UDC: 004.41 INFO M: str. 11-17

POSTUPAK EFIKASNOG PLANIRANJA RAZVOJA SLOŽENOG INFORMACIONOG SISTEMA
EFFICIENT PLANNING PRATICE OF COMPLEX INFORMATION SYSTEM DEVELOPMENT

Biljana Ribić

REZIME: U radu se razmatra problem efikasnog planiranja razvoja složenog informacionog sistema u kontekstu različitih modela procesa razvoja softvera i različitih planskih startegija. Za strukturiranje složenih sistema osim analize i dekompozicije naglašava se značaj postupka generalizacije.
KLJUČNE REČI: Projekt menadžment, plan softverskog projekta, proces razvoja softvera, analiza, dizajn, dekompozicija, generalizacija
ABSTRACT: This paper considers the problem of efficient planning pratice for complex information system development in context of different models of software development process and different planning strategies. Beside analysis and decomposition, it is emphasized importance of generalization process for complex system structure definition.
KEY WORDS: project management, software project plan, software development process, analysis, decomposition, generalisation

1. UVOD

Savremeni trend da se, u svim oblastima rada, pa tako i u oblasti softver inženjeringa, poslovi organizuju i vode kao projekti, zahteva veštinu projekt menažmenta kao preduslov uspeha. Nasuprot tekućem poslu, projekat se definiše kao privremeni posao kojim se pravi odredjen i proizvod ili usluga. Trajanje projekta je vreme potrebno za njegovo kompletiranje. Projekt menadžment treba da definiše i dostigne ciljeve kroz optimizaciju resursa u toku trajanja projekta. To se realizuje kroz kontrolu potrebnog vremena i troškova. Projekt menadžment obuhvata različite aktivnosti kao pisanje predloga projekta, inicijalnu procenu troškova i ostalih resursa , planiranje kompletnog posla, procenu rizika, proračun resursa, organizaciju posla, angažovanje resursa i dodelu poslova, upravljanje aktivnostima, kontrolu izvršenja, izveštavanje i analizu rezultata, prezentacije.

Ključna aktivnost za uspešno vodjenje projekta je svakako planiranje. Projektni plan treba da strukturira posao, tj da definiše projektne aktivnosti, njihovo trajanja i medjuzavisnosti, da odredi vremenski redosled svake aktivnosti, rasporedi ljude i druge resurse na aktivnosti, proceni i predupredi rizike, te da obezbedi mogućnost praćenja izvršenja aktivnosti kao i povremenu reviziju svih predefinisanih parametara. On se koristi od strane menadžmenta da bi se ustanovilo da će projekat zadovoljiti njihova očekivanja i potrebe, da se prati status projekta, da se verifikuje raspored i cena i planiraju timske aktivnosti. Sa druge strane, članovi tima ga koriste da bi razumeli kontekst svog dela posla.

Za planiranje i praćenje projekta, menadžer obično koristi egzaktne modele implementirane u odgovarajućim alatima za upravljanje projektima. Medjutim, uspešnost plana ne zavisi toliko od korištenja ovih alata i metoda koliko od iskustva i veštine projekt menadžera da razume suštinu i prirodu konkretnog projekta i njegovog okruženja, i to ne samo poslovnog i tehnološkog nego i kulturnog, pa i samih ličnosti ljudi koji odlučuju o projektu ili su na njemu angažovani. Zlatno pravilo za uspešan projekt menadžment je dobro poznavanje suštine posla i samog procesa rada zbog definicije jasnih projektnih ciljeva, uskladjenog delokruga i strukture celog posla.

Upravljanje softverskim projektom je dodatno otežano zbog specifičnih osobina i zahteva u odnosu na softver kao proizvod. Naime, softverski projekat se značajno razlikuje od klasičnog tehničkog projekta po sledećim osobinama

- Namena i struktura konkretnog softvera kao proizvoda nije unapred definisana. Potrebno je uraditi dobar deo analize i dizajna da bi se dobili elementi plana.
- Proizvod je žneopipljiv'. Teško je videti rezultat. Teško je proceniti koliki deo posla je obavljen. Menadžer se mora pouzdati u apstraktne dokumente.
- Projekt je obično žneponovljiv'. Stara iskustva obično nisu primjenjiva,. pojavljuju se nepredvidjeni problemi a i tehnologija se veoma brzo menja.
- Nema standardnog procesa razvoja softvera. Postoje razne metode i alati, ali se ne zna šta je najpogodnije u zadatim okolnostima.

