziju softvera.
2. *Struktura sistema ima tendenciju urušavanja sa dodavanjem novih inkremenata* (verzija). Bez ulaganja dodatnog novca i vremena da se sistem strukturno unapredi, regularne promene sistema vode ka njegovom postepenom urušavanju. Ubacivanje novih promena u sistemu, njegovo održavanje vremenom postaje skupo i teško.

U slučaju velikih sistema, neophodna je stabilnost arhitekture sistema i jasna odgovornost različitih timova koji rade na delovima sistema, a u odnosu na arhitekturu. To se mora unapred planirati, umesto inkrmentalno razvijati.

PRIMER

Model opšteg softverskog procesa koji najviše odgovara za upotrebu i za razvoj sledećih sistema:

Održavanje sistema virtualne realnosti
Ovo je sistem u kome će se zahtevi menjati i postojaće opsežne komponente korisničkog interfejsa. Inkrementalni razvoj sa, možda, nekim prototipom UI-a je najprikladniji model. Može se koristiti agilni proces.
Interaktivni sistem za planiranje putovanja koji omogućava korisnicima planiranje svojih putovanja za minimalnim uticajem promena
Ovaj sistem predstavlja sistem sa složenim korisničkim interfejsom koji mora biti stabilan i pouzdan. Inkrementalni razvoj je najprikladniji jer će podržati sve promene u zahtevima sistema

ZADATAK

Navesti primer korišćenja inkrementalnog modela razvoja softvera

Navesti jedan primer korišćenja inkrementalnog modela razvoja softvera i na izabranom primeru objasniti svaku od faza kao i aktivnosti svake faze.

2.3 Model razvoja upotrebom komponenata

RAZVOJ ZASNOVAN NA PONOVNOJ UPOTREBI KOMPONENATA

Razvoj zasnovan na ponovnoj upotrebi komponenata zasniva se na velikoj bazi višestruko upotrevljivih softverskih komponenti koje se biraju i integrišu u softverski sistem.

U većini softverskih projekata, prisutna je neka softverska komponenta koja je ranije razvijena. Ovo se obično dešava neformalno kada ljudi koji rade na projektu poznaju dizajn ili kod koji je sličan onome koji je tražen. Oni ga nalaze, modifikuju kao što je zahtevano i objedinjuju ga u sistem.

Razvoj zasnovan na ponovnoj upotrebi komponenata zasniva se na velikoj bazi ponovno korišćenih softverskih komponenti kojima se može pristupiti, kao i nekom integrativnom okviru za ove komponente. Ponekad ove komponente su sistemi (COTS ili Commercial-off-the-shelf sistemi) koji mogu biti korišćeni da obezbede posebnu funkcionalnost kao što je formatiranje teksta, numerički proračuni, itd.

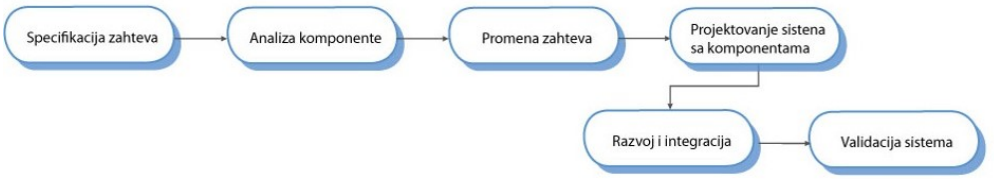
Dok su faza specifikacije početnih zahteva i faza validacije uporedive sa drugim procesima, međufaze u ovom procesu su različite. Ove faze su:.

1. *Analiza komponenti* – U skladu sa datom specifikacijom zahteva, sprovedena je pretraga za neophodnim komponentama. Obično, ne postoji potpuna podesivost i komponente koje mogu biti korišćene obezbeđuju samo deo zahtevane funkcionalnosti.
2. *Modifikacija zahteva* – Tokom ove faze, analizirani su zahtevi korišćenjem informacija o komponentama koje su pronađene. Oni su zatim modifikovani da budu u skladu sa dostupnim komponentama. Kada su modifikacije nemoguće, aktivnost analize komponenti može biti ponovno urađena da bi se pronašla alternativna rešenja.
3. *Projektovanje sistema sa komponentama* - Tokom ove faze, projektovan je strukturni okvir (framework) sistema ili je korišćen postojeći okvir. Projektanti uzimaju u obzir komponente koje mogu da se ponovno koriste i organizuju okvir da bude prilagođen njima.

4. *Razvoj i integracija* – Razvijen je softver koji ne može biti kupljen i komponente i COTS sistemi su integrisani da stvore sistem. Integracija sistema, u ovom modelu, pre može biti deo procesa razvoja nego odvojena aktivnost.

PREDNOSTI I NEDOSTACI RAZVOJA UPOTREBOM KOMPONENATA

Razvoj zasnovan na ponovnoj upotrebi komponenata smanjuje obim posla u razvoju softvera, što smanjuje cenu i vreme razvoja, ali ne uspeva da zadovolji sve zahteve korisnika.



Slika 2.3.1 Model razvoja softvera korišćenjem višestruko upotrebljivih komponenti

Model razvoja zasnovan na ponovnoj upotrebi komponenata ima očiglednu prednost da smanjuje količinu softvera koji treba da bude razvijen i takođe smanjuje troškove i rizike. To obično vodi ka bržoj isporuci softvera. Međutim, kompromisi zahteva su neizbežni i ovo može voditi ka sistemu koji neće u potpunosti ispuniti stvarne potrebe korisnika.

Postoje tri tipa komponenti koje se koriste:

1. *Veb servisi* koji se razvijaju u skladu sa standardima za servise i koji se mogu aktivirati iz udaljene lokacije.
2. *Kolekekcije objekata koji su razvijeni kao paketi* radi kasnije integrisanja u sisteme , a u okviru komponentinih okvira, kao što su .NET i J2EE.
3. *Samostalni softverski sistemi* koji se konfigurišu za upotrebu u određenom okruženju.

Razvoj zasnovan na ponovnoj upotrebi komponenata ima jasnu prednost u količini softvera koji treba razviti, jer je ona znatno smanjena, što se odražava na niže troškova i kraće vreme razvoja. Međutim, softverski sistem zbog upotrebe unapred definisanih komponenti, neminovno ne zadovoljava sve potrebe i zahteve korisnika. Takođe, gubi se i kontrola nad evolucijom softvera, sa upotrebom novih verzija komponenata, jer su one razvojene najčešće u drugim organizacijama.

PRIMER

Izbor pravog softverskog procesa

Sistem za računovodstvo univerziteta koji treba da zameni postojeći sistem

Ovo je sistem čiji su zahtevi prilično poznati i koji će se koristiti u okruženju u kombinaciji sa puno drugih sistema, kao što je sistem upravljanja privilegija za korišćenje sistema. Prema

tome, pristup koji zasniva na ponovnoj upotrebi komponenata je verovatno odgovarajući za ovo

ZADATAK

Navesti primer korišćenja modela razvoja softvera zasnovanog na ponovnoj upotrebi komponenata

1. Navesti jedan primer korišćenja modela razvoja softvera zasnovanog na ponovnoj upotrebi komponenata i na izabranom primeru objasniti svaku od faza kao i aktivnosti svake faze.

ZADACI ZA SAMOSTALNI RAD

Proverite stečeno znanje

Predložite model opšteg softverskog procesa koji najviše odgovara za upotrebu, za razvoj sledećih sistema:

1. Sistem za vođenje poslovanja arodroma koji treba da zameni postojeći sistem
2. Sistem za podršku u upravljanju vanrednim situacijama
3. Sistem za praćenje treninga i ishrane sportista
4. Sistem za online prodaju karata za bioskope, koncerte, pozorište
5. Sistem za podršku učešća građana u pametnim gradovima (smartParking, smartDrive...)

Poglavlje 3

Aktivnosti procesa

OSNOVNE AKTIVNOSTI SOFTVERSKIH PROCESA

Četiri osnovne aktivnosti softverskih procesa su : specifikacija, razvoj, validacija i evolucija

Stvarni softverski procesi su mešavina sekvenci tehničkih, kolaborativnih i upravljačkih aktivnosti sa ukupnim ciljem da specificiraju, projektuju, implemetiraju i testiraju neki softverski sistem. Softverski inženjeri u svom radu koriste različite softverske alate radi rada sa različitim tipovima dokumenata i uređivanja velike količine detaljnih informacija koji se stvara u projektu razvoja nekog velikog softverskog sistema.

Četiri osnovne aktivnosti softverskih procesa su :

1. Specifikacija softvera
2. Razvoj softvera
3. Validacija softvera i
4. Evolucija softvera

.

One se različito organizuju u različitim procesima razvoja softvera. U modelu vodopada, aktivnosti su organizovane redno (jedna posle druge), dok su pri inkrementalnom razvoju one izmešane. Kako se te aktivnosti realizuju, to zavisi od tipa projekta, ljudi, i organizacione strukture. Pri primeni agilne metodologije poznate kao Ekstremno programiranje, specifikacija se piše na karticama. Testovi se izvršavaju i razvijaju pre samog programa. Evolucija softvera može da obuhvati značajnu promenu strukture softvera

PRIMER - SISTEM ZA ONLINE IZNAJMLJIVANJE SKI OPREME

Sistem za online iznajmljivanje ski opreme

Opis softverskog sistema:
Ovo je sistem koji ima za cilj da obezbedi kupcu mogućnost da online rezerviše željenu ski opremu na određeni period. Prednost ovakvog softverskog sistema je ta sto će kupac moći da ima pregled dostupne opreme online, a pritom neće morati lično da dolazi do prodajnog objekta i vidi koja je oprema trenutno dostupna. Ciljna grupa ljudi koja bi koristila ovakav sistem jeste ona grupa ljudi koja se bavi skijanjem kako profesionalno tako i rekreativno.

Model softverskog procesa:
Model koji će se koristiti prilikom izrade ovog softverskog sistema je waterfalls ili (vodopad). Prvi korak je analiziranje kao i definisanje svrhe ovog softvera. Takodje je neophodno odrediti i stepen izvodljivosti realizacije definisanih zahteva jer ako su mogućnosti realizacije male onda će se odlučiti da li će se uopšte preći na sledeći korak ili ne. Nakon toga sledi projektovanje sistema i softvera. To je korak u kome se zahtevi raspoređuju svim komponentama sistema i uspostavlja se cela arhitektura sistema. Zatim sledi implementacija softvera odnosno njegovih programskih jedinica. Ovom koraku takođe pripada i proces testiranja svih programskih jedinica da bi se utvrdilo da li svaka programska jedinica ispunjava odgovarajuće funkcije. Nakon toga sledi korak u kome se sve programske jedinice integrišu u jednu i onda se vrši zavrsno testiranje odnosno testiranje celokupnog softvera zbog provere funkcionalnosti istog. Ukoliko je testiranje uspesno proteklo, sistem se isporučuje kupcu.
Poslednja faza u izradi softverskog sistema je operativni rad i održavanje. Pod ovim pojmovima se podrazumeva rad na poboljšanju softvera nakon njegovog plasiranja. Takođe ova faza služi i za otkrivanje nekih grešaka koje nismo uspeli da otkrijemo tokom faza u kojima smo testirali softver.

