UNIVERZITET METROPOLITAN BEOGRAD

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Softverski proces

Lekcija 02

PRIRUČNIK ZA STUDENTE

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Lekcija 02

SOFTVERSKI PROCES

UVOD

Cilj ove lekcije da vas uvede u pojam „softverski proces“, tj. proces razvoja softvera i skup aktivnosti koje čine taj proces.
Ključna pitanja:

1. Razumevanje koncepta softverskih proces i modela softverskih proces.
2. Koja su tri opšta modela softverskog procesa i kada se oni koriste?
3. Koje su osnovne aktivnosti inženjerstva utvrđivanja zahteva za softverom, za testiranjem i evolucijom?
4. Zašto procesi treba d a budu tako organizovani da mogu da se prilagođavaju promenama zahteva i projektnog rešenja softvera?

Poglavlje 1

Modeli softverskog procesa

TRI NAJPOZNATIJA MODELA SOFERSKOG PROCESA

Modeli predstavljaju samo apstrakcije procesa (uprošćeni prikaz) razvoja softvera.

Softverski proces je skup povezanih aktivnosti koji vodi proizvodnji softverskog proizvoda.
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 ovde dajemo vrlo uopšteni modeli softverskih procesa (tzv. „paradigme procesa“) koji se prikazuju iz perspektive arhitekture procesa. Opšti 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:

- **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 softvera, primena (implementacija), testiranje i dr. koje se jedna za drugom realizuju.
- **Model Inkrementalni** (postepenog) razvoja: On povezuje aktivnosti specifikacije, razvoja, i provere, kao niz serija verzija (inkremenata) softvera, pri čemu svaka verzija dodaje određenu funkcionalnost na prethodnu verziju.
- **Integracija i konfiguracija** Ovaj pristup se oslanja na dostupnost komponenti ili sistema za višekratnu upotrebu. Proces razvoja sistema se fokusira na konfigurisanje ovih komponenti za upotrebu u novom okruženju i njihovu integraciju u sistem.

Ne postoji univerzalni model procesa koji je ispravan za sve vrste razvoja softvera. Pravi proces zavisi od zahteva korisnika i regulatornih zahteva, okruženja u kome će se softver koristiti i vrste softvera koji se razvija. Na primer, softver koji je kritičan za bezbednost obično se razvija korišćenjem procesa vodopada jer je potrebno mnogo analiza i dokumentacije pre nego što implementacija počne. Softverski proizvodi se sada uvek razvijaju koristeći model inkrementalnog procesa. Poslovni sistemi se sve više razvijaju konfigurisanjem postojećih sistema i njihovo integracijom da bi se kreirao novi sistem sa funkcionalnošću koja je potrebna.

KOMBINOVANJE OPŠTIH MODELA

Za velike sisteme, ima smisla kombinovati neke od najboljih karakteristika svih opštih procesa

Većina praktičnih softverskih procesa zasnovana je na opštem modelu, ali često uključuje karakteristike drugih modela. Ovo posebno važi za inženjerstvo velikih sistema. Za velike sisteme, ima smisla kombinovati neke od najboljih karakteristika svih opštih procesa. Morate imati informacije o osnovnim sistemskim zahtevima da biste projektovali softversku arhitekturu koja podržava ove zahteve. Ne možete to postepeno razvijati. Podsistemi u okviru većeg sistema mogu se razviti korišćenjem različitih pristupa. Delovi sistema koji se dobro razumeju mogu se specificirati i razvijati korišćenjem procesa baziranog na vodopadu ili se mogu kupiti kao gotovi sistemi za konfiguraciju. Ostale delove sistema, koje je teško unapred navesti, uvek treba razvijati primenom inkrementalnog pristupa. U oba slučaja, softverske komponente će verovatno biti ponovo korišćene.
Učinjeni su različiti pokušaji da se razviju „univerzalni“ modeli procesa koji se oslanjaju na sve ove opšte modele. Jedan od najpoznatijih od ovih univerzalnih modela je Rational Unified Process (RUP) (Krutchen 2003), koji je razvila Rational, američka kompanija za softversko inženjerstvo. RUP je fleksibilan model koji se može primeniti na različite načine da bi se kreirali procesi koji liče na bilo koji od opštih modela procesa o kojima se ovde govori. RUP su usvojile neke velike softverske kompanije (posebno IBM), ali nije dobio široko prihvaćenost.

SOFTWARE DEVELOPMENT - GEORGIA TECH (VIDEO)

1:21 minuta

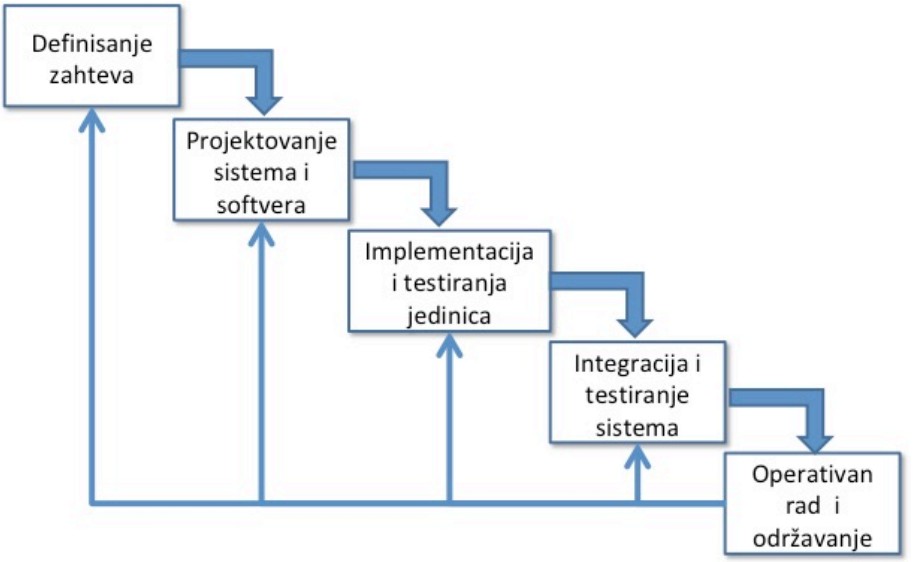
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

1.1 Model vodopada

FAZE RAZVOJA U MODELU VODOPADA

Model vodopada je primer planski vođenog softverskog procesa.

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).



Slika 1.1.1 Model vodopada [1, Fig.2.1]

Glavne faze modela vodopada su direktno povezane sa osnovnim aktivnostima razvoja softvera:

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
4. **Testiranje jedinica** utvrđuje da li svaka jedinica ostvaruje svoju planiranu funkciju.
5. **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.
6. **Operativni rad i održavanje:** Ovo je obično najduža faza životnog ciklusa softvera. Sistem je instaliran i pušten u operativni rad, tj. u upotrebu. Održavanje obuhvata ispravljanje grešaka koje nisu otkrivene u ranim fazama životnog cilusa, u toku poboljšavanja primene programskih jedinica, i poboljšanja usluga softverskog sistema u skladu sa novopostavljenim zahetvima..

NEKI PROBLEMI SA PRIMENOM MODELA VODOPADA

Kako se nove informacije pojavljuju u fazi procesa, dokumenti napravljeni u prethodnim fazama treba da budu modifikovani tako da odražavaju potrebne promene sistema

U principu, rezultat svake faze u modelu vodopada je jedan ili više dokumenata koji su odobreni („potpisani“). Sledeća faza ne bi trebalo da počne dok se prethodna faza ne završi. Za razvoj hardvera, gde su uključeni visoki troškovi proizvodnje, ovo ima smisla. Međutim, za *razvoj softvera, ove faze se preklapaju i daju informacije jedna drugoj*. Tokom projektovanja, identifikuju se problemi sa zahtevima; tokom projektovanja i kodiranja se pronalaze problemi i tako dalje. *Softverski proces, u praksi, nikada nije jednostavan linearni model, već uključuje povratne informacije od jedne faze do druge*.

