UNIVERZITET METROPOLITAN BEOGRAD

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Upravljanje softverom

Lekcija 08

PRIRUČNIK ZA STUDENTE

SE101 - RAZVOJ SOFTVERA I INŽENJERA SOFTVERA

Lekcija 08

UPRAVLJANJE SOFTVEROM

UVOD

Cilj predmeta i sadržaj ove lekcije

Cilj ove lekcije jeste da vas upozna sa konceptom upravljanja softvera. Kroz ovu lekciju naučićete:

- šta predstavlja i način na koji se vrši planiranje projekta
- kako se sprovodi kontrola projekta razvoja softvera
- šta predstavlja revidiranje životnog ciklusa softvera
- upoznaćete modele, metode i procese razvoja softvera
- kako se upravlja konfiguracijom razvoja softvera i koji su osnovni elementi upravljanja konfiguracijom
- kako se upravlja timovima koji rade na razvoju softvera
- koji su stilovi i mehanizmi za ogranizaciju timova
- kako se upravlja kvalitetom softvera i zbog čega je važno pratiti kvalitet izrade softvera
- kako se vrši procena cene izrade softvera

Poglavlje 1

Uvod u upravljanje softverom

PLANIRANJE PROJEKTA RAZVOJA SOFTVERA

Planiranje projekta u razvoju softvera pomaže lakše i preciznije vođenje softverskog projekta

Pre početka projekta razvoja softvera, on mora biti pažljivo isplaniran. Projektni plan može poslužiti kao vodič tokom projekta. Količina planiranja unapred zavisi od karakteristika problema. Glavni elementi plana projekta su:

1. Uvod - daju se pozadina i istorija projekta, zajedno sa njegovim ciljevima, rezultatima projekta, imenima odgovornih osoba i rezimeom projekta.
2. Model procesa - definisanje tačnog modela procesa koji treba slediti, koje aktivnosti se preduzimaju, koje prekretnice se mogu identifikovati, kako utvrditi da li su te prekretnice dostignute i koji su kritični putevi.
3. Organizacija projekta - u planu projekta mora se navesti koje informacije, usluge, resurse i objekte treba da obezbede korisnici i kada ih treba obezbediti. Unutar projektnog tima mogu se identifikovati različite uloge koje treba jasno razgraničiti i identifikovati odgovornosti svakog od njih. Ako postoje praznine u znanju bilo koje od ovih uloga, potrebno je identifikovati obuku i obrazovanje koji su potrebni za njihovo popunjavanje.
4. Standardi, smernice, procedure - u okviru softverskih projekata potrebna je jaka radna disciplina, u kojoj svaka uključena osoba prati standarde, smernice i procedure koje su dogovorene.
5. Aktivnosti upravljanja - rukovode se ciljevima i prioritetima postavljenim za projekat. Na primer, menadžment će morati da podnosi redovne izveštaje o statusu i napretku projekta. Takođe će morati da prati određene prioritete u balansiranju zahteva, rasporeda i troškova.
6. Rizici - potencijalni rizici moraju biti identifikovani što je ranije moguće mere za njihovo rešavanje moraju biti obezbeđene. Što su različiti aspekti projekta neizvesniji, to su rizici veći.
7. Osoblje - u različitim fazama projekta zahteva se osoblje sa različitim veštinama. Broj i stručnost osoblja neophodno je uzeti u obzir prilikom planiranja softverskog projekta.
8. Neophodno je poznavati metode i tehnike koje će biti korišćene tokom prikupljanja zahteva, projektovanja, implementacije i testiranja.
9. Garancija kvaliteta - definisanje organizacije i procedura koje imaju za cilj osiguravanje da softver koji se razvija ispunjava zahteve klijenata.
10. Radni paketi - veći projekti moraju biti podeljeni na aktivnosti kojima se može upravljati i koje se mogu dodeliti pojedinačnim članovima tima.

11. Resursi - tokom projekta potrebno je mnogo resursa. Hardver, alati i osoblje potrebni za podršku projektu moraju biti navedeni u dokumentu.
12. Budžet i raspored - ukupni budžet za projekat mora biti raspoređen na različite aktivnosti kako je naznačeno u strukturi rada.
13. Promene u softveru su neizbežne. Potrebne su jasne procedure o tome kako će se postupati sa predloženim promenama.
14. Isporuka - moraju se navesti procedure koje treba poštovati pri predaji sistema kupcu.

KONTROLA PROJEKTA RAZVOJA SOFTVERA

Kontrola projekta razvoja softvera obezbeđuje način provere da li svaki aspek projekta ide dobrim putem

Nakon što je plan projekta izrađen i odobren, može se pristupiti realizaciji projekta. Tokom projekta, kontrola se mora vršiti duž sledećih dimenzija (elemenata): vreme, informacije, organizacija, kvalitet i novac.

Napredak projekta razvoja softvera (vremenski aspekt) je teško izmeriti. Potrebno vreme je povezano sa veličinom sistema i ukupnom potrebnom radnom snagom. Deo problema kontrole za projekte razvoja softvera je razmena vremena sa ljudima. Što je više ljudi uključeno, biće potrebno više vremena za koordinaciju i komunikaciju. Nakon određene tačke, dodavanje još ljudi zapravo produžava vreme razvoja.

Informacija kojom se mora upravljati je, pre svega, dokumentacija. Pored tehničke i korisničke dokumentacije, ovo podrazumeva i dokumentaciju o projektu koja obuhvata trenutno stanje poslova, dogovorenih promena i donetih odluka.