ZADATAK

Proverite stečeno znanje o aktivnostima procesa razvoja softvera

Za sisteme:

- Upravljanja poslovanjem univerziteta,
- Za kontrolu i upravljanje automatske letelice

izaberite model razvoja, identifikujte i objasnite aktivnosti procesa razvoja softvera.

3.1 Specifikacija softvera

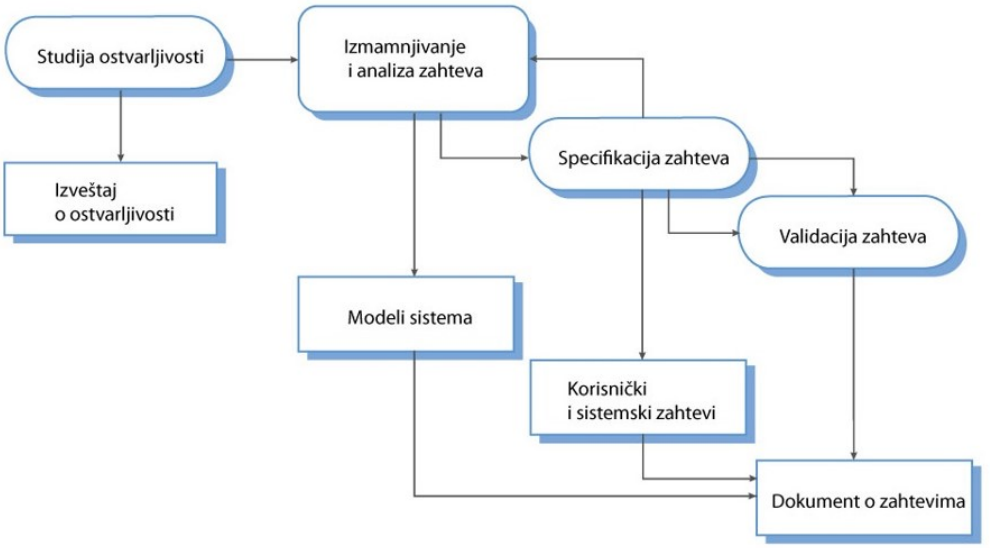
ŠTA JE SPECIFIKACIJA SOFTVERA?

Specifikacija softvera treba da ustanovi koji servisi su zahtevani od strane sistema i ograničenja rada i razvoja sistema

Specifikacija softvera treba da ustanovi koji servisi su zahtevani od strane sistema i ograničenja rada i razvoja sistema. Ova aktivnost se često zove inženjering zahteva. Inženjering zahteva je posebno kritična faza softverskog procesa zato što greške u ovoj fazi neizbežno vode kasnijim problemima u projektovanju i implementaciji sistema

Ovaj proces vodi ka stvaranju dokumenta zahteva koji je specifikacija za sistem. Zahtevi su u ovom dokumentu obično predstavljeni u dva nivoa detalja. Krajnim korisnicima i mušterijama

potreban je visoki nivo zahteva; inženjerima koji razvijaju sistem potrebna je jos detaljnija specifikacija sistema.



Slika 3.1.1 Proces specifikacije softvera

ČETIRI GLAVNE FAZE PROCESA INŽENJERINGA ZAHTEVA

Glavne faze procesa inženjeringa zahteva su: studija izvodljivosti, izvođenje i analiza zahteva, specifikacija zahteva i validacija zahteva.

Postoje četiri glavne faze procesa inženjeringa zahteva:

1. *Studija izvodljivosti* – Procena je napravljena u skladu sa tim da li identifikovane potrebe korisnika mogu biti zadovoljene korišćenjem postojećeg softvera i hardverskih tehnologija. Na osnovu studije se donosi odluka da li je predloženi sistem troškovno-efektivan iz poslovne tačke gledišta i da li može biti razvijen poštujući postojeća ograničenja budžeta. Studija izvodljivosti trebalo bi da bude relativno jeftina i treba da bude brzo urađena. Rezultat studije izvodljivosti treba da pruži informaciju koja će odlučiti da li ići napred sa mnogo detaljnijom analizom.
2. *Izvođenje i analiza zahteva* – Ovo je proces izvođenja zahteva sistema preko posmatranja postojećih sistema, diskusije sa potencijalnim korisnicima, analize zahteva, itd. Ovo može obuhvatiti razvoj jednog ili više različitih modela sistema i prototipova. Ovo pomaže analitičarima u razumevanju sistema za koji treba da se uradi specifikacija.
3. *Specifikacija zahteva* - Specifikacija zahteva je aktivnost prevođenja informacija skupljenih tokom analize u dokument koji definiše skup zahteva. U ovaj dokument mogu biti uključena dva tipa zahteva. Korisnički zahtevi su abstraktni iskazi zahteva sistema za mušteriju i krajnjeg korisnika; sistemski zahtevi su mnogo detaljniji opis funkcionalnosti koja treba da bude obezbeđena.
4. *Validacija zahteva* – Ova aktivnost proverava zahteve za realizmom, konzistencijom i potpunošću. Tokom ovog procesa, greške u dokumentaciji o zahtevima se neizbežno otkrivaju. Oni zbog toga moraju biti modifikovani da bi se korigovali ovi problemi.

Naravno, aktivnosti u procesu zahteva nisu jednostavno tretirane nekom strogom sekvencom.

PRIMER

Studija izvodljivosti

Polazna pretpostavka za proces softverskih zahteva predstavljaju jasno definisani strateški i operativni zahtevi organizacije koja ima potrebu za softverom. Pre inicijalizacije procesa, ponekad je neophodno realizovati studiju izvodljivosti da bi se sagledali svi faktori i ograničenja koji mogu uticati na proces softverskih zahteva, a kasnije i na ceo životni ciklus softvera. Ovim procesom najčešće rukovodi specijalista za softverske zahteve u okviru organizacije koja proizvodi softver. Naravno to je u slučaju da organizacija ima osobu sa takvim zaduženje. Na primer, u malim softverskim preduzećima vrlo je čest slučaj da je ista osoba angažovana u realizaciji većine aktivnosti u životnom ciklusu softvera. Sa aspekta učesnika u procesu, ovaj proces je multidisciplinarni i uključuje i eksperte iz domena problema i softverske inženjere.

ZADATAK

Definisati faze procesa inženjeringa zahteva sa detaljima

Definisati faze procesa inženjeringa zahteva sa detaljima za sistem poslovanja univerziteta.

3.2 Razvoj softvera

PROJEKTOVANJE SOFTVERA

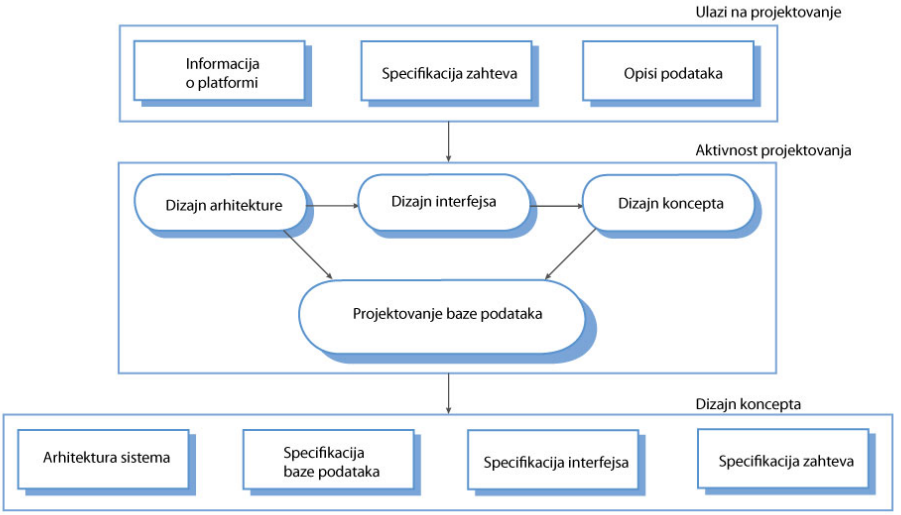
Projektovanje softvera je opis strukture softvera koji treba da bude razvijen, podataka koji su deo sistema, interfejsa između komponenti sistema, i ponekad korišćenih algoritama.

Implementaciona faza razvoja softvera je proces pretvaranja specifikacije sistema u izvršni sistem. Ona uvek obuhvata procese projektovanja i programiranja softvera, ali ako je korišćen evolucionarni pristup razvoja, on može takođe obuhvatiti prečišćavanje specifikacije softvera.

Projektovanje softvera je opis strukture softvera koji treba da bude razvijen, podataka koji su deo sistema, interfejsa između komponenti sistema, i ponekad korišćenih algoritama. Projektanti ne stižu do konačnog projekta trenutno, već razvijaju projekat kroz različite verzije. Proces projektovanja obuhvata dodavanje formalnosti i detalja dok se razvija projektno rešenje, sa čestim vraćanjem unazad da bi se korigovali prethodne verzije

Na slici je prikazan jedan apstraktan model ovog procesa koji prikazuje ulaze u proces projektovanja, aktivnosti procesa, i

proces projektovanja, aktivnosti procesa, i dokumenta koji su rezultat ovog procesa



Slika 3.2.1 Opšti model procesa projektovanja softvera

Dijagram na slici 1 ukazuje da se sve faze procesa projektovanja redno izvršavaju, što nije tako u praksi, jer su te aktivnosti izmešane. Povratne sprege koje povezuju dve faze izazivaju dodatan rad na promeni projektnog rešenja, jer je to neminovnost u svim projektima razvoja.

ČETIRI AKTIVNOSTI PROCESA PROJEKTOVANJA

Četiri osnovne aktivnosti procesa projektovanja softvera su: projektovanje arhitekture, projektovanje interfejsa, projektovanje komponenata i projektovanje baze podataka.

Najveći broj softvera komunicira sa drugim softverskim sistemima, kao što su: operativni sistemi, baze podataka, softver srednjeg sloja, i drugi aplikacioni sistemi. Sve to čini „softversku platformu“, tj. okruženje u kojoj softver radi. Informacija o softverskoj platformi je je značajan ulaz u proces projektovanja, jer na osnovu toga, projektanti treba da odluče kako na najbolji način da integrišu svoj softver sa softverskim okruženjem.