Kako se nove informacije pojavljuju u fazi procesa*, dokumenti napravljeni u prethodnim fazama treba da budu modifikovani tako da odražavaju potrebne promene sistema.* Na primer, ako se otkrije da je zahtev preskup za implementaciju, dokument sa zahtevima treba promeniti da bi se uklonio taj zahtev. Međutim, ovo zahteva odobrenje korisnika i odlaže celokupni proces razvoja. Kao rezultat toga, i *klijenti i programeri mogu prerano da zamrznu specifikaciju softvera tako da se u njoj ne vrše dalje promene*. Nažalost, to znači da su problemi ostavljeni za kasnije rešavanje, zanemareni ili programirani. *Prevremeno zamrzavanje zahteva može značiti da sistem neće raditi ono što korisnik želi.* To takođe *može dovesti do loše strukturiranih sistema jer se problemi u projektovanju zaobilaze trikovima implementacije.*
Tokom završne faze životnog ciklusa (rad i održavanje) softver se stavlja u upotrebu. Otkrivaju se greške i propusti u originalnim softverskim zahtevima.
Pojavljuju se programske i projektantske greške, i identifikuje se potreba za novom funkcionalnošću. Sistem stoga mora evoluirati da bi ostao koristan. Unošenje ovih promena (održavanje softvera) može uključivati ponavljanje prethodnih faza procesa

Važna varijanta modela vodopada je razvoj formalnog sistema, gde se kreira matematički model specifikacije sistema. Ovaj model se zatim poboljšava, koristeći matematičke transformacije koje čuvaju njegovu konzistentnost, u izvršni kod. Formalni razvojni procesi, kao što je onaj zasnovan na B metodi, se uglavnom koriste u razvoju softverskih sistema koji imaju stroge zahteve za bezbednost, pouzdanost ili bezbednost. Formalni pristup pojednostavljuje izradu sigurnosnog ili sigurnosnog slučaja. Ovo pokazuje kupcima ili regulatorima da sistem zaista ispunjava svoje bezbednosne ili bezbednosne zahteve. Međutim, *zbog visokih troškova razvoja formalne specifikacije, ovaj razvojni model je retko se koristi osim za inžinjering kritičnih sistema.*

KADA KORISTITI MODEL VODOPADA?

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

U stvarnosti, softver mora da bude fleksibilan i da prilagođava promene dok se razvija. Potreba za ranim angažovanjem i preradom sistema kada se izvrše promene znači da je model vodopada prikladan samo za neke tipove sistema:

1. **Ugrađeni sistemi** gde softver mora da se poveže sa hardverskim sistemima. Zbog nefleksibilnosti hardvera, obično nije moguće odložiti odluke o funkcionalnosti softvera dok se ne implementira.
2. **Kritični sistemi** gde postoji potreba za opsežnom analizom bezbednosti i sigurnosti specifikacije i dizajna softvera. U ovim sistemima, specifikacija i projektna

dokumentacija moraju biti potpuna da bi ova analiza bila moguća. Problemi u vezi sa bezbednošću u specifikaciji i dizajnu su obično veoma skupi za ispravljanje u fazi implementacije

3. **Veliki softverski sistemi** koji su deo širih inženjerskih sistema koje je razvilo nekoliko partnerskih kompanija. Hardver u sistemima može biti razvijen korišćenjem sličnog modela, a kompanijama je lakše da koriste zajednički model za hardver i softver. Štaviše, tamo gde je uključeno nekoliko kompanija, mogu biti potrebne kompletne specifikacije kako bi se omogućio nezavisni razvoj različitih podsistema.

Model vodopada nije pravi model procesa u situacijama kada je moguća neformalna timska komunikacija i kada se softverski zahtevi brzo menjaju. Iterativni razvoj i agilne metode su bolji za ove sisteme.

WATERFALL PROCESS - GEORGIA TECH (VDEO)

1:14 minuta

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

1.2 Model inkrementalnog razvoja

INKREMENTALNI RAZVOJ SOFTVERA

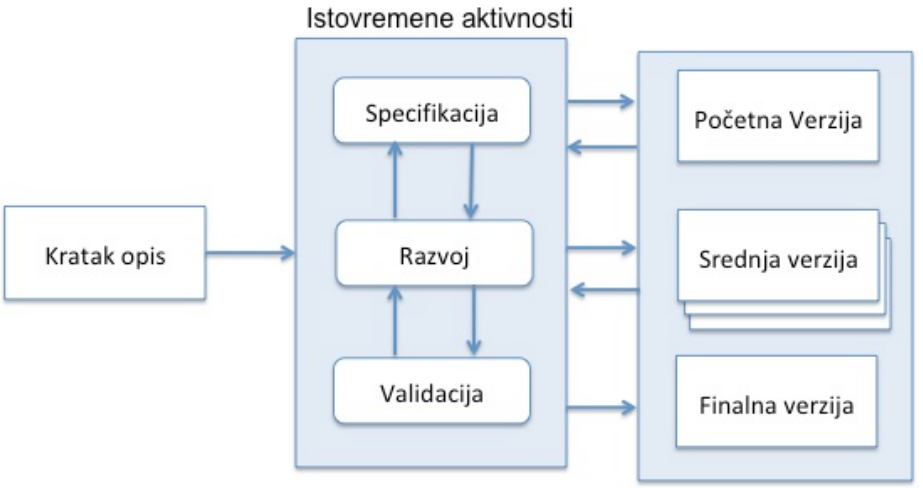
Svaki inkrement, tj. nova verzija softvera, sadrži neke od potrebnih funkcija koje traži korisnik

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.

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.

Kao i kod linearnog, tj. sekvencijalnog modela (modela vodopada), i kod inkrementalnog razvoja najpre se napravi lista zahteva koje softverski sistem treba da zadovolji (slika 1). 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.*



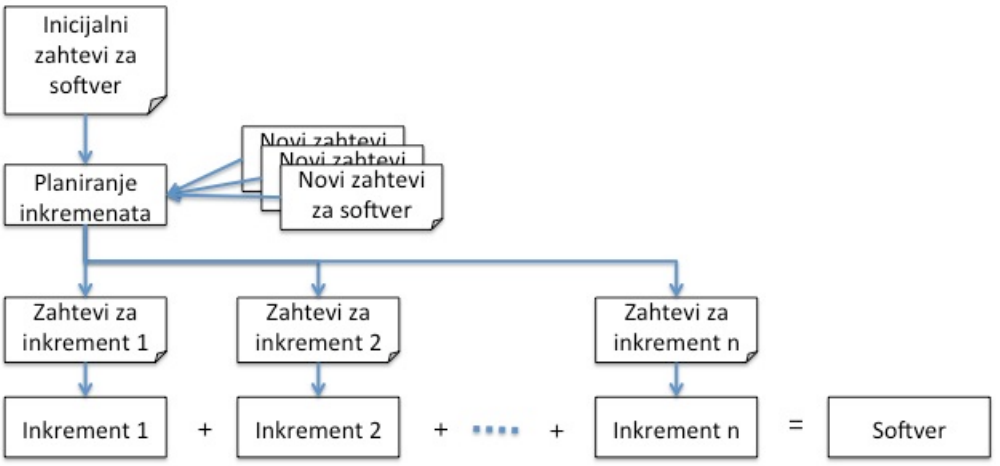
Slika 1.2.1 Model inkrementalnog razvoja softvera [1, Fig.2.2]