Svi članovi razvojnog tima moraju razumeti svoju ulogu u timu i šta se od njih očekuje. Veoma je važno da ova očekivanja budu jasna svim uključenim članovima. Ovi organizacioni aspekti zaslužuju kontinuiranu pažnju rukovodioca projekta. Drugo, organizacija tima i koordinacija uključenih ljudi će zavisiti od karakteristika projekta i njegovog okruženja.

Aspekt kvaliteta je od najveće važnosti. Kupci nisu zadovoljni čisto tehničkim rešenjima koja nude kompjuterski stručnjaci, već žele sisteme koji odgovaraju njihovim stvarnim potrebama. Zahtevi za kvalitet softvera i njegovog razvoja često su u sukobu jedni sa drugima. Tokom projekta procena kvaliteta mora da se vrši redovno, kako bi se mogle preduzeti blagovremene radnje. Kvalitet nije dodatna funkcija, već mora biti ugrađen u tok projekta.

Kontrola troškova (aspekt novca) u velikoj meri znači kontrolu troškova rada. Iako se troškovi hardvera i alata ne mogu zanemariti, oni se obično mogu prilično precizno proceniti u ranoj fazi projekta. Štaviše, oni su obično mnogo manji problem od troškova osoblja. Procena cene softvera stoga znači da se mora proceniti potrebna radna snaga za njegovu izradu.

Upravljanje projektima je veoma dinamična aktivnost. Da bi se na adekvatan način kontrolisao projekat, potrebno je prikupljanje kvantitativnih podataka. Ovi podaci su takođe dragoceni za evaluaciju, kada procenjujemo sadašnji projekat kako bismo poboljšali performanse u projektima koji tek dolaze.

Poglavlje 2

Revidiranje životnog ciklusa softvera

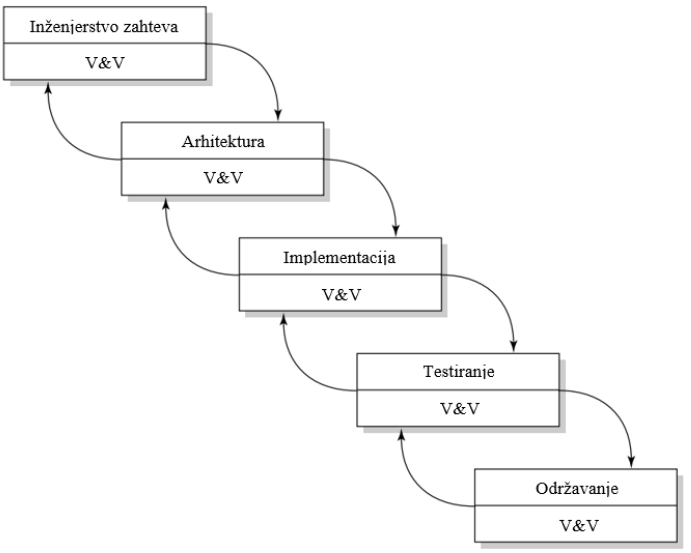
MODEL VODOPADA

Model vodopada prikazuje osnovni metod razvoja softvera

Da bi se mogao proceniti napredak tokom razvoja softvera, odlučuje se za fazni pristup sa nizom dobro definisanih prekretnica. Linearni poredak aktivnosti koji leži u osnovi tradicionalnog modela razvoja softvera,*modela vodopada*, čini ga nemogućom idealizacijom stvarnosti. Pretpostavlja se da se razvoj softvera odvija na uredan, uzastopan način. Pravi projekti se odvijaju na daleko manje racionalne načine.

Model vodopada posebno izražava interakciju između narednih faza. Testiranje softvera nije aktivnost koja striktno prati fazu implementacije. U svakoj fazi procesa razvoja softvera moraju se uporediti dobijeni rezultati sa onima koji su zahtevani. U svim fazama kvalitet se mora procenjivati i kontrolisati.

Na slici 1 koja predstavlja model vodopada, "V & V" označava Verifikaciju i Validaciju. Verifikacija pita da li sistem ispunjava svoje zahteve (da li ispravno gradimo sistem) i na taj način pokušava da proceni ispravnost prelaska na sledeću fazu. Validacija pita da li sistem ispunjava zahteve korisnika (da li gradimo pravi sistem).



Slika 2.1 Model vodopada [1,2]

AGILNE METODE RAZVOJA SOFTVERA

Prikazuje način razvoja softvera korišćenjem agilnih metoda

Kada se koristi striktni metod razvoja, teško je promeniti pravac. Kada je ugovor potpisan, posao razvojnog tima je da isporuči funkcionalnost kako je navedeno u ugovoru. Ako se stvarnost promeni ili korisnik dobije drugačiji uvid, to je teško postići.

Prave agilne metode posmatraju svet kao suštinski haotičan. Pretpostavljaju da je promena neizbežna. Njihov fokus je da isporuče vrednost kupcu što je brže moguće, a ne da se trude oko opsežnih planova i procesa koji se ionako neće poštovati. Ključne vrednosti agilnog pokreta su:

- Pojedinci i interakcije nad procesima i alatima.
- Radni softver ispred sveobuhvatne dokumentacije.
- Saradnja sa klijentima ispred pregovora o ugovoru.
- Reagovanje na promenu u skladu sa planom.