Specifikacija zahteva je opis funkcionalnosti softvera koju softver mora da obezbedi, kao i zahteve za performansama i zahteve za povezivanjem. Ako sistem obrađuje podatke, onda se podaci moraju opisati u specifikaciji platforme. U suprotnom, opis podataka bi trebalo da se obezbedi na ulazu u proces projektovanja, kako bi se definisala organizacija podataka u sistemu.

Slika 1 prikazuje četiri aktivnosti koje su deo procesa projektovanja informacionih sistema:

1. *Projektovanje arhitekture,* gde se utvrđuje ukupna struktura sistema, glavne komponente (podsistemi ili moduli), njihove veze, i način njihovog raspoređivanja.
2. *Projektovanje interfejsa*, kada se definišu interfejsi između komponenti sistema. Kada se definišu interfejsi, svaka komponenta se može nezavisno i istovremeno da razvija.

3. *Projektovanje komponenata,* kada se svaka komponenta sistema projektuje. To može biti i jednostavni iskaz o očekivanoj funkcionalnosti koja se mora primeniti, pri čemu se specifično projektno rešenje ostavlja da uradi programer. Može biti i lista promena koje se moraju izvršiti na komponenti ili može biti detaljno projektovani model. Projektovani model može se iskoristiti i za automatsku generaciju implementacije, tj. koda.
4. *Projektovanje baze podataka*, kada se projektuje struktura podataka i definiše kako se ona predstavlja u bazi podataka. Rad zavisi i od toga da li se radi nova baza, ili se menja neka postojeća baza podataka.

PROGRAMIRANJE

Programiranje, kao jedna individualizirana aktivnost koja najčešće ne sledi neki opšti model procesa. Najčešće, programeri vrše određeno testiranje koda koji su razvili

Slika 1 pokazuje i niz izlaznih rezultata aktivnosti procesa projektovanja koji se dosta razlikuju od sistema do sistema. *Za kritičke sisteme*, dokumentacija mora da definiše vrlo precizno i tačno opis sistema. Pri razvoju sistema na bazi modela, ovi izlazi su uglavnom u obliku dijagrama. U slučaju primene agilnih metoda, posebna dokumentacija i ne mora da postoji, jer se dokumentuje samo izvorni kod.

Programiranje, kao jedna individualizirana aktivnost koja najčešće ne sledi neki opšti model procesa. Neki programeri počinju rad prvo sa komponentama, koje razumeju, te ih i prve razviju. A onda prelaze na manje poznate komponente. Nasuprot ovome, drugi programeri ostavljaju poznate komponenta za kraj, jer znaju kako će ih razviti. Neki programeri vole da prvo definišu podatke u procesu, a onda razvijaju proces na bazi definisanih podataka.

Najčešće, programeri vrše određeno testiranje koda koji su razvili. Tim procesom tzv. dibaginga (engl. debugging) , tj. otklanjanja grešake iz koda. Testiranje defekata i dibaging su dva različita procesa. Testiranje utvrđuje postojanje defekata. Dibaging se usmerava da locira mesto greške i na ispravljanje grešaka u kodu.

Pre otklanjanja grešaka, programeri koriste alate za otklanjanje grešaka (debugging), jer oni omogućavaju praćenje trenutnih vrednosti promenljivih u programa pre kretanju od jedne dno druge linije u kodu.

PRIMER

Primer analize i razvoja softvera

Slika 3.2.2

ZADATAK

Faze razvoja softvera

Definisati faze procesa razvoja softvera sa detaljima za sistem poslovanja univerziteta.

3.3 Evolucija softvera

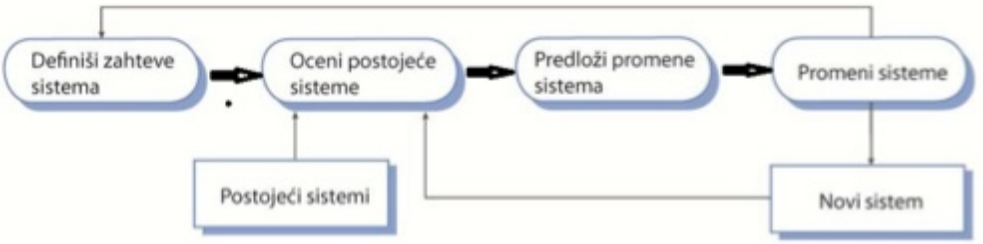
ŠTA JE EVOLUCIJA SOFTVERA?

Softversko inženjerstvo je evolucionarni proces gde se softver kontinualno menja tokom njegovog životnog veka da bi mogao da odgovori na promene zahteva i potreba korisnika.

Fleksibilnost softverskih sistema je jedan od glavnih razloga zašto je sve više i više sofvera objedinjeno u velike, kompleksne sisteme. Jednom kada je doneta odluka da se napravi hardver, vrlo je skupo praviti promene u projektovanom hardveru. Međutim za softver promene mogu biti napravljene u bilo kom trenutku tokom ili nakon razvoja sistema. Ove promene mogu biti vrlo skupe, ali još uvek mnogo jeftinije u poređenju sa odgovarajućim promenama u hardveru sistema.

Istorijski gledano, uvek je postojala linija razgraničenja između procesa softverskog razvoja i procesa evolucije (održavanja) softvera. Razvoj softvera se odnosio na kreativnu aktivnost gde je razvijan softverski sistem od početnog koncepta preko radnog sistema. Održavanje softvera je proces promene sistema jednom kada je sistem krenuo sa radom.

Mada su troškovi održavanja softvera često nekoliko puta veći od početnih troškova razvoja, procesi održavanja se smatraju manje izazovnim nego razvoj originalnog softvera. Ova linija razgraničenja postala je skoro nebitna. Malo softverskih sistema su kompletno novi sistemi i ima mnogo više smisla razmatrati razvoj i održavanje kao kontinuum. Pre nego dva odvojena procesa, mnogo je više realno razmatrati softverski inženjering kao evolucionarni proces gde se softver kontinualno menja tokom njegovog životnog veka da bi mogao da odgovori na promene zahteva i potreba korisnika. Ovaj evolucionarni proces je prikazan na slici.



Slika 3.3.1 Evolucija softvera

ZADATAK

Proverite stečeno znanje aktivnosti procesa

1. Navedite zašto je važno da se uoči razlika između razvoja zahteva korisnika i razvoja sistemskih zahteva?
2. Opišite glavne aktivnosti procesa projektovanja procesa i izlazne rezultate tih proesa. Upotrebom dijagrama, pokažite veze između izlaza iz ovih aktivnosti.
3. Za sisteme iz zadatka 3, definisati detaljnije aktivnosti procesa i skicirati odgovarajuće dijagrame

Poglavlje 4

Razvoj softvera u uslovima stalnih promena

KAKO RAZVIJATI SOFTVER U USLOVIMA STALNIH PROMENA?

Kako se u praksi često menjaju zahtevi u toku razvoja softvera, primenjuju se dva pristupa u razvoju: izrada prototipa softvera, radi provere zahteva, i inkrementalni razvoj softvera.

Kod svih projekata razvoja velikih sistema, promene zahteva su neminovnost, jer se menjaju i uslovi kod naručioca sistema. Zato je vrlo važno da se, bez obzira na primenjen model softverskog procesa, model procesa menja u skladu sa promenjenim zahtevima.

Promenjeni zahtevi obično uzrokuju ponavljanje nekih aktivnosti u razvoju, što povećava troškove razvoja. Koriste se dva pristupa smanjivanja ovih troškova:

1. ***Izbegavanje promena***, u slučajevima kada softverski proces sadrži aktivnosti u kojima se očekuju promene, pre nego što se zahteva značajniji ponovni rad. Na primer, pre nego što se pribegne promeni sistema koje zahtevaju veće troškove, razvije se poseban prototip pomoću koga se naručiocu pokazuje kako bi sistem radio, kada se izvrše njegove promene. Tek kada se naručilac složi sa očekivanim rezultatom, pristupa se aktivnostima kojima se menja stvarni sistem, radi ostvarivanja usaglašenih rezultata.
2. ***Tolerisanje promena***, neka forma inkrementalnog razvoja. Predložene promene se onda realizuju na inkrementu koji još nije razvijen. Na taj način, samo mali deo sistema (inkrement) je zahvaćen promenom, a ne i ceo sistem kada je proces .

tako projektovan da se promene mogu obavljati sa relativno niskim troškovima. U ovom slučaju, najčešće se primenjuje.

Imajući ovo u vidu, u praksi se primenjuju dva pristupa razvoju softvera u uslovima čestih promena zahteva:

1. ***Izrada prototipa sistema***, kojim se prototip sistema ili njegov deo brzo razvija radi provere zahteva naručioca i radi provere ostvarljivosti nekih projektnih rešenja. Promene sistema se odlažu sve dok se te promene ne verifikuju na prototipu. Na taj način se smanjuje broj zahteva za promenama posle isporuke softverskog sistema.
2. ***Inkrementalna isporuka***, kojim se naručiocu isporučuju inkrementi sistema radi komentarisanja i eksperimentisanja. To obuhvata i izbegavanje (prevremenih) promena i toleranciju promena. Time se izbegava prerano opredeljavanje da se nešto

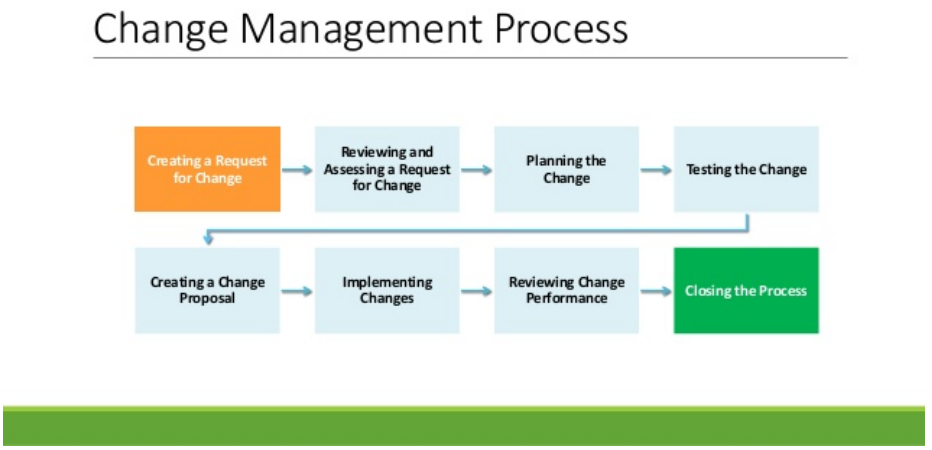
u sistemu menja, dok se to ne proveri na inkrementu. Takođe, umesto da promene obuhvate ceo sistem, one se odnose samo na inkrement, što je znatno jeftinije.