RAZVOJ INKREMENTA SOFTVERA

Svaki inkrement sadrži svoj skup zahteva koje treba da primeni

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



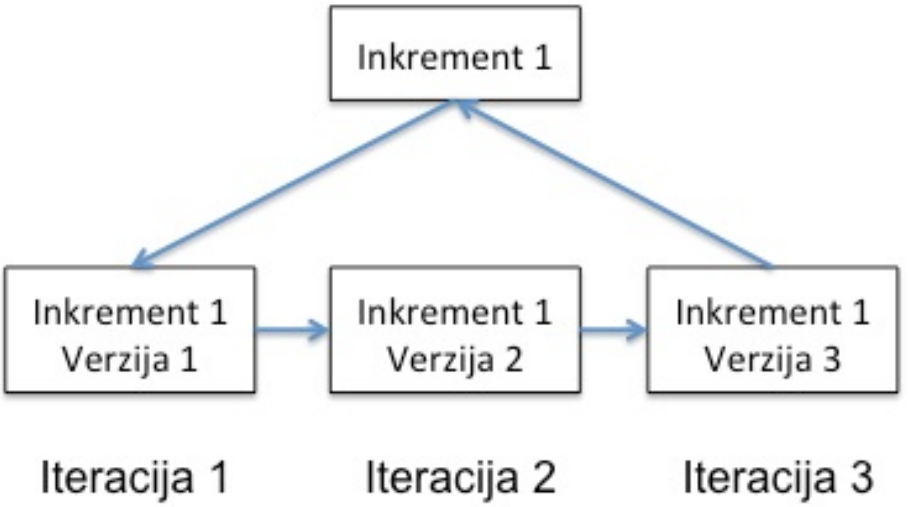
Slika 1.2.2 Proces inkrementalnog razvoja softvera [autor]

INTERACIJE U INKREMENTALNOM RAZVOJU SOFTVERA

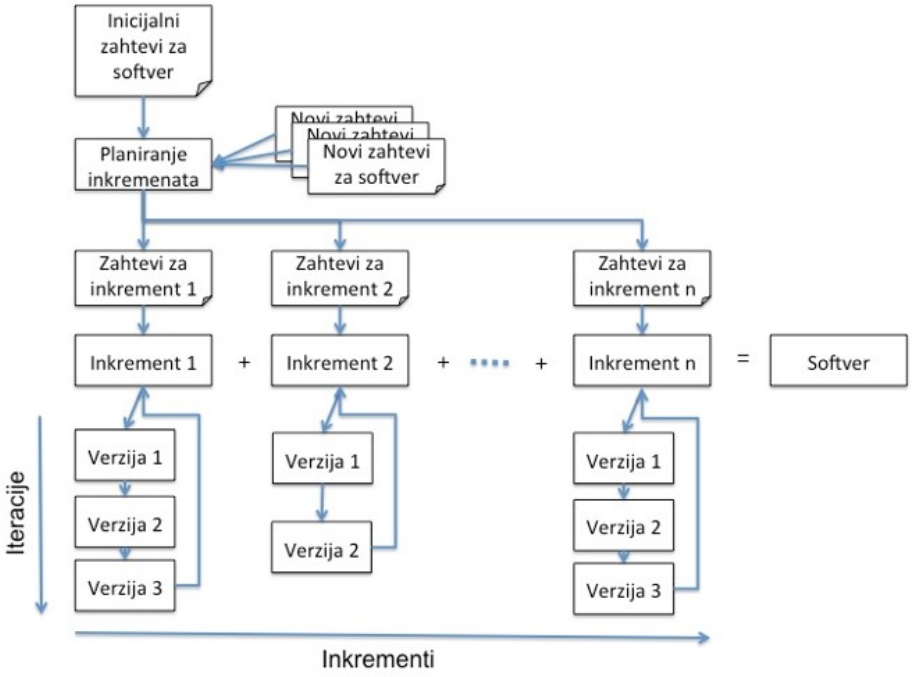
Svaki inkrement se razvija u više verzija, pre čemu svaka naredna verzija prestavlja poboljšanje prethodne verzije

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).

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



Slika 1.2.3 SE101-Slika 1.2.3 Iteracije internih verzija u okviru razvoja jednog inkrementa [autor]



Slika 1.2.4 Inkrementalni razvoj softvera sa iteracijama (verzija) unutar svakog inkrementa [autor]

PREDNOSTI INKREMENTALNOG RAZVOJA SOFTVERA

Niži troškovi, aktivo učešće korisnika, brža isporuka softvera - primena u metodima agilnog razvoja softvera

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.

1. **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.
2. **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..
3. 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.

PROBLEMI U PRIMENI INKREMENTALNOG RAZVOJA SOFTVERA

Proces nije vidljiv - teško se kontroliše. Struktura softvera se vremenom urušava.

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.

Problemi inkrementalnog razvoja postaju posebno vidljivi u slučaju velikih sistema, kada neophodna stabilnost arhitekture sistema i jasna odgovornost različitih timova koji rade na delovima sistema, a u odnosu na arhitekturu. To se mora unapred planirati primenu drugih modela razvoja softvera, umesto inkrementalnog razvoja. Iako inkrementalni razvoj ima mnoge prednosti, nije bez problema. Primarni uzrok poteškoća je činjenica da velike organizacije imaju birokratske procedure koje su evoluirale tokom vremena i može postojati neusklađenost između ovih procedura i neformalnijeg iterativnog ili agilnog procesa. Ponekad su ove procedure tu iz dobrih razloga. Na primer, mogu postojati procedure koje osiguravaju da softver ispravno ispunjava spoljne propise. Promena ovih procedura možda neće biti moguća, tako da konflikti procesa mogu biti neizbežni
Inkrementalni razvoj ne znači da morate svaki inkrement isporučiti korisniku sistema. Možete postepeno razvijati sistem i izlagati ga kupcima i drugim zainteresovanim stranama na komentar, bez da ga nužno isporučujete i primenjujete u okruženju korisnika. Inkrementalna isporuka znači da se softver koristi u stvarnim, operativnim procesima, tako da će povratne informacije korisnika verovatno biti realne. Međutim, pružanje povratnih informacija nije uvek moguće jer eksperimentisanje sa novim softverom može poremetiti normalne poslovne procese.

IS AGILE INCREMENTAL OR ITERATIVE? (VIDEO)

8:36 minuta

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

1.3 Integracija i konfiguracija komponenata

PONOVNA UPOTREBA SOFVERSKIH KOMPONENATA

Postojeće komponente se analiziraju, modifikuju i integrišu sa novim delom softvera.

U većini softverskih projekata postoji određena ponovna upotreba softvera. Ovo se često dešava neformalno kada ljudi koji rade na projektu znaju ili traže kod koji je sličan onome što je potrebno. Oni ih traže, modifikuju ih po potrebi i integrišu sa novim kodom koji su razvili.

Ova neformalna ponovna upotreba se odvija bez obzira na razvojni proces koji se koristi. Međutim, od 2000. godine, procesi razvoja softvera koji se fokusiraju na ponovnu upotrebu postojećeg softvera postali su široko korišćeni. Pristupi orijentisani na ponovnu upotrebu oslanjaju se na osnovu softverskih komponenti za višekratnu upotrebu i integrišući okvir za sastav ovih komponenti.

Tri vrste softverskih komponenti se često ponovo koriste:

1. **Samostalni aplikacioni sistemi** koji su konfigurisani za upotrebu u određenom okruženju. Ovi sistemi su sistemi opšte namene koji imaju mnogo funkcija, ali moraju biti prilagođeni za upotrebu u određenoj primeni.
2. **Kolekcije objekata** koji su razvijeni kao komponenta ili kao paket koji se integrišu sa komponentnim okvirom kao što je Java Spring framevork
3. **Veb servisi** koji su razvijeni prema standardima usluga i koji su dostupni za daljinsko pozivanje preko Interneta.

Softversko inženjerstvo orijentisano na ponovnu upotrebu komponenti i, zasnovan na njihovoj konfiguraciji i integraciji, **ima očiglednu prednost smanjenja količine softvera koji treba da se razvije i na taj način smanjenja troškova i rizika**. Obično dovodi i do brže isporuke softvera.