Agilne metode uključuju korisnike u svaki korak. Ciklusi razvoja su mali i inkrementalni. Niz razvojnih ciklusa nije opsežno planiran unapred, ali se nova situacija sagledava na kraju svakog ciklusa. Na kraju svakog ciklusa, sistem je pokrenut i radi. To jest, postoji sistem koji radi, onaj koji pruža vrednost svojim korisnicima. Slika 2 prikazuje agilnu metodologiju koja se naziva Scrum.

Slika 2.2 Scrum metodologija [1,3]

PROTOTIP SOFTVERSKOG PROIZVODA

Prototipa softverskog proizvoda omogućuje prikaz kako će softver raditi bez programiranja

Jedna od glavnih poteškoća za korisnike je da precizno izraze svoje zahteve. Prirodno je pokušati da se nejasnoće razjasne kroz izradu prototipa. Ovo se može postići brzim razvojem korisničkog interfejsa. Potencijalni korisnik tada može da radi sa sistemom koji sadrži komponentu interakcije, ali ne, ili u mnogo manjoj meri, softver koji zapravo obrađuje ulaz. Na ovaj način korisnik može steći dobar utisak o tome šta će mu budući sistem pružiti, pre

nego što se ulože velika ulaganja u realizaciju sistema. Tako izrada prototipa postaje alat za inženjering zahteva. Faze izrade prototipa i kasnije faze proizvodnje su jasno razdvojene. Ovo je prikladno, pošto se koriste različite tehnike tokom same faze proizvodnje, a i potrebno je staviti mnogo više naglaska na dokumentaciju. Čak je izvodljivo da se softverski proizvod ne prenosi iz faze izrade prototipa u fazu stvarne proizvodnje, već da se eksplicitno baci nakon što se faze izrade prototipa završe. Ovo je poznato kao *bacanje prototipa*.

Prednosti izrade prototipova:

- Dobijeni sistem je lakši za korišćenje
- Potrebe korisnika su bolje prilagođene
- Sistem ima manje karakteristika
- Problemi se otkrivaju ranije
- Dizajn je kvalitetniji
- Dobijeni sistem je lakši za održavanje
- Razvoj zahteva manje napora

Mane izrade prototipova:

- Dobijeni sistem ima više funkcija
- Učinak rezultirajućeg sistema je lošiji
- Dizajn je slabijeg kvaliteta
- Pristup izradi prototipa zahteva iskusnije članove tima

RACIONALNI UJEDINJENI PROCES

RUP omogućava korišćenje najboljih strana iz metoda vođenih dokumentima i agilnih metoda

Racionalni ujedinjeni proces je iterativni proces razvoja usmeren ka razvoju objektno orijentisanih sistema. On dopunjuje UML (**Unified Modeling Language**). RUP se može posmatrati kao nešto između metoda vođenih dokumentima i agilnih metoda. Ima dobro definisan proces, uključuje razumno opsežne aktivnosti inženjeringa zahteva unapred, ali naglašava uključenost zainteresovanih strana kroz prirodu vođenu slučajem upotrebe. RUP razlikuje četiri faze: **početak**, **razrada**, **izrada** i **tranzicija**. Unutar svake faze može doći do nekoliko iteracija. RUP razlikuje devet takozvanih tokova posla, kao što su tok posla sa zahtevima i tok rada testa. Ovi tokovi posla grupišu logičke aktivnosti i mogu se proširiti na sve faze, sa različitim nivoima pažnje. Dobre prakse RUP-a su:

1. Iterativni razvoj - Sistemi se razvijaju na iterativni način. Ovo nije nekontrolisan proces već su iteracije planirane, a napredak se pažljivo meri.
2. Upravljanje zahtevima - RUP ima sistematski pristup traženju, prikupljanju i upravljanju zahtevima, uključujući moguće promene ovih zahteva.
3. Arhitektura i upotreba komponenti - Rane faze RUP-a rezultiraju arhitekturom. Ova arhitektura se koristi u ostatku projekta. RUP podržava razvoj sistema zasnovanih na komponentama, u kojima je svaka komponenta složen komad softvera sa dobro definisanim granicama.

4. Modelovanje i UML - Veliki deo RUP-a odnosi se na razvoj modela, kao što su model slučaja upotrebe, model za testiranje, itd. Ovi modeli su opisani u UML-u.
5. Kvalitet procesa i proizvoda - Kvalitet nije dodatak, već odgovornost svih uključenih. Tok rada testiranja je usmeren ka potvrdi da je očekivani nivo kvaliteta ispunjen.
6. Upravljanje konfiguracijom i promenama - Iterativni razvojni projekti isporučuju ogromnu količinu proizvoda, od kojih se mnogi često menjaju. Ovo zahteva dobre procedure za to i odgovarajuću podršku alata.
7. Razvoj zasnovan na slučajevima korišćenja - Slučajevi korišćenja opisuju ponašanje sistema. Oni igraju glavnu ulogu u različitim tokovima posla, posebno zahtevima, dizajnu, testiranju i upravljanju.
8. Konfiguracija procesa - Iako se RUP može koristiti „kao što jeste“, može se i modifikovati da što bolje odgovara specifičnim okolnostima.
9. Podrška alata - Da bi bio efikasan, razvoj softvera treba podršku alata. RUP je podržan širokim spektrom alata, posebno u oblasti vizuelnog modeliranja i upravljanja konfiguracijom.