PRIMER - SISTEM ZA ONLINE GLEDANJE FILMOVA I SERIJA

Sistem za online gledanje filmova i serija.

Krajnji cilj sistema je da omogući korisniku pretraživanje i gledanje filmova i serija, ocenjivanje istih i dodavanje u listu omiljenih, kao i mogućnost korisnika da šalje zahteve administatoru sistema za postavljanje novog filma ili serije. Takođe sistem treba da obezbedi administratorima osnovne CRUD operacije nad filmovima i serijama. Sistem takođe prati najgledanije filmove/serije i postavlja ih na početnu stranicu. Pretraga se vrši po različitim kategorijama, kao što su naslov filma, godina izdavanja, žanr, rating itd.

Ako bi za ovaj sistem primenili inkrementalni model, jer nemamo precizno definisane zahteve omogućili bismo da se sistem koristi pre nego što su implementirane sve funkcionalnosti, prva faza razvoja bi trebalo da omogući CRUD funkcionalnost za filmove,serije i korisnike aplikacije, kao i pretragu i prikaz filmova i serija. Nove manje prioritetne funkcionalnosti bi se dodavale sa svakim novim inkrementom, kao što su na primer ocenjivanje filmova/serija, stavljanje u listu omiljenih, komentrarisenje, praćenje popularnosti. Ovako projektovan sistem bio bi u stanju da odgovori na promene koje nastaju usled izmene u zahtevima korisnika.



Slika 4.1.1

ZADATAK

Dati primer razvijanja softvera u uslovima stalnih promena primenom prototipa

Dati primer razvijanja softvera u uslovima stalnih promena primenom prototipa. Definisati aktivnosti i objasniti kako se ovakav razvoj bori sa promenama.

4.1 Primena prototipa softvera

ŠTA JE PROTOTIP SOFTVERA?

Prototip je početna verzija softverskog sistema koja se upotrebljava radi pokazivanja koncepata, probe projektnih opcija, i boljeg razumevanja problema i njegovih mogućih rešenja

Prototip je početna verzija softverskog sistema koja se upotrebljava radi pokazivanja koncepata, probe projektnih opcija, i boljeg razumevanja problema i njegovih mogućih rešenja. Brz, iterativan razvoj prototipa sa niskim troškovima, je neophodan, kako bi korisnici softverskog sistema mogli da eksperimentišu sa prototipom u ranim fazama softverskog procesa.

Softverski prototip se koristi u procesu razvoja softvera da bi se olakšala primena zahtevane promene:

1. *U procesu inženjeringa zahteva*, prototip može da pomogne da se izmame i potvrde sistemski zahtevi.
2. *U procesu projektovanja sistema*, prototip se koristi radi ispitivanja određenog softverskog rešenja i radi podrške projektovanog korisničkog interfejsa.

Prototipovi sistema pomažu korisnicima da vide koliko dobro sistem podržava njihov rad. Tada mogu da im se jave nove ideje za zahteve, a i da utvrde područja u kojima je softver slab ili jak. Tada oni predlažu nove zahteve. Pored toga, primena prototipova može da ukaže na greške i nedostatke u zahtevima koje su predloženi. Funkcija definisana u dokumentaciji, sama za sebe, može izgledati da je prikladna. Međutim, ista funkcija, kada se kombinuje sa drugim funkcijama, može se pokazati kao neprihvatljivom ili nekorektnom. U tom slučaju se menja specifikacija sistema kako bi bolje odražavala promene zahteva.

Prototip sistema se može koristiti u fazi projektovanja sistema radi sprovođenja eksperimenata i radi ocenjivanja ostvarljivosti predloženog projektnog rešenja. Primena brzih metoda radzvoja radi provere korisničkog interfejsa, na primer, je često u upotrebi.

PROCES RAZVOJA PROTOTIPA SOFTVERA

Proces razvoja prototipa određuje aktivnosti njegovog razvoja u skladu sa postvaljenim ciljem. , Ključno pitanje je šta prototip treba da obuhvati, a šta ne treba da bude u njemu.

Model procesa razvoja protoripa je prikazan da slici. Ciljevi razvoja prototipa bi trebalo da se jasno definišu na početku procesa. To može biti provera projektnog rešenje korisničkog interfejsa, ili validacija funkcionalnih zahteva sistema, ili razvoj sistema za proveru ostvarljivosti aplikacije. Isti prototip se može koristiti za više ciljeva. Bez jasnih ciljeva, može se postaviti pitanje namene prototipa.

Slika 4.2.1 Proces razvoja prototipa softvera

Pri razvoju prototipa, ključno pitanje je šta on treba da obuhvati, a šta ne treba da bude u njemu. Radi minimazacije troškova, prototip treba osloboditi nepotrebnih funkcija. Obično se izostavljavaju i zahtevi vezani za performanse, kao što je brzina sistema i korišćenje memorije. I otklanjanje grešaka može da bude izostavljeno, ako je cilj, na primer, provera dizajna korisničkog interfejsa.

Krajnja faza procesa je evaluacija prototipa, koja se vrši na bazi postavljenih ciljeva prototipa. Vremenom, korišćenjem sistema, korisnici otkrivaju greške u zahtevima i ispuštene zahteve.

PROTOTIP NIJE SOFTVER ZA ISPORUKU

Ponekad, menadžment vrši pritisak da razvojni tim pretvori protoip u konačno rešenje, da bi se uštedelo na vremu i novcu, što nije najčešće prihvatljivo

Jedan od problema rada sa protipom je što se on ne koristi isto kao i konačan sistem. Ako je prototip spor, korisnici će ga koristiti drugačije (idu skraćenicama) nego što će kasnije koristiti konačan sistem, koji je brži. To može da dovede do ispuštanja iz vida (i provere) nekih svojstava softvera u fazi ispitivanja prototipa, a koji se manifestuju pri radu konačnog sistema.

Ponekad, menadžment vrši pritisak da razvojni tim pretvori protoip u konačno rešenje, da bi se uštedelo na vremu i novcu, što nije najčešće prihvatljivo:

1. Najčešće je nemoguće podesiti prototip tako da zadovoljava nefunkcionalne zahteve, kao što su performanse, bezbednost, robusnost i pouzdanost, jer se na njih ne obraća pažnja pri razvoju prototipa.
2. Brze promene u toku razvoja, neophodno dovode do prototipa bez dokumentacije. Postoji samo kod. To nije dovoljno dobro za kasnije održavanje sistema.
3. Promene urađene na prototipu, najčešće kasnije negativno utiču na strukturu sistema. Sistem postaje težak i skup za održavanje.
4. Pri razvoju prototipa obično se ne koriste standardi kvaliteta organizacije.

Prototipovi ne moraju uvek da budu u vidu izvršnog koda. Mogu biti i samo na papiru, kao na primer, u slučaju provere dizajna korisničkog interfejsa, što znatno ubrzava proces i smanjuje cenu. I interakcija sa korisnikom, može da bude simulirana a ne stvarna, tako da odgovore korisnika prima čovek, a ne sistem.

KADA I KAKO KORISTITI PROTOTIPOVE?

Prototipovi se najčešće koriste da bi se proverili i verifikovali zahtevi ili da bise testirala tehnologija koja će se koristiti pri razvoju softvera.

Ovde ćemo rezimirato do sada izloženo. Najšće se prototipovi koriste u dva slučaja pri razvoju softvera:

1. **Utvrđivanje zahteva korisnika:** Pokazivanjem prototipa softvera korisniku, daje mu se prilika da potvrdi d ali je ton što očekuje ili da dopuni svoje zahteve. Tada se često utvrđuju suprostavljeni i nejasni zahtevi
2. **Procenjivanje tehničkih i arhitektonskih rizika:** Pri projektovanju softvera se najčešće koriste projektni šabloni. Međutim, kod složenih softvera njihova primena nije uvek moguća. Primena prototipa je neophodna d abi se testirala ipostavljena arhitektura softverskog sistema. U slučaju primene novih tehnologija, na ovaj način se proverava njena upotrebljivost u slučaju softvera koji se razvija.

Pri razzvoju softvera, prototipovi se koriste u dva oblika kao:

a) model procesa i kao

b) tehničko rešenje, tj. tehnologija.

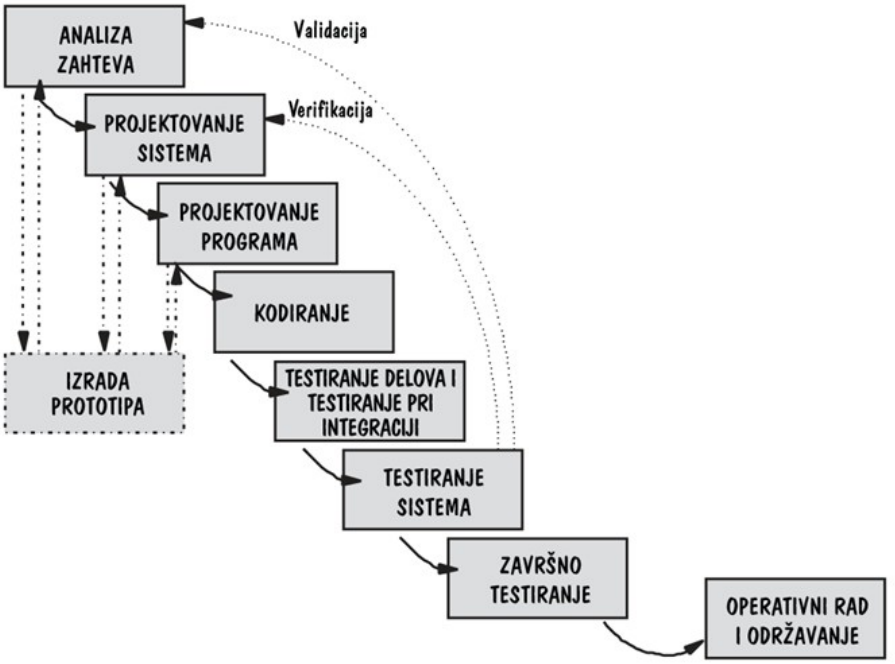
Ako se prototip koristi radi modelovanja procesa, onda se za svaku aktivnost procesa razvoja softvera koriste prototipova, kaoo bi se proverilo da li se dobija ono što se želi. Kada se to dobije, prelazi se na sledeću aktivnost, itd. Na kraju, posle poslednje aktivnosti, vrši se isporuka softvera razvijenog na ovakav način.