Sa druge strane, od softverskog proizvoda se očekuje da ima sledeće atribute kvaliteta:

- Mogućnost održavanja tj. mogućnost izmene softvera u skladu sa promenom potreba korisnika
- Pouzdanost i sigurnost tj. softver se mora ponašati na predvidiv način i ne sme izazivati fizičke ili ekonomske štete.
- Efikasnost tj. softver mora imati zadovoljavajuće performanse
- Upotrebljivost tj. softver treba raditi ono što korisnici od njega očekuju, treba imati pogodan interfejs i odgovarajuću dokumentaciju.

Iako se više ne može tvrditi da je razvoj softverskih projekta relativno nova disciplina, bez dovoljno duge istorije i iskustva, alati i tehnike koje se koriste pokazali su se samo delimično korisnim. Projekti se još uvek u značajnoj meri pokazuju kao neuspešni, bilo zbog nedovoljnog kvaliteta softvera, ili potrebe za velikim naknadnim ispravkama, promašenim ciljevima, prekoračenim rokovima i–ili troškovima,

neadekvatnim vodjenjem projekata, te visokim procentom odustajanja od daljeg rada na projektu. Zapravo sve vreme postoji realna potreba za unapedjenjem procesa razvoja softvera jer se konstantno 80 do 90 % projekata ocenjuje kao neuspešno. Situacija je možda još ozbiljnija jer se ovaj procenat uglavnom odnosi na projekte koji nisu realizovali postavljene ciljeve, ne uključujući ostalo. Nažalost, uspeh se češće meri sa isporukom u kratkom roku nego sa mogućnošću dugotrajnog održavanja.

Decenijama se pokusava definisati ponovljiv i predvidljiv proces ili metodologija koja unapredjuje produktivnost i kvalitet. Sa jedne strane, pokusavaju se sistematizovati i formalizovati konkretni poslovi koje obuhvata razvoj softvera, dok se sa druge strane primenjuju tehnike project management-a. A softverski projekat čine i plan, i razvoj, i upravljanje, i klijenti, u interreakciji.

Obično se od strane menadžera koji nastoje uspostaviti bolju kontrolu smatra da je više planiranja rešenje za sve probleme. A više planiranja ne znači i bolje planiranje - kad se problem pokušava rešiti sa više lošeg planiranja, on se samo komplikuje. Da bi se izbegla zamka lošeg planiranja treba pojednostaviti projektne zahteve i svesti ih na esencijalne u smislu da se budućem softveru odredi realno mesto unutar postojećeg sistema, odnosno da se izbalansiraju uloge softvera i ostalih elemenata sistema. Dodatni uzrok problema je što se u naporu da se minimizira rizik, usvoji najbolja praksa, isprati tehnološki razvoj, udovolji korisničkim zahtevima i budžetskim ograničenjima, uvode isuviše komplikovane šeme koje su daleko od produktivnog planiranja. Osnovno je da se sam proces pojednostavi i koncentriše na jednostavan i efektivan model u toku faze planiranja.

2. PLAN SOFTVERSKOG PROJEKTA

Dobar plan softverskog projekta prvenstveno obuhvata:

- studiju - analizu poslovnih potreba i konkretnih merljivih ciljeva,
- pregled postojeceg, važećeg sistema,
- konceptualni dizajn operativnosti novog sistema

zatim zahteve za opremom, finansijsku analizu uključujući i budžet, izbor članova projektnog tima. Plan osim toga uključuje i definiciju konkretnih zadataka, isporučivih jedinica i vremenskog rasporeda. Takodje se, uskladjeno sa procesom razvoja predvidja i faza testiranja kojom se verifikuje da proizvod zadovoljava potrebe korisnika, da su funkcije u skladu sa dizajnom, da je proizvod operativan sa hardverom i ostalim softverom i da nema gresaka. Faza implementacije obuhvata konverziju, dokumentovanje i obuku. U poslednjoj fazi postimplementacije mora se predvideti nastavak podrške korisnicima, ispravljanje grešaka i po potrebi ažuriranje i dopuna softvera. Ovo praktično znači da se detaljan plan može napraviti samo na bazi rezultata početnih aktivnosti samog procesa razvoja, odnosno da je konceptualni dizajn operativnosti sistema opredeljujući za ceo naredni posao.