LINIJE SOFTVERSKIH PROIZVODA

Linije softverskih proizvoda pomažu nam da pravimo softver na način da ga ponovo možemo korisiti

Kada se razvijaju slični proizvodi, treba razmisliti da li se mogu ponovo koristiti elementi ranijih proizvoda tokom razvoja novih. Međutim, to nije navika u razvoju softvera. U mnogim organizacijama ne postoji podsticaj da se ponovo koriste elementi (kod, dizajn ili bilo koji drugi artefakt) iz drugog sistema. Slično tome, nema podsticaja za proizvodnju elemenata za višekratnu upotrebu.

Alternativno, treba razmisliti o pojmu linija softverskih proizvoda, kao skupu softverskih sistema koji dele elemente. U liniji softverskih proizvoda, ponovna upotreba je planirana, a ne slučajna. Da bi se obim održao u razumnim granicama, ova planirana ponovna upotreba je vezana za dati domen.

Pretpostavimo da smo razvili uspešan bibliotečki sistem za biblioteku fakulteta informatike. Šanse su da će od nas biti zatraženo da razvijemo sličan sistem za, recimo, fakultet mašinskih nauka. Ponovo koristimo što je više moguće iz našeg prvog sistema. Takođe je verovatno da je potrebno fino podešavanje da bi se zadovoljile potrebe drugog fakulteta. Sledeće, još jedan fakultet može tražiti treći sistem. Umesto da delujemo reaktivno i ponovo koristimo odgovarajuće elemente iz prethodnih napora, možemo delovati proaktivno i planirati razvoj serije sistema u domenu automatizacije biblioteka od samog početka. Ovaj način rada uključuje dva procesa:*inženjering domena* i *inženjering aplikacija*.

U*inženjeringu domena* analizira se domen za koji se razvija softver. Ovaj proces ima svoj životni ciklus. Rezultat je skup komponenti za višekratnu upotrebu koje čine osnovu za proizvode koji se razvijaju. Obično se proizvodi referentna arhitektura za sve proizvode koji se razvijaju. Važan korak u ovom procesu je odlučivanje o obimu linije proizvoda. Određivanje opsega za linije proizvoda je teško pitanje. Na njega utiče strategija organizacije i

zahteva uvid u verovatnu evoluciju domena. Konačno, proces inženjeringa domena daje plan proizvodnje i vodič za razvoj proizvoda unutar porodice proizvoda.

Aplikacioni inženjering se tiče razvoja pojedinačnih proizvoda. Obično sledi šemu nalik modelu vodopada. Njegovi inputi su izlazi iz procesa inženjeringa domena: referentna arhitektura, plan proizvodnje i skup sredstava za višekratnu upotrebu

Poglavlje 3

Upravljanje konfiguracijom razvoja softvera

UPRAVLJANJE KONFIGURACIJOM

Glavni zadatak upravljanja konfiguracijom je održavanje integriteta ovog skupa artefakata

Upravljanje konfiguracijom se bavi upravljanjem svim artefaktima proizvedenim tokom projekta razvoja softvera. *Osnovna linija* je *„specifikacija ili proizvod koji je formalno pregledan i dogovoren, koji nakon toga služi kao osnova za dalji razvoj i koji se može promeniti samo kroz formalne procedure kontrole promena*“. Dakle, osnovna linija je zajednička baza podataka projekta koja sadrži sve odobrene stavke - stavke konfiguracije. *Stavka konfiguracije* je *„agregacija hardvera, softvera ili oboje, koja je određena za upravljanje konfiguracijom i koja se tretira kao jedan entitet u procesu upravljanja konfiguracijom“*.

Moguće stavke konfiguracije su: komponente izvornog koda, specifikacija zahteva, dokumentacija arhitekture, plan testiranja, slučajevi testiranja, rezultati testiranja i uputstvo za upotrebu.

Način da se uradi upravljanje konfiguracijom - dodavanje ili promena stavki u osnovnoj liniji podleže formalnoj šemi odobrenja. Za veće projekte, ovo je odgovornost posebnog tela, *Kontrolnog odbora za konfiguraciju (ili promene) (CCB)*. CCB obezbeđuje da svaka promena osnovne linije bude propisno odobrena i izvršena. CCB ima osoblje iz različitih strana uključenih u projekat, kao što su razvoj, testiranje i garancija kvaliteta. Svaka predložena promena osnovne linije naziva se zahtevom za promenu i on se može odnositi na detektovanu grešku, na neslaganje između projektnog dokumenta i zahteva itd.

U okviru upravljanja konfiguracijom moraju se definisati taskovi (zadaci) i obaveze koje se dodeljuju članovima tima koji učestvuju u razvoju softvera. *Taskovi* (zadaci) predstavljaju jedinice projekta razvoja softvera koje određeni član tima treba da izvrši dok *obaveze* zahtevaju određenu vrstu delovanja.

Obaveze predstavljaju stvari koje nisu kratkoročne (praćenje dolaska novih zahteva, njihovo odobrenje i kreiranje zadataka za izradu) dok su taskovi jedinice projekta koje su kratkoročne i koje se po završetku zatvaraju kao urađene. Primer: menadžer projekta ima obavezu da prati rad inženjera i sam tok napredka projekta, dodeljuje taskove inženjerima i prati njihovu realizaciju, dok inženjeri dobijaju taskove koje treba da realizuju i time ispune svoju obavezu.