Ako se, pak, prototip koristi da bi sestirala primenjena tehnologija, onda se on koristi unutar nekog od modela procesa. RUP i spiralni model razvoja koriste prototipove unutar svog modela procesa razzvoja softvera. Kada se prototipom izvrši provera tehnologije, on se odbacije, i za razvoj softvrea se ne koristi više prototip, već se razvija softver za korisnika.

Prototipovi se retko koriste kao modeli procesa, jer su retki slučaji da su zahtevi potpuno nepoznati ili da su postavljenje rakhitekture toliko rizične i nepoznate d a ih je potrebno proveriti.Mnogo ćešće se prototipovi koriste radi provere zahteve i tehnologija.

PRIMER

Primena prototipa softvera



Slika 4.2.2

Kreiranje prototipova za sistem podrazumeva:

Throwaway Prototyping

- podrazumeva kreiranje prototipa koji će u dogledno vreme biti odbačen
- nakon preliminarne faze zahteva, konstruiše se jednostavniji radni model sistema radi ilustracije korisniku
- izrada radnog modela različitih delova sistema u ranoj fazi razvoja
- može da se uradi veoma brzo
- testira se i User Interface

Evolutionary Prototyping (Breadboard prototyping)

- izgradnja vrlo robusnog prototipa na strukturiran način i njegovo konstantno usavršavanje
- gradi se samo na osnovu onih zahteva koji su dobro razmotreni i prihvaćeni
- dodavanje novih funkcionalnosti, evaluacija kroz različita operativna okruženja

ZADATAK

Provera naučenog

1. Objasnite zašto su promene neizbežne pri razvoju složenih sistema i dajte primere (pored korišćenja prototipa i inkrementalne isporuke) aktivnosti softerskog procesa koje pomažu da se predvide promene, kao i da se razvije softver koji je otporniji na promene.

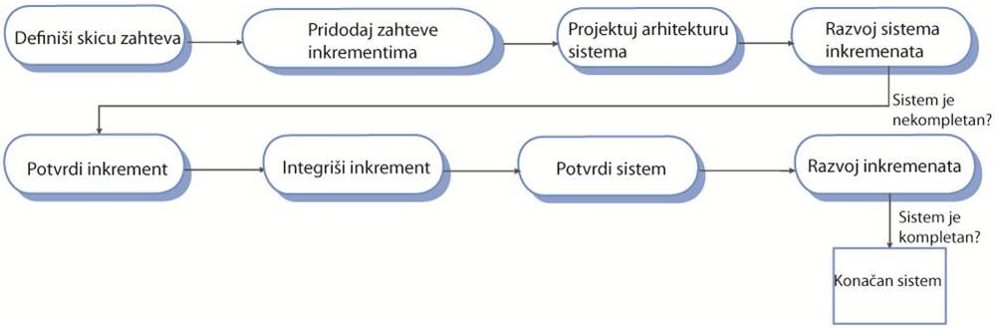
2. Objasnite zašto razvoj sistema u vidu prototipa ne treba koristiti pri razvoju produkcionih sistema.

4.2 Inkrementalna isporuka softvera

ŠTA JE INKREMENTALNA ISPORUKA SOFTVERA?

Inkrement je razvijeni deo sistema kojim se proširuje prethodno razvijeni softver i onda isporučuje kupcu. Detaljno se planira inkrement koji je u razvoju i njegovi zahtevi su konačni.

Inkrementalna isporuka softvera je pristup u razvoju softvera koja omogućava da se kupcima softvera isporučuju razvijeni delovi sistema (inkrementi), koji se onda instaliraju i puštaju u operativnu upotrebu. Na taj način, kupci utvrđuju servise koje treba da im obezbedi sistem. Oni određuju prioritetne servise, a koji su im najmanje važni. Onda se definiše određeni broj inkremenata, tako da svaki od njih obezbeđuje određen podskup funkcija sistema. Prvo se razvijaju inkrementi (moduli) sa najvećim prioritetom.



Slika 4.3.1 Inkrementalna isporuka softvera

Promene zahteva se prihvataju za planirane, a još ne razvijene inkremente, ali se ne prihvataju zahtevi koji se odnose na tekući inkrement. Zato se njegov razvoj mora detaljno da isplanira.

PREDNOSTI I NEDOSTACI INKREMENTALNE ISPORUKE SOFTVERA

Inkrementalni razvoj nudi niz prednosti: kupac ranije dobija softver za prioritetne funkcije i dozvoljava lako definisanje novih zahteva . Međutim, ovaj pristup ima i nedostatke.

Inkrementalna isporuka softvera nudi niz prednosti:

1. Kupci mogu da koriste prve inkremente kao prototipove i steknu iskustvo, koje koriste pri definisanju zahteva za kasnije inkremente. Za razliku od prototipova, to su delovi

stvarnog sistema, te nema potreba da ponovo uče njegovo korišćenje, kao u slučaju korišćenja prototipova.
2. Kupci ne moraju da čekaju da se razvije ceo sistem da bi im bio isporučen, te ranije dobijaju korist od korišćenja delova sistema. Obično prvi inkrement repava njihove najvažnije zahteve.
3. Proces omogućava relativno lako ubacivanje novih promena u sistem.
4. Kako se prvo isporučuju inkrementi sa funkcijama najvećeg prioriteta, to oni i dobijaju i najdetaljnije testiranje, te taj, najvažniji deo sistema onda im najmanje zadaje probleme.

Međutim, inkrementalna isporuka softvera ima i nekih nedostataka:

1. Najveći broj sistema zahtevaju ispunjenje osnovnih funkcija, a koje obezbeđuju pojedini delovi sistema. Kako zahtevi nisu detaljno definisani sve dok se jedan inkrement ne primeni, teško je da se utvrde osnovne, zajedničke funkcije koje će biti realizovane od strane svih inkremenata.
2. Otežan je iterativni razvoj kada se razvija novi sistem. Korisnici žele celu funkcionalnost starog sistema i često nisu voljni da eksperimentišu sa nekompletnim novim sistemom. Zato, teško je dobiti korisne odzive korisnika sistema.
3. Srž iterativnog razvoja je u paralelizmu definisanja specifikacije i razvoja softvera. Međutim, to je u suprotnosti sa praksom ugovaranja i plaćanja kada se ugovara i plaća kompletan sistem. Kod inkrementalnog pristupa, nema specifikacije kompletnog sistema, sve dok se ne specifikacija ne pripremi i za poslednji inkrement. To zahteva novu formu ugovora, koje najčešće ne odgovaraju velikim državnim agencijama.

VELIKI SISTEMI I INKREMENTALNA ISPORUKA SOFTVERA

Kod nekih velikih softverskoh sistema, inkrmentalni pristup razvoju nije i najbolji pristup.

Kod nekih velikih softverskoh sistema, inkrmentalni pristup razvoju nije i najbolji pristup. To je slučaj kada se sistem razvija na više lokacija, ili u slučaju pojedinih ugrađenih sistema kod kojih softver zavisi od razvoja hardvera, i u slučaju kritičkih sistema, kod kojih svi zahtevi se moraju analizirati jedinstveno radi provere interakcija, a u cilju provere bezbednosti i sigurnosti sistema. U ovim slučajevima, treba koristiti proces u kome se koristi iterativno razvijen prototip sistema kao platforma za ekperimentisanje sa sistemskim zahtevima i projektnim rešenjima. Na osnovu tako stečenog iskustva, utvrđuju se konačni zahtevi sistema.

PRIMER

Informacioni sistem univerziteta

Kako razvoj informacionog sistema univerziteta obuhvata kompleksan skup funkcionalnosti, primenom inkrementalne isporuke softvera životni ciklus se sastoji od nekoliko sekvencijalnih iteracija. Svaka iteracija ima za cilj da obezbedi pojedinačnu funkcionalnost po modulima

(eProfesor, eStudent, studentska služba...).
Svaka iteracija proizvodi izvršne verzije pojedinačnog modula i svaka iteracija uključuje integraciju i testiranje.
Cilj završetka iteracije je stabilan, integrisan, testiran deo celokupnog softverskog sistema koji se gradi.

Ovakav razvoj će dozvoliti da u uslovima stalnih promena sistem može zamenom pojedinačnih komponenti da odgovori na sve zahteve bez ugrožavanja prethodno funkcionalnih i istestiranih sistema.

ZADATAK

Proverite svoje razumevanje primene inkrementalnih metoda razvoja softvera

1. Objasnite zašto je inkrementalni razvoj najefikasniji pristup za razvoj poslovnih softverskih sistema.
2. Zašto je ovaj model manje prikladan za inženjering sistema u realnom vremenu?
3. Na primeru sistema za kontrolu saobraćaja i upravljanja poslovanjem aerodroma definisati prednosti i nedostatke inkrementalne isporuke softvera.

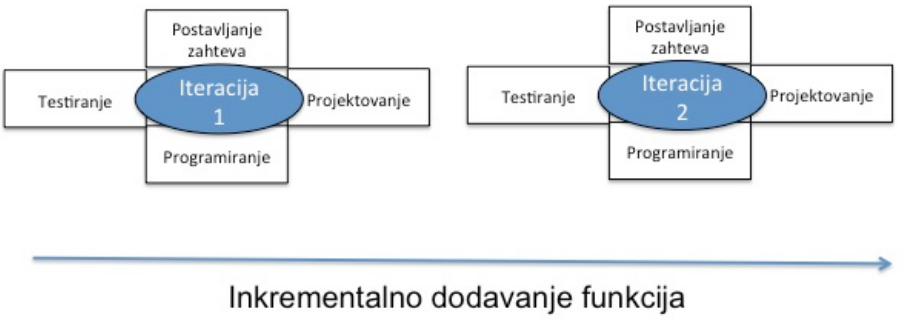
4.3 Agilni model procesa razvoja softvera

OSNOVNE VREDNOSTI AGILNOG MODELA

Inkrementalni i iterativni proces se primenjuje primenom specifičnih principa agilnosti: važan je tim, softver koji radi, saradnja sa kupcem i reagovanje na promene.

Agilni model procesa razvoja softvera se oslanja na inkrementalan i iterativan metod razvoja softvera. Agilni model ima za cilj da omogući: razvoj pouzdanog softvera, i to brzo, eliminisanje nepotrebnih aktivnosti .

U agilnom modelu procesa softver se inkrementalno razvija. Svaki inkrement dodaje novu funkcionalnost softvera. Pri razvoju svakog inkrementa softvera, iterartvno se primenjuju uobičajene aktivnosti razvoja softvera: postavljanje zahteva, projektovanje softverskog rešenja, programiranje i testiranje softverskog inkrementa (slika 1). Pri ovome, ne daje se poseban fokus ni na jednu od ovih aktivnosti.One se realizuju delimično i paralelno i uz iterativno ponavljanje, od strane tima programera.