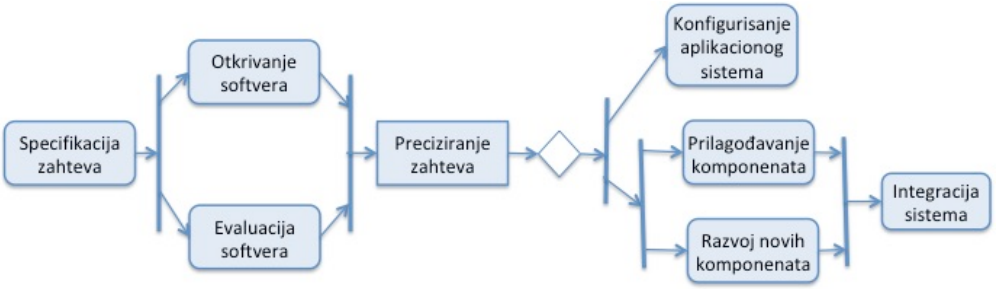
Međutim, *kompromisi u pogledu zahteva su neizbežni, a to može dovesti do sistema koji ne zadovoljava stvarne potrebe korisnika*. Štaviše, određena kontrola nad evolucijom sistema je izgubljena jer nove verzije komponenti za višekratnu upotrebu nisu pod kontrolom organizacije koja ih koristi

INTEGRACIJA I KONFIGURACIJA POSTOJEĆIH KOMPONENATA

Faze procesa razvoja softvera zasnovanog na ponovnoj upotrebi komponenata

Slika 1 prikazuje opšti model procesa za razvoj zasnovan na ponovnoj upotrebi, zasnovan na integraciji i konfiguraciji. Faze u ovom procesu su:

1. **Specifikacija zahteva** Predloženi su početni zahtevi za sistem. One ne moraju biti detaljno razrađene, ali treba da sadrže kratke opise osnovnih zahteva i poželjnih karakteristika sistema.
2. **Otkrivanje i evaluacija softvera** S obzirom na pregled softverskih zahteva, vrši se pretraga komponenti i sistema koji obezbeđuju potrebnu funkcionalnost. Komponente i sistemi kandidata se procenjuju da bi se videlo da li ispunjavaju osnovne zahteve i da li su generalno pogodni za upotreba u sistemu.
3. **Preciznost zahteva** Tokom ove faze, zahtevi se preciziraju korišćenjem informacija o otkrivenim komponentama i aplikacijama za višekratnu upotrebu. Zahtevi su modifikovani da odražavaju dostupne komponente, a specifikacija sistema je ponovo definisana. Tamo gde su modifikacije nemoguće, aktivnost analize komponenti se može ponovo uneti radi traženja alternativnih rešenja.
4. **Konfiguracija aplikacionog sistema** Ako je dostupan aplikativni sistem koji ispunjava zahteve, on se tada može konfigurisati za korišćenje za kreiranje novog sistema.
5. **Prilagođavanje i integracija komponenti** Ako ne postoji gotov sistem, pojedinačne komponente za višekratnu upotrebu mogu se modifikovati i razviti nove komponente. Oni se zatim integrišu da bi se kreirao sistem.



Slika 1.3.1 Opšti model procesa za razvoj softvera zasnovan na ponovnoj upotrebi komponenti, njihovoj integraciji i konfiguraciji [1, Fig.2.3]

Poglavlje 2

Aktivnosti softverskog procesa

OSNOVNE AKTIVNOSTI SOFTVERSKOG PROCESA

Softverski proces je skup povezanih aktivnosti koji vodi proizvodnji softverskog proizvoda

Softverski proces je *skup povezanih aktivnosti koji vodi proizvodnji softverskog proizvoda.* Ove aktivnosti mogu da dov.edu 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. 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.

Najčešće se softverski procesi sadrže sledeće četiri osnovne aktivnosti koje čine osnovu softverskog inženjerstva (slika 1):

1. **Specifikacija softvera**: Definiše funkcionalnost softvera i ograničenja na rad softvera.
2. **Projektovanje 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.

Slika 2.1.1 Osnovne aktivnosti (faze) softverskog porcesa [autor]

USLOVI KOJI ODREĐUJU REDOSLED AKTIVNOSTI

Softverski procesi zavise od ljudi koji rade u organizacijama i odražavaju specifične karakteristike tih organizacija. Zato se softverski procesi razliku od organizacije do organizacije.

Aktivnosti softverskog procesa su same po sebi složene aktivnosti i **uključuju pod-aktivnosti** kao što su validacija zahteva, arhitektonsko projektovanje i testiranje jedinica. Procesi takođe uključuju druge aktivnosti, kao što su upravljanje konfiguracijom softvera i planiranje projekta koji podržavaju proizvodne aktivnosti.
Kada opisujemo i raspravljamo o procesima, obično govorimo o aktivnostima u tim procesima, kao što su specificiranje modela podataka i dizajniranje korisničkog interfejsa, kao i planiranje redosleda ovih aktivnosti. Svi ovo je povezano a onim što rade inženjeri softvera da bi razvili softver. Međutim, kada se opisuju procesi, takođe je važno opisati ko je uključen, šta se proizvodi i uslove koji utiču na redosled aktivnosti:

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

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.

TRADITIONAL SOFTWARE PHASES - GEORGIA TECH (VIDEO)

0:58 minuta

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

2.1 Specifikacija softvera – inženjerstvo zahteva

SPECIFIKACIJA SOFTVERA

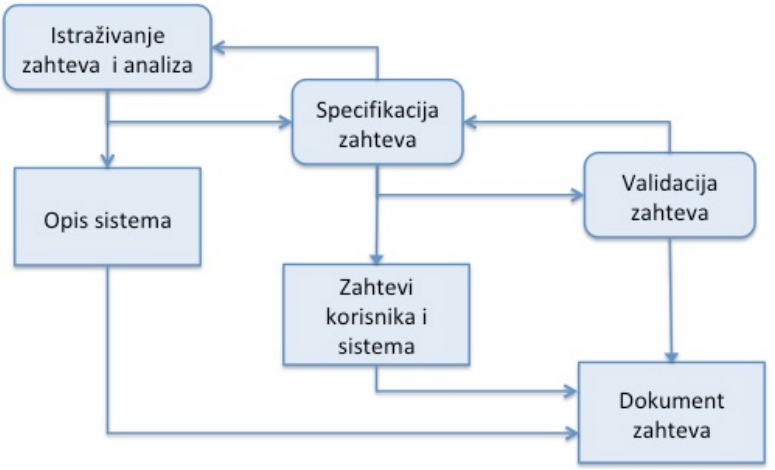
Specifikacija zahteva je proces razumevanja potreba i definisanja usluga softvera

Specifikacija softvera ili inženjering zahteva je proces razumevanja i definisanja usluga koje se zahtevaju od sistema i identifikacije ograničenja za rad i razvoj sistema. Inženjerstvo zahteva je posebno kritična faza softverskog procesa, jer greške napravljene u ovoj fazi neizbežno dovode do kasnijih problema u dizajnu i implementaciji sistema.

Pre nego što počne proces inženjerstva zahteva, kompanija može da sprovede studiju izvodljivosti ili marketing kako bi procenila da li postoji potreba ili tržište za softverom i da li je tehnički i finansijski realno da se razvije potreban softver. Studije izvodljivosti su *kratkoročne,*

relativno jeftine studije koje donose odluku o tome da li da se nastavi sa detaljnijom analizom ili ne. Proces inženjerstva zahteva (slika 1) ima za cilj da proizvede dokument sa usaglašenim zahtevima koji specificira sistem koji zadovoljava zahteve zainteresovanih strana. Zahtevi su obično predstavljeni na dva nivoa detalja. Krajnjim korisnicima i kupcima je potrebna izjava na visokom nivou o zahtevima; programerima sistema potrebna je detaljnija specifikacija sistema. Postoje tri glavne aktivnosti ili faze procesa inženjerstva zahteva (slika 1):

1. Izdvajanje i analiza zahteva: Ovo je proces izvođenja sistemskih zahteva kroz posmatranje postojećih sistema, diskusije sa potencijalnim korisnicima i naručiocima, analizu zadataka, itd. Ovo može uključivati razvoj jednog ili više modela i prototipova sistema. Ovo vam pomaže da razumete sistem koji treba navesti.