PLAN UPRAVLJANJA KONFIGURACIJOM

Plan upravljanja konfiguracijom predstavlja dokument koji opisuje metode za identifikaciju konfiguracionih stavki

Procedure za upravljanje konfiguracijom su navedene u dokumentu, Plan upravljanja konfiguracijom. Za sadržaj ovog plana prati se odgovarajući IEEE standard (IEEE828, 1990). Ovaj dokument opisuje metode za identifikaciju konfiguracionih stavki, za kontrolu zahteva za promenom i za dokumentovanje implementacije tih zahteva za promenom. Primer sadržaja

Glavni elementi ovog plana su:

- *Upravljanje* Opisuje kako je projekat organizovan. Posebna pažnja je posvećena odgovornostima koje direktno utiču na upravljanje konfiguracijom: kako se obrađuju zahtevi za promenu, kako se završavaju faze razvoja, kako se održava status sistema, kakvi su interfejsi između komponenti...? Takođe, ocrtan je odnos sa drugim funkcionalnim organizacijama, kao što su razvoj softvera i garancija kvaliteta.
- *Aktivnosti* Opisuje način identifikacije i kontrole konfiguracije i kako će se njen status evidentirati. Konfiguracija se identifikuje osnovnom linijom. Takvu konfiguraciju moraju zvanično odobriti sve uključene strane.

Potrebne su jasne i precizne procedure u vezi sa obradom zahteva za izmene ako se želi kontrolisati projekat razvoja softvera. CCB ima odgovornost da proceni i odobri ili odbije predložene promene. Ovlašćenje, odgovornost i članstvo CCB-a moraju biti navedeni. Pošto su softverske komponente obično ugrađene u biblioteku, moraju se uspostaviti i procedure za kontrolu ove biblioteke.
Da bi se kontrolisao projekat razvoja softvera, podaci se moraju prikupljati i obraditi. Informacije koje se obično zahtevaju uključuju: sadašnji status komponenti, verzija i zahtevi za izmenama, kao i izveštaji o odobrenim promenama i njihovoj primeni.

Promene konfiguracionih stavki mogu da utiču na stavke izvan opsega plana, kao što su hardverske stavke. Ove eksterne stavke moraju biti identifikovane i njihovi interfejsi kontrolisani. Na sličan način, interfejsi za stavke razvijene van projekta moraju se identifikovati i kontrolisati.

Poglavlje 4

Upravljanje i organizacija timova

UPRAVLJANJE LJUDIMA

Upravljanje članovima angažovanim na projektu razvoja softvera je od velike važnosti jer loš menadžment ljudi može ugroziti projekat

Tim se sastoji od pojedinaca od kojih svako ima lične ciljeve. Zadatak menadžmenta projekta je da od pojedinaca napravi tim, pri čemu se pojedinačni ciljevi sklapaju u jedan cilj za projekat u celini. Izuzetno je važno je identifikovati ciljeve projekta u ranoj fazi i nedvosmisleno ih preneti članovima projekta. Kada je projekat u toku, učinak članova projekta u odnosu na ciljeve treba da se prati i procenjuje.

U idealnom slučaju, želeli bismo da imamo indikaciju isporučene funkcionalnosti. Produktivnost se uglavnom definiše kao broj isporučenih linija koda po čoveku mesečno. Jedna od velikih opasnosti korišćenja ove mere je da ljudi teže da proizvedu što više koda. Ovo ima veoma štetan efekat. Pisanje manje koda je jeftinije, a ponovna upotreba postojećeg koda je jedan od načina da se uštedi vreme i novac. Praćenje količine koda po osobi mesečno ne daje podsticaj za ponovnu upotrebu softvera.

Jedan od najvećih problema u razvoju softvera je koordinacija aktivnosti članova tima. Kako razvojni projekti postaju sve veći i složeniji, problemi koordinacije se brzo akumuliraju. Da bi se suprotstavio ovim problemima, menadžment formalizuje komunikaciju, pravi formalne sastanke na projektu, vrši inspekcije i sl. Upravljanje timom podrazumeva mnogo aspekata, aliprimarni element treba biti briga o ljudima. Osnovni mehanizmi koordinisanja timova su:

1. Jednostavna struktura - Može biti jedan ili nekoliko menadžera i više ljudi koji obavljaju posao. Mehanizam koordinacije naziva se direktnim nadzorom. Često se nalazi u novim, relativno malim organizacijama. Koordinaciju vrše ljudi, koji su odgovorni za rad drugih.
2. Mašinska birokratija - Izvršavanje i procena zadataka na osnovu preciznih uputstava. Masovna proizvodnja i montažne linije su tipični primeri ovog tipa konfiguracije. Malo je obuke i mnogo specijalizacije i formalizacije. Koordinacija se ostvaruje standardizacijom procesa rada.
3. Divizionizovana forma - Svakoj diviziji (ili projektu) je data značajna autonomija u pogledu toga kako postići cilj. Operativni detalji su prepušteni samoj diviziji. Koordinacija se postiže standardizacijom rezultata rada. Kontrola se vrši redovnim merenjem učinka divizije.
4. Profesionalna birokratija - Ukoliko nije moguće precizirati ni krajnji rezultat ni sadržaj rada, koordinacija se može postići standardizacijom veština radnika. U profesionalnoj birokratiji, kvalifikovanim stručnjacima se daje znatna sloboda u pogledu načina na koji obavljaju svoj posao.

5. Adhokratija - U projektima koji su veliki ili inovativni, posao je podeljen među mnogim stručnjacima. Uspeh projekta zavisi od sposobnosti grupe kao celine da postigne neodređeni cilj na neodređen način. Koordinacija se ostvaruje međusobnim prilagođavanjem.

STILOVI UPRAVLJANJA

Stil upravljanja koji je najprikladniji za projekat razvoja softvera zavisi od vrste posla koji treba da se uradi.