Plan softverskog projekta u delu strukturiranja posla, procene i definicije redosleda aktivnosti bitno zavisi karakteristika samog procesa razvoja softvera i izbora konkretne metode razvoja ali i od primenjene planske strategije. Ovaj izbor projekt menadžer pravi u zavisnosti od ličnog znanja i iskustva, veličine i nivoa kritičnosti projekta, procene učestalosti budućih zahteva za promenama, zatim od brojnosti, znanja i iskustva članova tima kao i kulturnog obrasca u okruženju projekta. Postoje različiti modeli procesa razvoja softvera koji odredjuju poželjni način odvijanja i medusobnog povezivanja osnovnih aktivnosti – sekvencijalno ili simultano. Pojedinačne metode (funkcionalno ili objektno orijentisane), kao konkretizacija odabranog modela, uvode specifičnu terminologiju, dele osnovne aktivnosti u pod-aktivnosti i propisuju obuhvat i način dokumentovanja pojedine pod-aktivnosti. Sa druge strane, samom poslu planiranja takodje se može pristupiti na različite načine, odnosno u primeni su različite planske strategije. U praksi se obično bira usaglašena kombinacija izbora strategije planiranja i modela razvoja, a zavisno od procene i umeća projekt menadžera, primenjuju se i suprotne kombinacije. Zbog same prirode softverskog projekta, ovaj izbor utiče na kvalitet i strukturu proizvoda i krajnji uspeh projekta u mnogo većoj meri nego u drugim poslovima.

3. KARAKTERISTIKE PROCESA RAZVOJA SOFTVERA

U svim modelima procesa razvoja softvera, više ili manje su prisutne sledeće osnovne aktivnosti: specifikacija, dizajn, implementacija, verifikacija i validacija, i na kraju, održavanje-evolucija.

Analiza (specifikacija) i dizajn se prvenstveno bave problemom savladjivanja složenosti realnih sistema što se u velikoj meri u klasičnom pristupu svodi na dekompoziciju većih delova u manje u skladu s jednim od dva aktuelna pristupa:

- Funkcionalni pristup modelira sistem kroz funkcije (procese) tj. delovi su moduli i pojedine funkcije. Kreće se od procesa na najvišem nivou, koji se postepeno strukturiraju do hjerarhijskog sistema koji se lako implementira u klasičnim programskim jezicima. Stanje sistema se sadrži u globalnim podacima koji su dostupni svim funkcijama.
- Objektna analiza i dizajn modeliraju sistem razvojem objektnih modela uočavanjem objekata i klasa. Analiza je manje precizna i koncentrisana na modele aplikativnog domena, a se dizajn bavi modeliranjem budućeg softvera, do detalja svih potrebnih interfejsa.

Aktivnosti specifikacije i dizajna su od posebnog značaja za plan projekta jer se njima definiše kontekst, delokrug i namena budućeg softverskog sistema kao i njegova struktura, tj. njegovi podsistemi kao elementi plana.

Specifikacija :

Specifikacija je početna faza softverskog procesa. Analizom zahteva korisnika utvrduje se šta softver treba da radi, po mogućnosti bez prejudiciranja kako da se to postigne. Implicitno se utvrdjuje i šta softver ne treba raditi, odnosno utvrdjuje se kontekst i delokrug, a time i veličina softverskog projekta, kao bitni elementi plana projekta. Same zahteve delimo na dve vrste:

- Funkcionalni zahtevi koji opisuju funkcije sistema, odnosno usluge koje bi on trebao obavljati, izlaze koje sistem daje za zadate ulaze, kao i njegovo ponašanje u konkretnim situacijama.
- Nefunkcionalni (sistemski) zahtevi izražavaju ograničenja na funkcije sistema, na primer poštovanje odredjenih standarda, vreme odziva, nivo pouzdanosti, obavezu korištenja odredjene softverske ili hardverske tehnologije itd.

Rezultat faze specifikacije su

- korisnički zahtevi kao tekst prilagoden krajnjim korisnicima, koji opisuje funkcije sistema i ograničenja pod kojima sistem radi.
- sistemski zahtevi kao detaljan i precizan opis funkcija sistema i ograničenja koji često služi kao osnova za ugovor izmedu kupca i proizvodjača softvera.
- analitički modeli sistema, kao sredstvo bolje komunikacije sa korisnicima.