Slika 4.4.1 Agilni model procesa razvoja softvera

Agilni model razvoja softvera se oslanja na sledeće vrednosti:

1. *Programeri i njihove interakcije su važniji od procesa i alata*. Važan je složan tim koji sarađuje, te je vrlo važno da se formira.
2. *Softver koji rad je važniji od kompletnosti dokumantacije*. Dokumentacija je važna, ali ako je ima previše, postaje skupa a teško je redovno ažurirati. Dovoljno je formirati manje detaljnu dokumentaciju.
3. *Saradnja sa kupcem je važnija nego ugovaranje*. Važni su komentari kupca i njihov rad sa timom programera. Na taj način se definišu buduće iteracije.
4. *Reagovanje na promene je važnije od postavljenog plana*. Planovi moraju da se fleksibilno menjaju. Pr.imeruje se planiranje na nivoa: detaljan nedeljni plan, plan iteracije, ukupan plan

PRINCIPI AGILNIH PROCESA

Vrednosti agilnih procesa se postižu primenom 12 principa agilnih procesa.

Pri primeni agilnog modela, postavljaju se sledeća pitanja: *Koliko planiranja? Do koje mere treba slediti planiran proces? U kom obimu treba priprmiti dokumentaciju analize i projektovanja kao i generisanisanih modela*? Potrebno je onoliko dokumentacije i procesa koliko je nephodno da bi se zadovoljili navedene vrednosti aglnih modela. Te vrednosti se ostvaruju primenom sledećih principa agilnosti:

- Najveći prioritet se daje zadovoljenju kupca što bržom isporukom softvera, i to stalno u toku procesa njegovog razvoja.
- Prihvataju se promene zahteva i zadnjem momentu. Agilni procesi prihvataju promene radi obezbeđenja konkuretnosti prednosti kupcu.
- Treba proizvoditi novi softver koji radi u što kraćim vremenskim intervalima, od par nedelja do par meseci.
- Poslovni ljudi i programeri treba da svakodnevno rade zajedno tokom celog trajanja projekta.

- U projekat uključiti motivisane pojedince. Treba im dati potrebno okruženje i podršku koju traže, kao i ukazati im poverenje da će posao uraditi.

- Najefikasniji i najefektniji način razmene informacija unutar tima je direktni razgovor njegovih članova.
- Mere napretka u razvoju je softver koji radi.
- Agilni procesi promovišu održivi razvoj.
- Poklanja se stalna pažnja postizanju tehničke izvršnosti i agilnosti u postizanju dobrog projektnog rešenja.
- Teži se jednostavnim rešenjima, koji zadovoljavaju zahteve korisnika.
- Najbolja arhitektura, specifikacija zahteva, i projektno rešenje se dobija od samoorganizovanog tima. Tim deli odgovornost.
- Povremeno, tim analizira svoj rad analizirajući kako da bude efektniji, te se međusobno usaglaža

PRIMER

Primer agilnog razvoja

Billiard pro je online bilijar igrica koju mogu da igraju prijavljeni korisnici. Ukoliko nemaju nalog moraju napraviti jedan. Igrač ima pristup svom profilu, prodavnici, ima opciju da pokrene igru i može videti koliko ima pobeda (W) i koliko ima čipova (C). Igrač u toku partije ima pravila koja mora da poštuje, ima mogućnost predaje meča, kao i opciju da se izloguje sa svog profila ili da menja podatke profila. Igrač može da primeni svoju dekoraciju stola koju je kupio u prodavnici. Igrač ima pregled statistike, listu od top 100 igrača rangiranu po broju pobeda (W). Sto za kojim se igra ima malu površinu podloge na kojoj se igra, a malo veće kugle koje su žute i crvene boje. Uslovi za igru na tom stolu su 50(C) i pobednik nosi 100(C) i 50(W).
Model koji se primenjuje za realizaciju ove aplikacije je agilni model jer primenjuje iterativno 4 uobičajene aktivnosti razvoja softvera: postavljanje zahteva, projektovanje, programiranje i testiranje, a svaki inkrement će voditi ka novoj funkcionalnosti aplikacije, jer ona ne može biti unapred određena, već uvek ima mesta za neki dodatak.
Prikupljanje zahteva:
Kako u agilnom modelu ne moramo da imamo strogo određene zahteve, prikupljamo zahteve koji su nam dovoljni za početak kako pi napravili prvi prototip koji bi kasnije pokazali klijentu i na osnovu tog prototipa on nama daje dodatne zahteve i/ili predefiniše trenutne zahteve, u agilnom modelu je moguće izvršiti promene u bilo kom trenutku.
Projektovanje:
Projektovanje se vrši u Power Designer-u, pravljenjem UML dijagrama omogućavamo klijentima da bolje shvate kako aplikacija funkcioniše ako im je dokumentacija aplikacije manje razumljiva.
Programiranje:
Na osnovu UML dijagrama koji smo isprojektovali pisemo kod aplikacije, prateći sve zahteve korisnika.
Testiranje:
Vrši se provera koda, da li aplikacija ispunjava sve zahteve korisnika, rešavaju se greške ako ih ima.
Nakon testiranja se vrši isporuka prototipa klijentu kako bi on utvrdio nove zahteve ako ih ima i dao nam predloge za dizajn aplikacije, u koliko želi da ih menja, kao i korekciju prethodno datih zahteva.

ZADATAK

Primena agilnog razvoja softvera

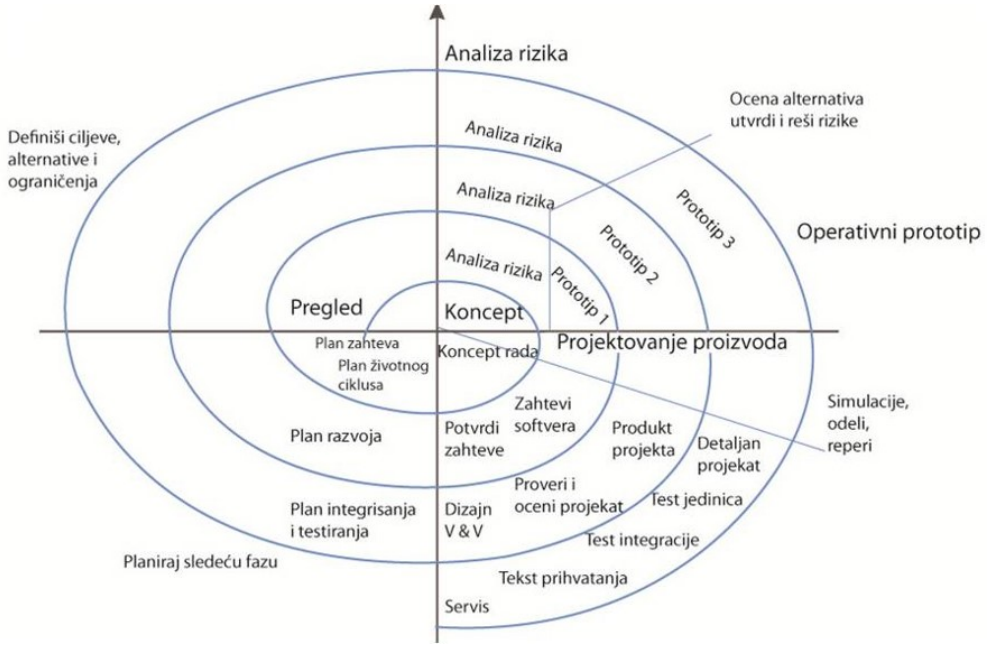
Dati primer i objasniti iteracije u razvoju softvera agilnom metodom. Obrazložiti zašto je za navedeni primer najbolje upotrebiti agilni razvoj softvera.

4.4 Spiralni model razvoja

ŠTA JE SPIRALNI MODEL RAZVOJA?

Softverski proces je predstavljen kao spirala pre nego sekvenca aktivnosti. Svaka petlja u spiralnom modelu predstavlja fazu procesa.

Spiralni model softverskog procesa (prikazan na slici 1) prvobitno je bio predložen od Boehm-a (1998) i sada je naširoko poznat. Proces je predstavljen kao spirala pre nego sekvenca aktivnosti sa povratnom spregom*. Svaka petlja u spiralnom modelu predstavlja fazu procesa*. Prema tome, krajnja unutrašnja petlja može se odnositi na izvodljivost sistema, sledeća petlja na definiciju zahteva sistema, sledeća petlja na projektovanje sistema, itd



Slika 4.5.1 Spiralni model razvoja softvera

ČETIRI SEKCIJE PETLJE U SPIRALI MODELI

Važna razlika između spiralnog modela i drugih modela softverskih procesa je eksplicitno razmatranje rizika u spiralnom modelu

Svaka petlja u spirali je podeljena u četiri sekcije kao što je prikazano na slici 3.7. Sekcije spiralnog modela su:

1. **Definisanje ciljeva** – Definisani su specifični ciljevi za ovu fazu projekta. Identifikovana su ograničenja procesa i proizvoda i skiciran je detaljni plan upravljanja. Identifikovani su rizici. Mogu biti planirane alternativne strategije zavisno od rizika.
2. **Određivanje i redukcija rizika** - Za svaki od identifikovanih projekata rizika, izvedena je detaljna analiza. Da bi redukovali rizik uvedeni su koraci. Na primer, ako postoji rizik za koji su zahtevi nezadovoljavajući, može biti razvijen prototip sistema.
3. **Razvoj i validacija** – Nakon proračuna rizika, izabran je razvojni model za sistem. Na primer, ako su rizici korisničkog interfejsa dominantni, odgovarajući razvojni model može biti evolucionarni prototip. Ako su rizici sigurnosti dominantni, može biti pogodan razvoj baziran na formalnoj transformaciji, itd. Model vodopada može biti najpogodniji razvojni model ako je glavni identifikovani rizik integracija pod-sistema.
4. **Planiranje** – Projekat je razmotren i doneta je odluka da li nastaviti sa narednom petljom spirale. Ako je doneta odluka da se nastavi, crtaju se planovi za narednu fazu projekta.

Važna razlika između spiralnog modela i drugih modela softverskih procesa je eksplicitno razmatranje rizika u spiralnom modelu. Neformalno, rizik je jednostavno nešto što može da krene pogrešno. Na primer, ako je intencija da koristimo nove programske jezike, rizik je da su postojeći kompajleri nepouzdani ili da ne proizvode dovoljno efikasan objektni kod. Rezultat rizika u projektnim problemima kao što su raspored i prekoračenje troškova je vrlo važna aktivnost upravljanja projektom.