Slika 2.2.1 Proces inženjerstva zahteva [1, Fig.]

2. Specifikacija zahteva: Specifikacija zahteva je aktivnost prevođenja informacija prikupljenih tokom analize zahteva u dokument koji definiše skup zahteva. U ovaj dokument mogu biti uključene dve vrste zahteva. Zahtevi korisnika su apstraktne izjave sistemskih zahteva za kupca i krajnjeg korisnika sistema; Sistemski zahtevi su detaljniji opis funkcionalnosti koje treba obezbediti.

3. Validacija zahteva: Ova aktivnost proverava zahteve za realističnost, doslednost i potpunost. Tokom ovog procesa, greške u dokumentu sa zahtevima se neizbežno otkrivaju. Zatim se mora modifikovati da bi se ispravili ovi problemi.

NOVI ZAHTEVI

Analiza zahteva se nastavlja tokom definisanja i specifikacije, a novi zahtevi se pojavljuju tokom procesa inženjerstva zahteva.

Analiza zahteva se nastavlja tokom definisanja i specifikacije, a novi zahtevi izlaze na videlo tokom procesa. Stoga se aktivnosti analize, definicije i specifikacije prepliću. U agilnim metodama, specifikacija zahteva nije posebna aktivnost, već se posmatra kao deo razvoja sistema. Zahtevi su neformalno specificirani za svaki inkrement sistema neposredno pre nego što se taj prirast razvije. Zahtevi se određuju prema prioritetima korisnika. Iskazivanje zahteva dolazi od korisnika koji su deo razvojnog tima ili blisko sarađuju sa njim.

REQUIREMENTS ENGINEERING - GEORGIA TECH (VIDEO)

2:15 minuta

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

2.2 Projektovanje i implementacija softvera

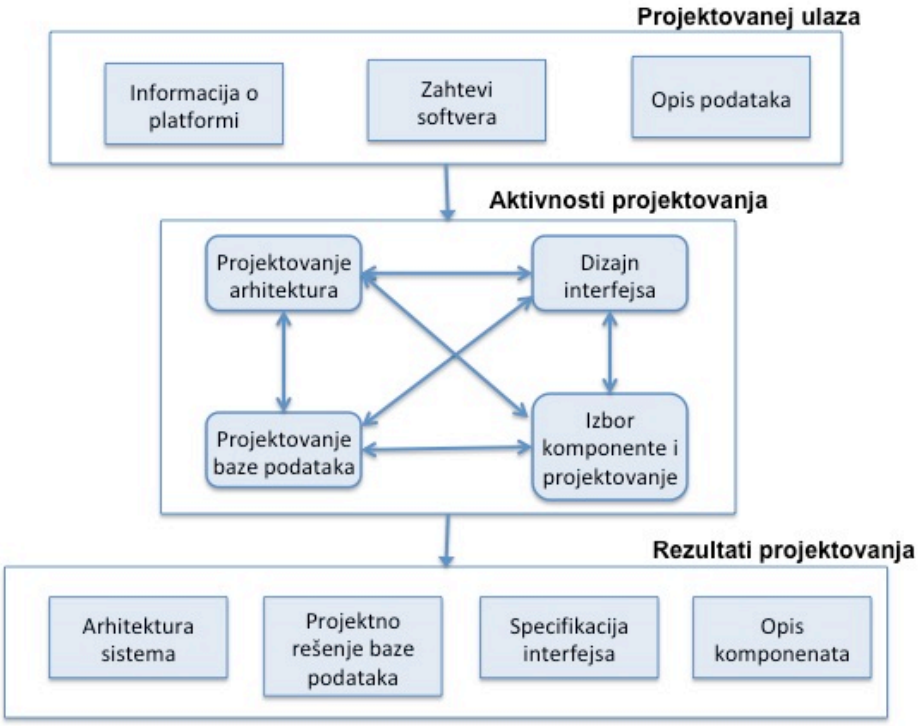
OPŠTI MODEL PROCESA PROJEKTOVANJA SOFTVERA

Model procesa projektovanja prikazuje ulaze u proces projektovanja, procesne aktivnosti i izlazne rezultate procesa.

Faza implementacije razvoja softvera je proces razvoja izvršnog sistema za isporuku kupcu. Ponekad ovo uključuje odvojene aktivnosti projektovanja softvera i programiranja. Međutim, ako se koristi agilan pristup razvoju, projektovanje i implementacija se prepliću, bez formalne projektne dokumenata koji se proizvode tokom procesa. Naravno, softver je i dalje projektovan, ali se projektno rešenje neformalno beleži na tablama i programerskim sveskama.

Projektovanje softvera je opis strukture softvera koji će se implementirati, modela podataka i struktura koje koristi sistem, interfejsa između komponenti sistema i, ponekad, korišćenih algoritama. Projektanti ne dolaze do gotovog projektnog rešenja odmah, već ga razvijaju u fazama. Oni dodaju detalje dok razvijaju svoje projektno rešenje , uz stalno vraćanje unazad kako bi modifikovali ranije dizajne.

Slika 1 je apstraktni model procesa projektovanja koji prikazuje ulaze u proces projektovanja, procesne aktivnosti i izlazne rezultate procesa. Aktivnosti procesa projektovanja su i međusobno zavisne. Nove informacije o projektnom rešenju se stalno generišu, a to utiče na prethodne odluke o projektnom rešenju. Prerada projektnog rešenja je stoga neizbežna



Slika 2.3.1 Opšti model procesa projektovanja softvera [1, Fig. 2.5]

AKTIVNOSTI U PROCESU PROJEKTOVANJA

Projektovanje arhitekture, baze podataka, interfejsa i komponenti.

Aktivnosti u procesu projektovanja variraju u zavisnosti od tipa sistema koji se razvija. Na primer, sistemi u realnom vremenu zahtevaju dodatnu fazu projektovanja vremena, ali možda ne uključuju bazu podataka, tako da ne uključuju projektovanje baze podataka. Slika 4 prikazuje četiri aktivnosti koje mogu biti deo procesa projektovanja informacionih sistema:

1. **Projektovanje arhitekture**, gde identifikujete celokupnu strukturu sistema, glavne komponente (ponekad se nazivaju podsistemima ili modulima), njihove odnose i način na koji su raspoređeni.
2. **Projektovanje baze podataka,** gde projektujete strukture podataka sistema i kako one treba da budu predstavljene u bazi podataka. Opet, posao ovde zavisi od toga da li će se ponovo koristiti postojeća baza podataka ili će se kreirati nova.
3. **Projektovanje interfejsa,** gde definišete interfejse između komponenti sistema. Interfejsi definišu poruke koje komponente mogu da primaju ili šalju. Te poruke mogu da sadrže podatke koje oni razmenjuju, ali i operacije nad podacima koje one prenose s jedne na drugu komponentu. Ova specifikacija interfejsa mora biti nedvosmislena. Sa preciznim interfejsom, komponentu mogu koristiti druge komponente, a da one ne moraju da znaju kako je implementirana. Kada su specifikacije interfejsa dogovorene, komponente se mogu zasebno projektovati i razvijati.
4. **Izbor i projektovanje komponenti**, gde tražite komponente za višekratnu upotrebu i, ako nisu dostupne odgovarajuće komponente, projektujete nove softverske komponente. Projektovanje u ovoj fazi može biti jednostavan opis komponente sa detaljima implementacije prepuštenim programeru. Alternativno, to

može biti lista promena koje treba izvršiti na komponenti za višekratnu upotrebu ili detaljan model projektnog rešenja izraženo u UML-u ("standardizovan grafički jezik" koji primenom grafičkih simbola definiše sve elemente projektnog rešenje). Model projektnog rešenja se tada može koristiti za automatsko generisanje implementacije, tj. programskog koda (u Javi, C++,C# ili Python-u, na primer.