U fazi specifikacije se često pojavljuju različiti problemi pogotovo ako novi sistem treba da promeni ustaljeni način rada jer nema odgovarajućeg iskustva na kojem se zasnivaju zahtevi. Korisnici su obično uvereni da znaju šta softver treba da radi, pa je tim pre potrebna velika veština i iskustvo da se izdvoje nekompletni, nejasni i kontradiktorni zahtevi. Konačni zahtevi su neizbežno kompromis. Takodje, finansijeri sistema i ostali žstakeholderi' i njegovi korisnici često nisu isti ljudi pa se sa korisničkim zahtevima sukobljavaju zahtevi motivisani budžetskim, organizacionim ili drugim ograničenjima. Na kraju, poslovno, tehničko a kod nas i zakonsko okruženje se često menja, a time se menjaju i zahtevi.

Ako se zahtevi na drugi način ne mogu jasno definisati, pokušava se metodom izrade prototipa - jednostavno i brzo razvjenog programa, koji oponaša budući sistem, lako se menja i nadogradjuje i pruža mogućnost da se isprobaju razne ideje i opcije na zahtevima koji su nejasni i koje treba istražiti.

Dizajn:

Dizajn odnosno modeliranje softverskog sistema je kreativna aktivnost koju je teško precizno opisati. U okviru dizajna modelira se softverska arhitektura kao apstraktna reprezentacija sistema, tj. struktura sistema, način rada komponenti i definišu se interfejsi izmedju komponenti. Ovim se projektuje rešenje koje odredjuje kako će softver raditi. Za izradu detaljnog plana projekta, neophodni su barem početni rezultati ove aktivnosti, tzv. konceptualni dizajn.

Dizajn u užem smislu obuhvata dizajn arhitekture, apstraktne specifikacije, dizajn interfejsa, dizajn delova sistema, dizajn struktura podataka i algoritama. U dizajnu arhitekture se uočavaju i dokumentuju podsistemi i njihove veze u smislu medusobne komunikacije i medusobne kontrole. Podsistem se definiše kao relativno samostalna celina, čje funkcionisanje uglavnom ne zavisi od servisa koje pružaju drugi podsistemi. Specifikacije za svaki podsistem podrazumevaju apstraktni i precizni opis obuhvaćenih zahteva. U sklopu dizajna interfejsa, oblikuje se i dokumentuje interfejs prema drugim podsistemima i prema korisniku. Dizajn delova znači da se svaki podsistem dalje rastavlja na sastavne delove, koji se zasebno opisuju. U dizajnu struktura podataka i algoritama opisuju se složene strukture podataka i algoritmi koji pripadaju u sistemu.

Očigledno je da dizajn softvera bitno utiče na njegov kvalitet. Pristup zavisi od toga koji od atributa kvaliteta softvera se u konkretnom slučaju smatra važnijim. Na primer, ako je bitnija efikasnost (performanse), tada je dobro da se dizajn sastoji od malog broja relativno velikih delova, tako da se smanji obim njihove komunikacije. Pošto razne funkcije direktno pristupaju podacima, ovde postoji potreba za globalnim podacima.Takav dizajn se dobija funkcionalnim pristupom. S druge strane, ako prvenstvo ima mogućnost održavanja, tada je dobro da se dizajn sastoji od velikog broja malih delova sa visokom unutrašnjom kohezijom i labavim spoljnim vezama. Takav dizajn dobija se primenom objektnog pristupa. Objektni dizajn podrazumeva pod-aktivnosti koje su u priličnoj meri isprepletene i teško ih je obavljati u nekom fiksiranom redosledu što znači i precizno planirati na tom nivou. Sistem se modelira kao skup objekata u medusobnoj komunikaciji. Intereakcija objekata odvija se na način da jedan objekat (kljent) pokrene operaciju drugog objekta (servera) a prilikom poziva operacija, objekti razmjenjuju podatke (parametre i rezultate). Struktura sistema obično nije hjerarhijska i može se posmatrati iz perpektive nasljedivanja, agregacije, odnosno korištenja operacija medu klasama.

Za izradu plana najbitnija je ta definicija strukture sistema iz koje se definišu razvojni koraci i njihov redosled. Praktično odluke koje se donose u ovoj fazi imaju presudni uticaj na ključne parametre kojima se ocenjuje uspešnost jednog softverskog projekta.