Ciklus spirale počinje sa razradom ciljeva kao što su performanse, funkcionalnost, itd. Zatim su nabrojani alternativni načini dostizanja ovih ciljeva i ograničenja zadata od strane svake alternative. Svaka alternativa je ocenjena nasuprot svakom cilju. Ovo obično rezultira identifikacijom izvora rizika projekta. Sledeći korak je proračunavanje ovih rizika preko aktivnosti kao što su detaljna analiza, simulacija, itd. Jednom kada su rizici ocenjeni, izvršen je razvoj i ovo je praćeno planiranjem aktivnosti za narednu fazu procesa.

SVOJSTVA SPIRALNOG MODELA

Spiralni model je pogodan za velike i složene projekte,ali je skup i složen za korišćenje .

Primena spiralnog modela je odgovarajuće za slučaj velikih i složenih projekata, koje prate veliki razici, jer se u spiralnom modelu posebna pažnja poklanja analizi rizika.

Dobro svojstvo spiralnog modela je što *omogućuje ikrementalnu isporuku* softvera i što u sebi *uključuje i tehnike korišćenja prototipova*, kako bi se smanjili rizici pri razvoju softvera. Ustvari, spiralni model podržava korišćenje i inkrementalnog i sekvencijalnog modela razvoja, kao i upotrebu modela sa prototipovima. Ova kombinacija pristupa, čini ovaij model vrlo moćnim.

Međutim*, složenost modela mu je istovremeno i slabost*, jer je složen za upravljanje i negovu primenu prate visoki troškovi korišćenja, pre svega zbog analiza rizika i korišćenja prototipova

u svakoj petlji. Zato, nije pogodan za primenu u malim i srednjim projektima razvoja softvera ili u slučajevima gde je agilnost procesa vrlo bitna.

PRIMER

Primena spiralnog modela razvoja softvera

Primer razvoja sistema za centralizaciju dokumenata na teritoriji jedne države primenjuje spiralni model razvoja i definiše iterativni razvoj u četiri iteracije:

- zahtevi i planiranje životnog ciklusa
- planiranje razvoja
- planiranje integracije i testiranja
- implementacija

Spiralni model je izabran jer vrši:

- redukovanje rizika; sistem se razvija izradom prototipa za najvažnije karakteristike, a onda se po testiranju prototipa, unose izmene u
  sistem; tako su rizici najmanji
- dobru kontrolu troškova; pošto su prototipovi mali, troškovi se lako procenjuju aktivno učešće korisnika

Problemi koji mogu nastati:

- ograničena primena; model nije pogodan za manje projekte, već najbolje radi u slučaju velikih i složenih projekata
- neophodno znanje o rizicima; potrebna velika veština u radu sa rizicima; uvođenje rizika uvećava troškove i to uvećanje može
  da bude veće od troškova izrade sistema
- složenost modela; striktno definisan protokol razvoja je nekada teško ispoštovati

ZADATAK

Proverimo vaše razumevanje spiralnog modela.

1. Objasnite zašto je spiralni model prilagodljiv i promenama i aktivnostima koje su tolerantne na promene.
2. U praksi, ovaj model se najčešće koristi. Objasnite zbog čega?
3. Kada biste preporučili upotrebu spiralnog modela? Objasnite svoj odgovor.
4. Navesti primer korišćenja spiralnog modela razvoja softvera. Definisati aktivnosti svake od sekcija modela.

Poglavlje 5

Rational ujedinjeni proces (RUP)

RUP MODEL

RUP model se bavi analizom rizika i podržava razvoj koji primenjuje slučajeve korišćenja. Posebnu pažnju posvećuje arhitekturi softvera i ona je u centru njegove pažnje.

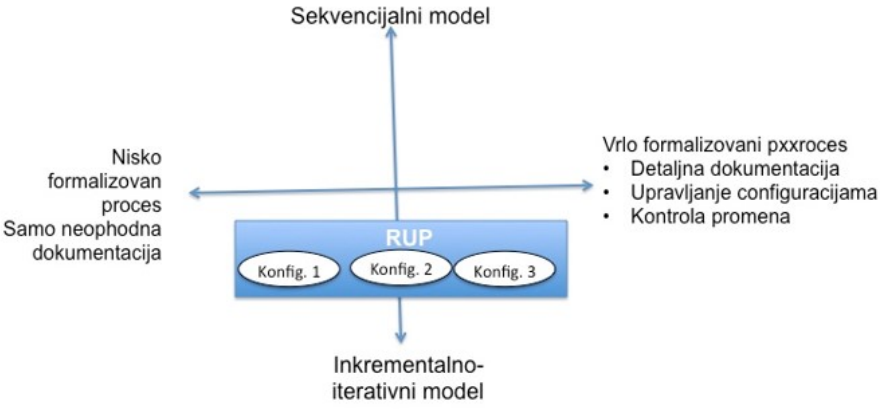
Rationa l Unified Process ili RUP je model procesa koji se bazira na inkrementalnom i iterativnom proces razvoja softvera. RUP model se bavi analizom rizika i podržava razvoj koji primenjuje slučajeve korišćenja. Posebnu pažnju posvećuje arhitekturi softvera i ona je u centru njegove pažnje.

RUP ie takođe i okvir za modeliranje procesa. Omogućuje prilagođavanje procesa potrebama korisnika (process customization) i omogućava definisanje procesa. Na taj načim, omogućava dobijanje različitih konfiguracija procesa. Te specifične konfiguracije omogućavaju:

- Podršku razvojnim timovima različite veličine (male, srednje i velike)
- Primenu vrlo formalizovanog ili malo formalizovanog procesa razvoja softvera, ttj. metoda razvoja softvera.

Da bi jasnije pozicionirali RUP u odnosu na bitne parametre koje klasifikuje različite pristupe u postavljanju proceesa razoja softvera, koristićemo dve ose sa po dve ekstremne vrednosti tih parametara (slika 1).

Na vertikalnoj osi prikazan je metod isporuke razvijenog softvera. Na vrhu je ekstremni slučaj primene metoda vodpada, a na dnu je drugi ekstrem – primena inkrmentalnog i iterativnog načina isporke, tj. razvoja softvera. Na horizontalnoj osi se prikazuje mera formalizacecije procesa razvoja softvera (novo proizvedene i korišćene dokumenstacije, upravljanja konfiguracijama i kontrole promena u softveru. Na desnoj strani se daje najveći stepen korišćenja projektne i druge dokumentacije, metoda konfigurisanja softvera i kontrole promena u softveru. Na drugom, levom kraju je dat drugi ekstremni slučaj: vrlo malo dokumentacije i velika fleksibilnost u definisanju procesa razvoja.



Slika 5.1 Konfiguracije RUP-a

FAZE SOFTVERSKOG PROCESA U RUP MODELU

RUP model ima četiri faze softverskog procesa u kome su ove faze povezane sa aktivnostima, koje su su povezane u skladu sa poslom, a ne tehnikom, za razliku od modela vodopada.

Rational ujedinjeni proces (engl. The Rational Unified Process – RUP) je primer modernog modela procesa koji primenjuje UML (engl. Unified Modelling Language) za modeliranje procesa. Kao model hibridnog procesa, on sadrži elemente sadržane u svim opštim modelima procesa, i predstavlja dobru praksu pripreme specifikacije i projektnog rešenja softvera, kao i podrške primene prototipova i inkrementalne isporuke softvera.

RUP opisuje proces iz **tri perspektive:**

1. Dinamička perspektiva – pokazuje faze model tokom vremena.
2. Statička perspektiva – prikazuje aktivnosti procesa
3. Perspektiva prakse – sugeriše dobre prakse upotrebe modela

Najčešće se primenjuje kombinovanje statičke i dinamičke perspektive u vidu jednoh jedinstvenog dijagrama.

RUP model ima **četiri faze softverskog procesa**. Za razliku od modela vodopada, u kome su ove faze povezane sa aktivnostima, kod RUP modela, ove faze su povezane u skladu sa poslom, a ne tehnikom (slika). Naredna faza počinje kada se prethodna završi, ali u okviru svake faze postije više iteracija. Te faze su:

1. **Početak** :Cilj početne faze je postavljanje poslovnog scenarija (slučaja) sistema, tj. određivanja šta sistem treba da radi. Utvrđuju se spoljni akteri (ljudi i sistemi) koji su interakciji sa sistemom i definišu se te interakcije. *Vrši se procena efekata rada sistema na poslovanje i procena rizika*. Ako su efekti mali, može da dođe i do zaustavljanja projekta.
2. **Razrađivanje** (elaboracija): Cilj ove faze je razvoj razumevanja domena problema, postavljanje arhitektonskog okvira sistema (koncepcijsko rešenje arhitekture), razvoj plana projekta, i utvrđivanje ključnih rizika projekta. Na kraju ove faze, dobija *se*

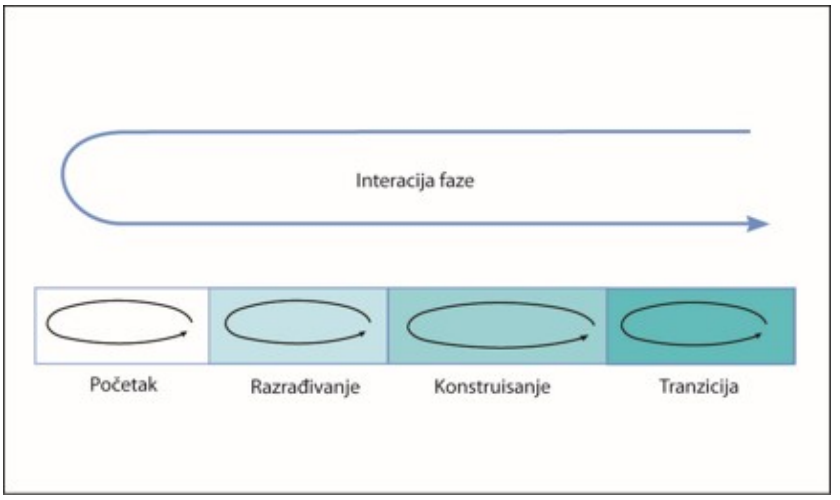
model zahteva sistema u vidu skupa UML slučajeva primene, opis arhitekture i razvijen plan projekta.

3. **Konstrukcija**: Faza konstrukcije obuhvata projektovanje sistema, programiranje i testiranje. Delovi sistema se paralelno razvijaju u ovoj fazi. Na kraju ove faze, *sistem je u radnom stanju*, zajedno sa pratećom dokumentacijom, te je *spreman za isporuku korisnicima.*
4. **Tranzicija**: Konačna faza RUP-a se bavi prenosom sistema iz razvojnog okruženja u korisničko okruženje, i stavlja ga u rad u stvarnom okruženju. Na kraju ove faze, *sistem je opremljen kompletnom softverskom dokumentacijom i ispravno radi u operativnom (radnom) okruženju*.