Ove aktivnosti dovode do ishoda projektovanja, koji su takođe prikazani na slici 1. *Za kritične sisteme, rezultati procesa projektovanja su detaljni projektni dokumenti koji postavljaju precizne i tačne opise sistema.* Ako se koristi pristup vođen modelom, izlazni rezultati projektovanja su UML dijagrami koji prikazuju projektno rešenje. Ako se koriste agilne metode razvoja, rezultati procesa projektovanja ne mogu biti zasebni dokumenti specifikacije, već mogu biti predstavljeni u kodu programa.

IMPLEMENTACIJA SOFTVERA

Projektovanje i razvoj programa se često prepliću

Razvoj programa za implementaciju sistema prirodno sledi iz projektovanja sistema. Iako se neke klase programa (koje opisuju tipove objekata u programu, tj. podatke i operacije koje se mogu izvršiti nad podacima), kao što su sistemi od ključne važnosti za bezbednost, obično detaljno projektuju pre nego što započne bilo kakva implementacija, češće je da se projektovanje i razvoj programa prepliću. Alati za razvoj softvera mogu se koristiti za generisanje skeleta, tj. strukture programa iz projektnog rešenja. Ovo uključuje kod za definisanje i implementaciju interfejsa i, u mnogim slučajevima, programer treba samo da doda detalje o radu svake komponente programa.

Programiranje je individualna aktivnost i ne postoji opšti proces koji se obično prati. Neki programeri počinju sa komponentama koje razumeju, razvijaju ih, a zatim prelaze na manje razumljive komponente. Drugi imaju suprotan pristup, ostavljajući poznate komponente za kraj jer znaju kako da ih razviju. Neki programeri vole da definišu podatke rano u procesu, a zatim ih koriste za pokretanje razvoja programa; drugi ostavljaju podatke neodređenim što je duže moguće.
Obično programeri sprovode neka testiranja koda koji su razvili. Ovo često otkriva defekte (greške) programa (bagove) koje se moraju ukloniti iz programa. Pronalaženje i otklanjanje grešaka u programu naziva se otklanjanje grešaka. Testiranje i otklanjanje grešaka su različiti procesi. Testiranjem se utvrđuje postojanje nedostataka. Otklanjanje grešaka se bavi lociranjem i ispravljanjem ovih nedostataka. Kada otklanjate greške, morate da generišete hipoteze o vidljivom ponašanju programa, a zatim da testirate ove hipoteze u nadi da ćete pronaći grešku koja je izazvala anomaliju izlaza. Testiranje hipoteza može uključivati ručno praćenje programskog koda. Možda će biti potrebni novi testni slučajevi da bi se problem lokalizovao. Interaktivni alati za otklanjanje grešaka, koji pokazuju srednje vrednosti programskih promenljivih i trag izvršenih naredbi, obično se koriste kao podrška procesu otklanjanja grešaka.

DESIGN - GEORGIA TECH (VIDEO)

1:19 minuta

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

IMPLEMENTATION - GEORGIA TECH (VIDEO)

1:20 minuta

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

2.3 Validacija softvera

TRI FAZE TESTIRANJA SOFTVERA

Testiranje programa je glavna tehnika validacije (potvrđivanja)

Validacija softvera ili, uopštenije, verifikacija i validacija (V & V) *ima za cilj da pokaže da je sistem usklađen sa svojom specifikacijom (verifikacija, tj. provera) i da ispunjava očekivanja korisnika sistema (validacija, tj. potvrđivanje).* Testiranje programa, gde se sistem izvršava korišćenjem simuliranih testnih podataka, je glavna tehnika validacije. Validacija takođe može da uključuje procese provere, kao što su inspekcije i pregledi, u svakoj fazi softverskog procesa od definisanja zahteva korisnika do razvoja programa. Međutim, većina V&V vremena i truda se troši na testiranje programa.

Osim za male programe, sisteme ne treba testirati kao jednu, monolitnu jedinicu. Na slici 1 prikazan je proces testiranja u tri faze u kome se komponente sistema pojedinačno testiraju, a zatim se testira integrisani sistem. Za prilagođeni softver, testiranje korisnika uključuje testiranje sistema sa stvarnim podacima dobijenih od klijenta. Za proizvode koji se prodaju kao aplikacije, testiranje korisnika se ponekad naziva beta testiranje gde odabrani korisnici isprobavaju i komentarišu softver.

Slika 2.4.1 Tri faze testiranja softvera [1, Fig. 2.6]

Faze u procesu testiranja su:
1. **Testiranje komponenti:** Komponente koje čine sistem testiraju ljudi koji razvijaju sistem. Svaka komponenta se testira nezavisno, bez drugih komponenti sistema. Komponente mogu biti jednostavni entiteti kao što su funkcije ili klase objekata ili mogu biti koherentne grupe ovih entiteta. Obično se koriste alati za automatizaciju testiranja, kao što je JUnit za Javu, koji mogu ponovo da pokreću testove kada se kreiraju nove verzije komponente.
2. **Testiranje sistema**: Komponente sistema su integrisane da bi se stvorio kompletan

sistem. Ovaj proces se bavi pronalaženjem grešaka koje su rezultat nepredviđenih interakcija između komponenti i problema sa interfejsom komponenti. Takođe se bavi pokazivanjem da sistem ispunjava svoje funkcionalne i nefunkcionalne zahteve i testiranjem svojstava sistema koji se pojavljuju. Za velike sisteme, ovo može biti višestepeni proces gde su komponente integrisane da formiraju podsisteme koji se pojedinačno testiraju pre nego što se ovi podsistemi integrišu u konačni sistem.

3. **Testiranje korisnika:** Ovo je poslednja faza u procesu testiranja pre nego što se sistem prihvati za operativnu upotrebu. Sistem testira korisnik sistema (ili potencijalni kupac) sa svojim podacima, a ne simuliranim testnim podacima. Za softver koji napravljen po meri, tj. u skladu sa potrebama kupca, testiranje korisnika može otkriti greške i propuste u sistemu definicije zahteva, jer stvarni podaci primenjuju razvijen sistem na različite načine u odnosu na podataka korišćene prilikom testiranja od strane razvojnog tima.

TESTIRANJA SOFTVERA

Defekti komponenti bi trebalo da se se otkriju rano u procesu testiranja

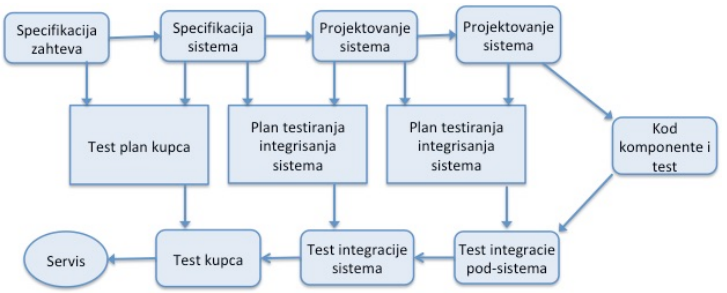
Testiranje korisnika takođe može otkriti probleme sa zahtevima kada sistemske mogućnosti ne zadovoljavaju potrebe korisnika ili su performanse sistema neprihvatljive. Za softverske proizvode (razvijene za tržište, a ne za posebnog naručioca), testiranje kupaca pokazuje koliko dobro softverski proizvod zadovoljava potrebe korisnika.

U idealnom slučaju, defekti komponenti bi trebalo da se se otkriju rano u procesu testiranja, a problemi sa interfejsom se nalaze kada se izvrši integracija sistema. Međutim, kada se otkriju greške u sistemu, u program se moraju da otklone greške, a to može zahtevati ponavljanje drugih faza u procesu testiranja. Greške u programskim komponentama, recimo, mogu izaći na videlo tokom testiranja sistema. Proces je stoga iterativan sa informacijama koje se vraćaju iz kasnijih faza u ranije delove procesa.

Obično je testiranje komponenti deo normalnog procesa razvoja. Programeri prave sopstvene testne podatke i postepeno testiraju kod kako ga razvijaju. Programer poznaje komponentu i stoga je najbolja osoba za generisanje test slučajeva.