Najpoznatiji stilovi upravljanja su:

- *Stil razdvajanja* - najefikasniji za rutinski rad. Efikasnost je centralna tema. Menadžment se ponaša kao birokrata i primenjuje pravila i procedure. Rad se koordinira hijerarhijski. Donošenje odluka je odozgo nadole, formalno i zasnovano je na autoritetu. Glavna prednost ovog stila je što rezultira stabilnom organizacijom projekta. S druge strane, prave inovacije je teško ostvariti.
- *Stil odnosa* - najefikasniji u situacijama kada ljudi moraju biti motivisani, koordinisani i obučeni. Zadaci koje treba izvršiti vezani su za pojedince. Posao je inovativan, složen i specijalizovan. Donošenje odluka uključuje pregovore i izgradnju konsenzusa. Slaba tačka je da može rezultirati beskonačnim haotičnim sastancima. Sposobnost menadžera da kreira efikasno donošenje odluka je ključni faktor uspeha.
- *Stil posvećenosti* - najefikasniji ako se posao obavlja pod pritiskom. Da bi ovaj stil bio efikasan, menadžer mora da zna kako da postigne ciljeve bez izazivanja ljutnje. Odluke se ne donose na sastancima, već su implicirane zajedničkom vizijom tima u pogledu ciljeva projekta. Potencijalna slaba tačka je to što tim ne reaguje na promene u svom okruženju, već se slepo spotiče na zacrtanom putu.
- *Stil integracije* - odgovara situacijama u kojima je rezultat neizvestan. Rad je istraživačke prirode i različiti zadaci su veoma međusobno zavisni. Zadatak menadžera je da stimuliše i motiviše. Donošenje odluka je neformalno, odozdo prema gore. Ovaj stil promoviše kreativnost, a pojedinci su pred izazovom da izvuku najbolje iz sebe. Moguća slaba tačka je da ciljevi pojedinih članova tima postaju nepovezani sa ciljevima projekta i da počinju da se takmiče jedni sa drugima.

ORGANIZACIJA TIMOVA

Organizacija timova predstavlja najbolji način deljenja uloga u projektu

U svakom timu se mogu razlikovati različite uloge. Postoje menadžeri, testeri, dizajneri, programeri... U zavisnosti od veličine projekta, više od jedne uloge može obavljati jedna osoba ili različiti ljudi mogu imati istu ulogu. Odgovornosti i zadaci svake od ovih uloga moraju biti precizno definisani u planu projekta.

Neke od osnovnih organizacija timova su:

- Hijerarhijska organizacija

- Matrična organizacija
- Tim glavnog programera
- SWAT (Skilled With Advanced Tools) tim
- Agilni tim

Generalni principi za organizaciju timova su:

- Koristiti manji broj kvalitetnih ljudi
- Pokušati da zadatke prilagodite mogućnostima i motivaciji ljudi koji su na raspolaganju
- Na duge staze, organizacija je bolja ako pomaže ljudima da izvuku maksimum iz sebe
- Pametno je izabrati ljude tako da se dobije dobro izbalansiran i harmoničan tim
- Treba ukloniti nekoga ko ne odgovara timu

Poglavlje 5

Upravljanje kvalitetom softvera

MERE I BROJEVI KVALITETA SOFTVERA

Mere i brojevi nam služe da proverimo da li izrada softvera ide u dobrom smeru

Da bi izrazili neki atribut kvaliteta, recimo složenost programskog koda, to se mora uraditi u jednoj numeričkoj vrednosti. Veće vrednosti su namenjene za označavanje složenijih programa. Pošto će složenije programe biti teže razumeti i održavati, ova vrsta informacija je veoma korisna, npr. za planiranje napora održavanja.

Merenje je mapiranje iz empirijskog, „stvarnog" sveta u formalni, relacioni svet. *Mera* je broj ili simbol koji se ovim mapiranjem dodeljuje atributu entiteta. Dodeljena vrednost očigledno ima određenu jedinicu, npr. linije koda. Jedinica zauzvrat pripada određenoj skali, kao što je skala odnosa za linije koda.

Osnovni elementi merenja su:

- Entitet
- Atribut
- Veze između atributa
- Vrednost
- Jedinica
- Tipovi skala: nominalni, redni, intervalni, odnosni, apsloutni
- Model veza atributa

PERSPEKTIVE O KVALITETU

Perspektive o kvalitetu pomažu da se kvalitet softvera sagleda iz više uglova

Korisnici će proceniti kvalitet softverskog sistema po stepenu do kojeg im pomaže da ostvare zadatke i zadovoljstvu pri korišćenju. Menadžer tih korisnika će verovatno suditi o kvalitetu istog sistema prema njegovim prednostima. Ove prednosti se mogu izraziti u uštedi troškova ili boljoj usluzi klijentima.

Tokom testiranja, preovlađujuće dimenzije kvaliteta će biti broj otkrivenih i otklonjenih nedostataka, ili izmerena pouzdanost, ili usklađenost sa specifikacijama. Za programera održavanja, kvalitet će se odnositi na složenost sistema, njegovu tehničku dokumentaciju i slično.