RADNI TOKOVI U RUP MODELU

Statički pogled na RUP se fokusira na aktivnosti razvojnog procesa. One se nazivaju i radnim tokovima

Iteracije u RUP-u se javljaju unutar svake faze, ali i na nivou celog procesa (vraćanje na prethodne faze).



Slika 5.2 Iteracije u RUP modelu

Statički pogled na RUP se fokusira na aktivnosti razvojnog procesa. One se nazivaju i radnim tokovima (eng. workflows). Postiji šest ključnih radnih tokova procesa i tri radna toka podrške (videtiTtabelu 1). Kako je RUP projektovan primenom UML, to je i opis ovih radnih tokova izvršen u vidu odgovarajućih UML modela.

Radni tok	Opis
Modelovanje poslovanja	Poslovni procesi se modeluju primenom UML slučajeve upotrebe (UML use cases)
Zahtevi	Utvrđeni su akteri koji su interakciji sa sistemom i slučajevi upotrebe su razvijeni radi modelovanja zahteva sistema.
Analiza i projektovanja	Kreira se i dokumentuje model projektovanja upotrebom modela arhitektura, modela komponenti, objektnih modela i sekvencnih modela
Implementacija	Komponente sistema su primenjene i strukturisane u podsistemima. Automatska generacija koda iz projektnog modela ubrzavaju proces.
Testiranje	Testiranje je iterativni proces koji se sprovodi zajedno sa implementacijom. Testiranje sistema se vrši posle završetka implementacije.
Instaliranje	Napravljena je konačna verzija proizvoda, podeljena je korisnicima i instalirana na mestima upotrebe.
Konfigurisanje i upravljanje promenama	Ovaj radni tok upravlja promenama sistema.
Upravljanje projektima	Ovaj tok rada upravlja razvojem sistema.
Okruženje	Ovaj tok rada omogućava da tim za razvoj softvera koristi odgovarajućie softverske alate.

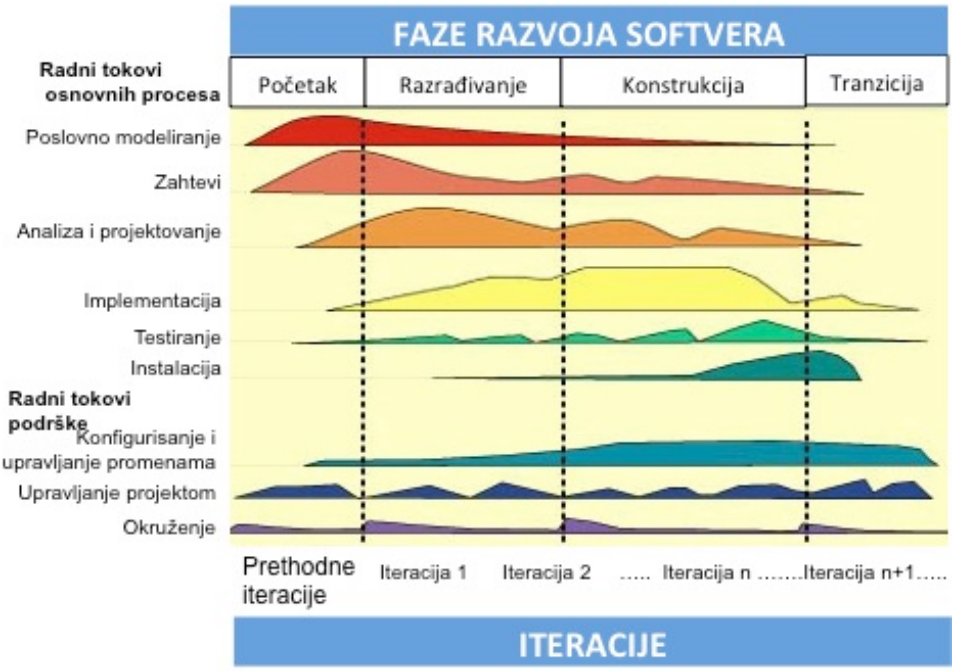
AKTIVNOSTI RADNIH TOKOVA PROCESA RAZVOJA

Aktivnosti radnih tokova procesa razvoja, a i procesa podrške, prisutne se u svim fazama razvoja softvera i izvršavaju se s arazličitim intenzitetom.

Na slici 4 su prikazani radni tokovi osnovnih procesa i radni tokovi podrške tokom četiri faze razvoja softvera. Radne tokove osnovnih procesa čini šest radnih tokova, a tri radna toka čine radne tokove podrške. Aktivnosti ovih devet radnih tokova se realizuju u svim fazama, sa različitim intenzitetom. Na primer, u početnoj fazi, najintenizivniji je rad na poslovnom modelovanju (slučajevi upotrebe, scenriji) i na definisanju zahteva. U fazi razrađivanja, naintenzivnije aktivnosti su na detaljnijoj specifikaciji zahteva i na analizi sistema i projektovanju. U fazi konstrukcije softvera, najintenzivniji je rad na implementaciji projektnog rešenja, tj. na programiranju u testiranju. U fazi tranzicije najviše se radi na završnom i korisničkom testiranju.

Aktivnosti radnih tokova podrške se odvijaju tokom celog proce sa različitim intenzitetom.

U svima fazama razvoja softvera, ikreiraju, se primenom više iteracija, različite verziije softverskih jedinica i konfiguracije celog softverskog sistema.



Slika 5.3 Radni tokovi u RUP modelu

NAJBOLJA PRAKSA U KORIŠĆENJU RADNIH TOKOVA RUP-A

Najveća inovacija RUP-a je odvajanje faza i radnih tokova, kao i prepoznavanje raspoređivanja (instalacije) softvera u radno okruženje korisnika kao deo procesa

U principu, svi radni tokovi mogu biti aktivni u svim fazama procesa razvoja. Neki su aktivniji u početnim fazama, neki u završnim fazama. Preporučuje se šest osnovnih najboljih praksi:

1. *Iterativni razvoj softvera:* Planirirani inkremente sistema na osnovu prioriteta kupca.
2. *Uređivanje zahteva*: Jasno definisati zahteve kupaca i beležite promene tih zahteva. Analizirati posledice usvajanja novih zahteva na sistem, pre nego što se prihvate.
3. *Upotrebiti arhitektura baziranu na komponentama*: Strukturisati arhitekturu sa komponentama.
4. *Vizualno modelovati softver*: Upotrebiti UML modele za predstavljanje statičkih i dinamičkih pogleda na softver.
5. *Proveriti kvalitet softvera:* Obezbediti proveru da li softver zadovoljava standarde kvaliteta organizacije.
6. *Kontrolisati promene u softveru:* Uređivati promene u softveru upotrebom sistema za upravljanje promena sistema i alate i procedure upravljanja konfiguracijom.

RUP nije pogodan za sve sisteme, kao što su na primer, ugrađeni sistemi. Najveća inovacija RUP-a je odvajanje faza i radnih tokova, kao i prepoznavanje raspoređivanja (instalacije) softvera u radno okruženje korisnika kao deo procesa. Faze su dinamičke i imaju svoje ciljeve. Radni tokovi su statički i predstavljaju tehničke aktivnosti koje nisu povezane sa

pojedinačnom fazom, već se mogu upotrebiti za vreme celog razvoja radi ostvarivanja ciljeva svake faze.

RUP je okvir koji vam omogućava da izaberete model procesa koji najviše odgovara specifičnostima softvera koji razvijate, jer možete birati stepen formalizma procesa.

PRIMER

Primena RUP-a

Aplikacija za kontrolu aktivnosti (To-do planer) za mobilne uređaje. Koristeći RUP proces, faze u razvoju su sledeće:

- **Početak** - u toku početne faze, inicijalna evaluacija se sprovodi u cilju utvrđivanja da li je projekat vredan pažnje. Poslovni plan je detaljno definisan i urađena je procena troškova razvoja i rasporeda. Postignut je dogovor o obimu projekta sa svim zainteresovanim stranama. Neke od funkcija se mogu ukloniti kako bi se smanjilo vreme i troškovi razvoja - na primer, lista naloga je izvorno podržala tri različita tipa zadataka, ali će sada podržavati samo jedan
- **Razrađivanje** - u toku razrade, vrši se detaljnija evaluacija, kreira se plan razvoja i ublažavaju ključni rizici. Razvojni tim piše 80% svih slučajeva upotrebe, stvara sistemsku arhitekturu i plan razvoja.
- **Konstrukcija** - u toku izgradnje, stvara se softverski sistem - kod je napisan i testiran. Posvećeni tim proizvođača radi na projektu. Tim takođe testira rezultujući softver. Ključni izlaz ove faze je operativni softver
- **Tranzicija** - u fazi tranzicije, softver se izdaje krajnjem korisniku. Aplikacija se šalje na store. Kada se prijava prihvati, projekat je formalno objavljen, a tim nastavlja da ga održava.

ZADATAK

Proverite vaše razumevanje RUP metoda razvoja softvera

1. Koje su prednosti i nedostaci obezbeđivanja statičkih i dinamičkih pogleda softverskog procesa u RUP modelu?

Poglavlje 6

Zaključak

ZAKLJUČAK

Softver ne nastaje trenutno, već procesom koji obuhvata niz aktivnosti, kojim se obezbeđuje ispunjenje postavljenih zahteva.

1. Softverski procesi su aktivnosti koje se realizuju pri razvoju softverskog sistema. Modeli softverskih procesa su apstraktna predstavljanja ovih procesa.
2. Opšti modeli procesa opisuju organizaciju softverskih procesa. Primeri ovih opštih modela su: model vodopada, inkrementalni razvoj i razvoj korišćenjem višestruko upotrebljivih komponenata.
3. Inženjerstvo zahteva je proces razvoja specifikacije softvera. Specifikacije povezuju potrebe kupaca i razvojni tim softvera.
4. Procesi projektovanja i implementacije vrše transformaciju zahteva u izvršni softverski sitem. Sistematski metodi projektovanja mogu se upotrebiti za deo ovih transformacija.
5. Evolucija softvera se javlja zbog promena softverskog sistema tokom svog rada, a zbog zadovoljavanja novih zahteva korisnika. Promene su stalne i softver se mora menjati d abi ostao koristan.
6. Procesi treba da sadrže i aktivnosti koje se bave promenama. To može da obuhvati fazu izrade prototipa, koja može da pomogne u izbegavanju donošenja loših odluka u vezi zahteva i projektnog rešenja. Procesi mogu biti strukturisani za iterativni razvoj i isporuku, tako da se promene mogu realizovati bez ugrožavanja celine sistema.
7. Rational ujedinjeni proces (RUP) je moderan opšti model procesa koji je organizovan u fazama (početak, razrađivanje, konstrukcija i tranzicija), i aktivnostima (zahtevi, analiza, projektovanje i dr) u ovim fazama.