Ako se koristi inkrementalni pristup razvoju, *svaki inkrement treba testirati kako se razvija, pri čemu se ovi testovi zasnivaju na zahtevima za taj inkrement*. U razvoju softvera vođen testom, koji je normalan deo agilnih procesa, *testovi se razvijaju zajedno sa zahtevima pre nego što razvoj počne*. Ovo pomaže testerima i programerima da razumeju zahteve i osigurava da nema kašnjenja prilikom kreiranja test slučajeva.

Kada se koristi softverski proces vođen planom (npr. za razvoj kritičnih sistema), *testiranje je vođeno skupom planova testiranja*. Nezavisni tim testera radi na osnovu ovih planova testiranja, koji su razvijeni na osnovu specifikacije i projekta sistema Slika 2 ilustruje vezu između aktivnosti testiranja i razvoja. Ovo se ponekad naziva V-model razvoja koji prikazuje aktivnosti validacije softvera koje odgovaraju svakoj fazi modela procesa vodopada.



Slika 2.4.2 7 Faze testiranja u softverskom procesu vođenog planom [1, Fig. 2.7]

Kada se sistem prodaje kao softverski proizvod, često se koristi proces testiranja koji se naziva **beta testiranje.** Beta testiranje uključuje isporuku sistema određenom broju potencijalnih kupaca koji pristaju da koriste taj sistem. Oni prijavljuju probleme programerima sistema. Ovo izlaže proizvod stvarnoj upotrebi i otkriva greške koje možda nisu očekivane od strane programera proizvoda. Nakon ovih povratnih informacija, softverski proizvod može biti modifikovan i pušten za dalje beta testiranje ili opštu prodaju.

VERIFICATION & VALIDATION - GEORGIA TECH (VIDEO)

1:35

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

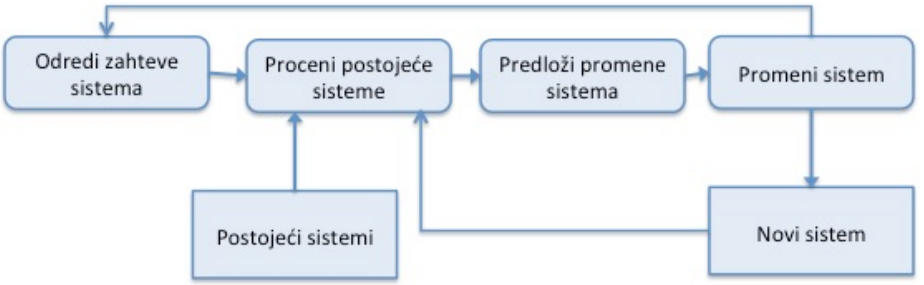
2.4 Evolucija softvera

EVOLUTIVAN RAZVOJ SOFTVERA - KONTINUIRAN RAZVOJ

Realnije je razmišljati o softverskom inženjerstvu kao o evolucionom procesu gde se softver neprestano menja tokom svog životnog veka.

Fleksibilnost softvera je jedan od glavnih razloga zašto se sve više softvera ugrađuje u velike, složene sisteme. Kada se donese odluka o proizvodnji hardvera, veoma je skupo izvršiti izmene u projektnom rešenju hardvera. Međutim, promene u softveru se mogu izvršiti u bilo kom trenutku tokom ili nakon razvoja sistema. Čak su i velike promene i dalje mnogo jeftinije od odgovarajućih promena sistemskog hardvera.
Istorijski gledano, uvek je postojala podela između procesa razvoja softvera i procesa evolucije softvera (održavanje softvera). Ljudi misle o razvoju softvera kao kreativnoj aktivnosti u kojoj se softverski sistem razvija od početnog koncepta do radnog sistema. Međutim, oni ponekad smatraju da je održavanje softvera dosadno i nezanimljivo. Oni misle da je održavanje softvera manje zanimljivo i izazovno od originalnog razvoja softvera.
Ova razlika između razvoja i održavanja je sve više irelevantna. Veoma mali broj softverskih sistema su potpuno novi sistemi, i mnogo je logičnije posmatrati razvoj i održavanje kao

kontinuitet. Umesto dva odvojena procesa, realnije je razmišljati o softverskom inženjeringu kao o evolucionom procesu (slika 1) gde se softver neprestano menja tokom svog životnog veka kao odgovor na promenljive zahteve i potrebe korisnika.



Slika 2.5.1 Evolucija softverskog sistema

MAINTENANCE - GEORGIA TECH (VIDEO)

2:09 minuta

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

Poglavlje 3

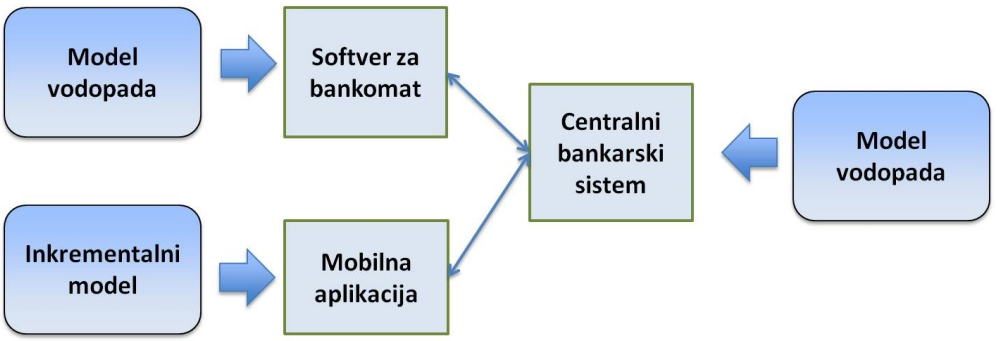
Vežba - Pokazni primeri

PRIMER 1: IZBOR MODELA SOFTVERSKOG PROCESA - BANKARSKI SISTEM

Hibridni modela softverskog procesa je pogodan za bankarski sistem

Tradicionalno se softver u finansijskom sektoru razvija primenom modela vodopada. Dobro definisane faze razvoja i predvidljivost privlače kompanije koje se bave finansijama da izaberu baš ovakav model. Povećana interakcija krajnjih korisnika sa finansijskim sistemima dovodi do češćih promena u zahtevima, kao i do potrebe za sve bržom isporukom softvera zbog sve veće konkurencije na tržištu.

U ovakvoj situaciji moguće rešenje je kombinovanje tradicionalnog modela vodopada sa inkrementalnim modelima. Na taj način se dobija hibridni model softverskog procesa. Delovi sistema koji su najpodložniji promenama i koji najviše interaguju sa krajnjim korisnikom se razvijaju inkrementalno, a ostatak sistema modelom vodopada.
U našem primeru jednostavnog bankarskog sistema, softver za bankomat bi se razvijao modelom vodopada, a mobilna aplikacija inkrementalno. Pored toga, model vodopada je najpogodniji za centralni bankarski sistem. Softver za bankomat ima jasno definisane zahteve koji se retko menjaju, a pored toga je potrebno da softver bude veoma bezbedan i pouzdan. Mobilna aplikacija mora dosta češće da se prilagođava novim zahtevima korisnika, kao i novim trendovima na tržištu



Slika 3.1 Primena hibridnog modela softvera za slu;aj bankarskog sistema

PRIMER 2: IZBOR MODELA SOFTVERSKOG PROCESA ZA SISTEM ZA ONLINE IZNAJMLJIVANJE HOTELSKOG SMEŠTAJA

Sistem za online iznajmljivanje hotelskog smeštaja

Sistem omogućava rezervaciju hotelskog smeštaja (soba, apartmana, noćenja sa doručkom, polupansion, punpansion i sl.), iznajmljivanje vikendica, vila, bungalova, odnosno rezervaciju kompletne ponude fizičkih lica koja postavljaju svoju ponudu.