Osnovne definicija kvaliteta softvera:

- Transcendentna definicija - subjektivna procena kvaliteta softvera od strane eksperta
- Definicija zasnovana na korisniku - subjektivna procena kvaliteta softvera od strane korisnika
- Definicija zasnovana na proizvodu - kvalitet se odnosi na kvaliteta. Razlike u kvalitetu su uzrokovane razlikama u vrednostima tih atributa.
- Definicija zasnovana na proizvodnji - odnosi se na usklađenost sa specifikacijama
- Definicija zasnovana na vrednosti - bavi se troškovima i profitom

Programeri softvera imaju tendenciju da se koncentrišu na definicije kvaliteta zasnovane na proizvodima i proizvodnji. Rezultujući zahtevi kvaliteta mogu se izraziti u kvantifikabilnim terminima, kao što je broj otkrivenih nedostataka po čovekom mesecu ili broj odluka po modulu

SISTEM KVALITETA

Standardi o kvalitetu koje softver treba da zadovolji

ISO, Međunarodna organizacija za standardizaciju, razvila je ISO 9000, seriju standarda za sisteme upravljanja kvalitetom. Serija se sastoji od tri dela: ISO 9000:2000, ISO 9001:2000 i ISO 9004:2000.

ISO 9000 daje osnove i rečnik serije standarda o sistemima kvaliteta.

ISO 9001 utvrđuje zahteve za sistem kvaliteta za svaku organizaciju koja treba da pokaže svoju sposobnost da isporuči proizvode koji zadovoljavaju zahteve kupaca.

ISO 9004 sadrži smernice za poboljšanje performansi. Primenljiv je nakon implementacije ISO 9001.

ISO 9001 je generički standard koji se može primeniti na bilo koji proizvod. Koristan dodatak softveru je ISO/IEC 90003:2004 koji sadrži smernice za primenu ISO 9001 na računarski softver. To je zajednički standard ISO-a i IEC-a, Međunarodnog elektrotehničkog komiteta. Opseg ISO/IEC 90003 je opisan kao „Ovaj međunarodni standard utvrđuje zahteve za sistem menadžmenta kvalitetom gde organizacija

- treba da pokaže svoju sposobnost da dosledno obezbedi proizvod koji ispunjava zahteve kupaca i primenljive regulatorne zahteve, i
- ima za cilj da unapredi zadovoljstvo kupaca kroz efikasnu primenu sistema, uključujući procese za kontinuirano poboljšanje sistema i osiguranje usklađenosti sa zahtevima kupaca i primenljivim regulatornim zahtevima"

Standard je veoma sveobuhvatan. Koristi pet perspektiva iz kojih se razmatra upravljanje kvalitetom u softverskom inženjerstvu:

- sistemsku perspektivu - uspostavljanje i dokumentacija samog sistema kvaliteta
- perspektiva upravljanja - definisanje i upravljanje politikama za podršku kvalitetu
- perspektiva resursa - bavi se resursima potrebnim za implementaciju i unapređenje sistema menadžmenta kvaliteta i za ispunjavanje zahteva kupaca i regulatora

- perspektiva proizvoda - bavi se procesima za stvarno kreiranje kvalitetnih proizvoda
- perspektiva poboljšanja - aktivnosti praćenja, merenja i analize radi održavanja i poboljšanja kvaliteta.

Poglavlje 6

Procena cene softvera

CENA SOFTVERA

Cena izrade softvera predstavlja proizod cene po satu i broja sati potrošenih na razvoj

Procena troškova projekta razvoja softvera je prilično neistražena oblast, u kojoj se prečesto oslanjamo na puka nagađanja. Postoje izuzeci od ove procedure, na sreću. Sada postoji niz algoritamskih modela koji nam omogućavaju da procene ukupne troškove i vreme razvoja projekta razvoja softvera na osnovu procena za ograničen broj relevantnih pokretača troškova.

U većini modela za procenu troškova, pretpostavlja se jednostavan odnos između cene i napora. Napor se može meriti u odnosu čovek/mesec, na primer, a za svaki čovek/mesec se uzima fiksni iznos, recimo, od 5000 dolara. Ukupni procenjeni troškovi se onda dobijaju jednostavnim množenjem procenjenog broja čovek/mesec sa ovim konstantnim faktorom.

Pojam ukupnih troškova se obično uzima da ukaže na cenu inicijalnog napora u razvoju softvera, tj. troškove faze inženjeringa zahteva, arhitekture, implementacije i testiranja. Dakle, troškovi održavanja se ne uzimaju u obzir. Vreme razvoja će se smatrati: vreme između početka faze inženjeringa zahteva i trenutka kada je softver isporučen kupcu. Troškovi ne uključuju ni moguće troškove hardvera. To se tiče samo troškova osoblja uključenih u razvoj softvera.

Postoje različiti modeli za procenu troškova izrade softvera:

- *Algoritamski model* - koriste se kompleksi algoritmi bazirani na prethodnim isustvima
- *Valstom-Feliksov model (Walston-Felix)* - baziran na broju linija koda po čoveku po mesecu
- *COCOMO (COnstructive COst MOdel)* - procena cene se vrši na osnovu domena za koji se razvija softver
- *Putnam* - zasniva se na broju problema koji se nalaze u softveru i moraju se prevazići
- *Analiza tačaka funkcije* - zaobilaze se problemi povezani sa određivanjem očekivane količine koda.

UPUTSTVA ZA PROCENU CENE

Koristeći uputstva dolazimo do cene izrade softvera

Jedan od glavnih problema u primeni ovih modela je čist nedostatak kvantitativnih podataka o prošlim projektima. Zato mora da se pređe na druge metode za procenu troškova. Ove druge metode su zasnovane na stručnosti procenitelja.