Sistem treba da poseduje:

- mogućnost rezervacije smeštaja uz mogućnost online plaćanja,
- formu koja izračunava dostupnost smeštaja i cene ponuđenog smeštaja u skladu sa zahtevima klijenta,
- kontrolu ponuđenog smeštaja preko admin panela posebno za fizička lica koja postavljaju neku ponudu, a posebno za administratora celokupnog sistema

Model koji će se koristiti prilikom izrade ovog softverskog sistema je model vodopada. Zahtevi su jasno definisani što omogućava da koraci idu sekvencijalno.
Prvi korak je analiziranje kao i definisanje svrhe ovog softvera, određivanje stepena izvodljivosti realizacije definisanih zahteva, kao i jasno definisanje samih zahteva. Nakon toga sledi projektovanje sistema i softvera u kome se zahtevi raspoređuju svim komponentama sistema i uspostavlja se arhitektura sistema. Zatim sledi implementacija softvera odnosno njegovih programskih jedinica. Paralelno sa implementacijom razvijaju se i testovi 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 završno testiranje. Ukoliko je testiranje uspešno 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

Poglavlje 4

Vežba - zadaci

ZADACI ZA INDIVIDUALNI RAD STUDENATA

Zadaci za samostalno rešavanje zadataka na vežbi i kod kuće

Posle predavanja a pre časova vežbanja (održavaju se dva dana kasnije), poželjno je da pokušate da rešite neke od ovih zadatka radom kod kuće i da rezultate pošaljete mejlom, preko Zimbre, saradniku koji drži vežbe, najkasnije jedan sat pre održavanja vežbi. Na vežbama ćete raditi jedan deo zadataka na individualnoj bazi (svaki student sam radi svoje zadatke) uz pomoć saradnika, ako bude potrebno. Ako neki zadatak nije urađen pre ili za vreme vežbi, preporučuje se studentu da ih uradi posle vežbi, kod kuće.

1. Predložite najprikladniji generički model procesa softvera koji bi se mogao koristiti kao osnova za upravljanje razvojem sledećih sistema. Objasnite svoj odgovor prema vrsti sistema koji se razvija:
a) Sistem za kontrolu kočenja protiv blokiranja u automobilu
b) Sistem virtuelne realnosti koji podržava održavanje softvera
c) Univerzitetski računovodstveni sistem koji zamenjuje postojeći sistem
d) Interaktivni sistem planiranja putovanja koji pomaže korisnicima da planiraju putovanja sa najmanjim uticajem na životnu sredinu
2. Inkrementalni razvoj softvera mogao bi se veoma efikasno koristiti za kupce koji nemaju jasnu predstavu o sistemima potrebnim za njihovo poslovanje. Diskutujte ovo rešenje.
3. Razmotrite model procesa integracije i konfiguracije prikazan na slici 1 u akciji: "Integracija i konfiguracija postojećih komponenata". Objasnite zašto je neophodno ponoviti aktivnosti inženjering zahteva u procesu.
4. Predložite zašto je važno napraviti razliku između razvoja korisničkih zahteva i razvoja sistemskih zahteva u zahtevima inženjerskog procesa. .
5. Koristeći primer, objasnite zašto su projektantske aktivnosti arhitektonskog projektovanja, projektovanja baze podataka, projektovanje interfejsa i projektovanja komponenti su međusobno zavisni.
6. Objasnite zašto testiranje softvera uvek treba da se inkrementalno sprovodi, tj. da bude postepena aktivnost. Da li su programeri najbolji ljudi za testiranje programa koje su razvili?

DOMAĆI ZADATAK BR. 2

Sami definišite svoj zadatak na postavljenu temu zadatka, i svoj odgovor pošaljite saradniku u roku od 7 dana od održavanja časova vežbi.

Domaći zadatak ima za cilj da utvrdi da li je student, posle časova predavanja i vežbanja, sposoban da samostalno reši zadatak koji je sličan po karakteru i obimu rada zadacima koji su rađeni na vežbama i koji su diskutovani i razjašnjeni na vežbama. Obim zadatka može da zahteva najviše 30 do 40 minuta rada studenta. Svaki student mora da ima poseban zadatak, a na temu koja se ovde definiše. Studenti koji pošalju identična ili vrlo bliska rešenja, neće im ta rešenja biti prihvaćena, tj. dobiće 0 poena. Da bi se to izbeglo, student treba da pri formulaciji svog zadatka na zadatu temu formuliše svoj zadatak za koji postoji mala verovatnoća da će drugi student definisati isti ili vrlo sličan zadatak, na primer, korišćenjem primera koji je njima poznat i koji je verovatno unikatan. Studenti mogu o svojim zadacima da razmenjuju informacije ili preko foruma predmeta ili preko neke od socijalnih mreža. Međutim, ne smeju kopirati rešenja sa Interneta i socijalnih mreža, već moraju da pripreme svoj sopstveni odgovor na zadatak koji su postavili.
Rok za predaju zadataka je 7 dana nakon izdavanja, tj. od dana održavanja vežbi određene lekcije, a posle tog roka umanjuje se ostvaren broj poena za 50%. Krajnji rok za predaju domaćeg zadatka i za studente tradicionalne nastave i za studente onlajn nastave je 10 (deset) dana pre ispitnog roka u kome student polaže ispit.

Tekst domaćeg zadatka br. 2:

1. Definišite konkretan tekst Vašeg zadatka koji bi se odnosio na dole zadatu tematsku oblast.
2. Tematska oblast zadatka: *Koristeći odgovarajući primer (iz svakodnevnog života ili poslovanja), objasnite zašto su međusobno zavisne sledeće aktivnosti softverskog procesa: projektovanje arhitekture (strukture) softvera, projektovanja baze podataka, projektovanje interfejsa i projektovanja komponenti.*
3. Pripremite vaš "esejski" odgovor na zadatak definisan u tački 1, a na temu iz tačke 2, i mejlom ga dostavite saradniku u roku od 7 dana od održavanja vežbi. Ocenjen odgovor vam može doneti do 4 poena. Broj poena se umanjuje za 50%, ako ga dostavite posle navedenog roka.

Poglavlje 5

Zaključak

ZAKLJUČAK

Ključne poruke ove lekcije

1. Softverski procesi su aktivnosti uključene u proizvodnju softverskog sistema. Modeli softverskih procesa su apstraktni prikazi ovih procesa.
2. Opšti modeli procesa opisuju organizaciju softverskih procesa. Primeri ovih opštih modela uključuju model vodopada, inkrementalni razvoj i konfiguraciju i integraciju komponenti za višekratnu upotrebu.
3. Inženjering zahteva je proces razvoja softverske specifikacije. Specifikacije imaju za cilj da saopšte sistemske potrebe korisnika programerima sistema.
4. Procesi dizajna i implementacije bave se transformacijom specifikacije zahteva u izvršni softverski sistem.
5. Validacija softvera je proces provere da li je sistem usklađen sa svojom specifikacijom i da li zadovoljava stvarne potrebe korisnika sistema.
6. Evolucija softvera se dešava kada promenite postojeće softverske sisteme da biste ispunili nove zahteve. Promene su kontinuirane, a softver mora da se razvija da bi ostao

LITERATURA

1. Obavezna literatura:

1. Nastavni materijal za e-učenje na predmetu SE101 Razvoj softvera i inženjera softvera, Univeziitet Metropolitan, školska 2022/23. godina
   - Nastavni materijal je pripremljen korišćenjem reference 2.
2. **Ian Sommervile**, *Software Engineering - Chapter 2 Software Processes, poglavnje 2.1 i 2.2*, Tenth Edition, Pearson Education Inc., 2016.

2. Dopunska litertura

1. **King Graham,** Wang Yingxu, *Software Engineering Processes - Principles and Applications*,CRC Press (2000)
2. **Ian Sommervile,*Software Products*** *- An Introduction to Modern Software Engineering*, Global Edition, Pearson (2021)

3. Veb lokacije :

1. https://www.javatpoint.com/software-processes
2. https://www.educative.io/blog/software-process-model-types