Jorgensen daje sledeće smernice za ekspertsku procenu napora:

- nemojte mešati procenu, planiranje i licitiranje
- kombinujte metode procene,
- tražiti dodatnka pojašnjenja,
- odaberite stručnjake sa iskustvom iz sličnih projekata
- prihvatite i procenite neizvesnost
- obezbedite prilike za učenje

Iskustva iz prakse govore da se cena izrade softvera najbolje procenjuje na osnovu iskustva inženjera sa prethodnim projektima. Analizirajući zahteve iskusni inženjer može proceniti vreme potrebno za realizaciju i dati vremensku procenu izrade koju menadžment pomnoži sa cenom po satu.

Poglavlje 7

Vežba - Pokazni primeri

UPRAVLJANJE RAZVOJEM SOFTVERSKOG SISTEMA ELEKTRONSKOG DNEVNIKA - PRVI ČAS

Prikaz upravljanja projektom

U okviru časa vežbi, studentima treba prikazati način upravljanja projektom na primeru izrade sistema elektronskog dnevnika, i to prolaskom kroz sledeće tačke:

1. Odabir modela po kome će se sistem izrađivati. Iako je model vodopada dobar za razvoj sistema elektronskog dnevnika, najbolje je koristiti hibridni model (modela vodopada i iterativni model) iz razloga što sistemu može da se pristupi i mobilnim telefonom pa se ti zahtevi češće menjaju zbog potreba tržišta. Kontrolni deo softvera (deo za profesore) se razvija po modelu vodopada dok se softver za mobilne telefone razvija inkrementalno.
2. Zahtev za promenom. Objasniti ulogu CCB-a. (promena autentikacije kao primer)
3. Tehnika upravljanja timovima. Objasniti mašinsu birokratiju kao tehniku koja se koristi (ilzvršavanje i procena zadataka se vrši na osnovu preciznih uputstava). Objasniti podelu na timove (frontend tim, backend tim, database tim, mobile app tim).
4. Objasniti načine ocene kvaliteta od strane eksperta i korisnika. (Ekspert daje svoje mišljene o arhitekturi i samom načinu implementacije dok korisnici ocenjuju sam rad softvera)
5. Objasniti tehnike precene cene izrade sistema. Procena cene izrade softvera vrši se kombinacijom COCOMO i putnam modela procene cene (na osnovu domena za koji se izrađuje softver i broja kompleksnih problema)

UPRAVLJANJE PROJEKTOM - DRUGI ČAS

Zadatak za grupni rad studenata

Prema stavkama navedenim na prethodnom času studenti grupno analiziraju i daju predloge za način upravljanja projektom izrade softvera za biblioteku. Sistem se sastoji iz dva dela: administratorski deo, za zaposlene u biblioteci i korisnički deo za korisnike biblioteke koji naručuju knjige.

Poglavlje 8

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.

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 rešenjabiti prihvaćena, tj. dobiće 0 poena.

Tekst domaćeg zadatka:

1. Odaberite softverski sistem za koji ćete definisati način upravljanja izrade softvera (različit za svakog studenta).
2. Prikažite i objasnite koji je metod izrade softvera najpogodniji za taj sistem.
3. Prikažite na koji način se upravlja konfiguracijom sistema
4. Prikađite kako su organizovani timovi za izradu datog softvera.
5. Objasnite kako se definiše kvalitet softvera za dati sistem.
6. Objasnite kako se procenjuje cena izrade predloženog softvera.

Poglavlje 9

Zaključak

ZAKLJUČAK

Sumiranje stečenih znanja

- Pre nego što se krene u projekat razvoja softvera, on mora biti pažljivo isplaniran što uključuje procenu svojstava projekta koja mogu uticati na proces razvoja.
- Aspekti kontrole (aspekat informacije, organizacije time, kvaliteta i kontrole troškova) u velikoj meri utiču na sam razvoj projekta.
- Model vodopada razvoja softvera pogodan je u slučaju jasno definisanih i nepromenljivih zahteva dok su agilne metode prihvatljivije za sisteme kod kojih se zahtevi često menjaju i dolaze novi.
- Prototipovi se izrađuju da bi se lakše potvrdile funkcionalnisti sistema sa klijentima.
- Racionalni objedinjeni proces je iterativni proces razvoja, usmeren ka razvoju objektno orijentisanih sistema
- Linije softverskih proizvoda predstavljaju način izrade softvera da bi mogao ponovo da se koristi
- Upravljanje konfiguracijom se bavi upravljanjem svim artefaktima proizvedenim tokom projekta razvoja softvera
- Taskovi i obaveze predstavljaju odgovornosti koje članovi tima imaju u razvoju softvera
- Plan upravljanja konfiguracijom predstavlja procedure za upravljanje konfiguracijom
- Upravljanje ljudima je od velike važnosti jer loš menadžment timova može ugroziti projekat dok dobra koordinacija čini projekat uspešnim
- Stil upravljanja koji je najprikladniji za projekat razvoja softvera zavisi od vrste posla koji treba da se uradi
- Merenje kvaliteta softvera predstavlja način provere da li softver radi ono što se od njega zahteva. Postoje standardi kvaliteta koje softver treba da zadovolji.
- Cena izrade softvera pretpostavlja se jednostavan odnos između cene i napora i daje se u čovek/mesec ili čovek/sat jedinicama
- Procenu cene softvera uglavnom vrše iskusni inženjeri na osnovu prethodnih projekata

LITERATURA

Pogodna literatura za učenje

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. Software engineering: Principles and Practice, Hans van Vliet, 2007.
3. Permana, P. A. G, "Scrum Method Implementation in a Software Development Project Management", Bali, International Journal of Advanced Computer Science and Applications, 2015

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)