Ostale aktivnosti procesa razvoja softvera:

Ostale aktivnosti procesa razvoja mogu da utiču na plan softverskog projekta, ali u mnogo manjoj meri. Implementacija (programiranje) podrazumeva da se modelirano rešenje realizuje tj. kodira uz pomoć raspoloživih programskih jezika i alata. Ovo je uobičajen i često najveci deo posla. Na plan u ovoj fazi može da utiče izbor alata broj i kvalitet raspoloživih programera . Verifikacija i validacija obuhvata proveru da li softver radi prema specifikaciji, odnosno da li radi ono šta korisnik želi. Obično se svodi na testiranje, mada postoje i druge tehnike. Testiranje se obično strukturira na 3 nivoa: razvojno, korisničko i integrativno. Održavanje odnosno evolucija obuhvata dalje popravke, menjanje i nadogradjnju softvera nakon uvodenja u upotrebu, da bi se obuhvatili novi problem, u skladu s promenom ili novim shvatanjima potreba korisnika. Obično se za svaki projekat, zavisno od njegovog obima predvidi jedan period za neophodne inicijalne dorade (tzv. postimplementacija). Značajno je istaći da, prema aktuelnim statistikama, održavanje čini oko 2/3 ukupnog posla u oblasti softver inženjeringa. Sve ove aktivnosti podrazumevaju i dokumentovanje – kao veoma važnu stavku koja se planom treba eksplicitno predvideti, a od posebnog je značaja za eksterne interfejse i za poslove naknadnih održavanja i proširenja.

4. MODELI PROCESA RAZVOJA SOFTVERA

Različiti modeli razvoja softvera se razlikuju po načinu odvijanja i medusobnog povezivanja osnovnih aktivnosti procesa razvoja i po načinu i obimu dokumentovanja, pa se sa tim uskladjuje tehnika planiranja projekta.

Model vodopada (Waterfall) strukturira softverski proces kao niz vremenski odvojenih aktivnosti koje se odnose na razvoj celog sistema i na sledeći razvojni korak prelazi se tek kad se potpuno završi prethodni. Prednosti ovog modela su u tome što je moguće jednostavno praćenje stanja u kojem se softverski proces nalazi i dobro je prihvaćen od rukovodstva jer dozvoljava da se softverski projekat planira po istom principu kao ostali. Nažalost, postoje i značajne mane ovog modela jer je u praksi teško razdvojiti faze razvoja pa dolazi do naknadnog otkrivanja grešaka i vraćanja u prethodne faze. Pri tom je dodatna teškoća medjuzavisnost projektnih izlaza. Takodje, rezultati posla nisu dostupni korisnicima sve do faze testiranja, samim tim nije moguća korekcija eventualnih grešaka u ranim koracima i nakon celog procesa može se ustanoviti da postoje greške koje su teško otklonive.Zbog poštovanja rokova pojedine faze se moraju jednom fiksirati kao završene i dešava se da je sistem u trenutku puštanja u rad već neažuran i zastareo u odnosu na promene realnih procesa. Ovo je posebno izraženo kod velikih projekata jer posao kompletne specifikacije i dizajna zahteva dug vremenski priod u toku kojeg se realni sistem obično promeni. Zato se ovaj model može koristiti samo za one velike sisteme gde postoje relativno jasni i stabilni zahtevi.

Model evolucionog (spiralnog) razvoja predvidja da se osnovu približnog opisa problema razvije inicijalna verzija sistema koja se pokazuje korisniku. Na osnovu korisnikovih primedbi, ta verzija se poboljšava, opet pokazuje, itd. Konačna verzija sistema se dobija nakon dovoljnog broja iteracija. Unutar svake iteracije, osnovne aktivnosti razvoja se obavljaju simultano i ne mogu se eksplicitno razdvojiti. Prednosti ovog modela su u tome što on proizvedi brz odgovor na zahteve korisnika a postoji mogućnost da se specifikacija postepeno razvija u skladu sa sve boljim korisnikovim razumevanjem problema. Ali, proces nije transparentan za menadžere jer se ne može egzaktno oceniti koliki deo posla je napravljen i kad će sistem biti gotov. Takodje, konačni sistem nije dobro strukturiran zbog stalnih promena, i nepogodan je za održavanje. Model je pogodan za razvoj web sajtova, i za male sisteme s kratkim životnim vekom, pogotovo za sisteme s nejasnim zahtevima.

Model usmeren na ponovnu upotrebu (reuse-oriented) polazi od pretpostavke da već postoje gotove i upotrebljive softverske celine ili delovi prethodno razvijenih sistema. Novi sistem se nastoji u što većoj meri realizovati spajanjem postojećih delova, u skladu sa primenom iste ideje u drugim tehničkim disciplinama. Prednosti ovog modela su u smanjenju količine softvera koji treba razviti, a time i potrebnog vremena, troška i rizika jer se oslanja na proverene i testirane delove softvera. Mane su što zbog kompromisa u specifikaciji sistem obično u potpunosti ne odgovara stvarnim potrebama korisnika niti postoji potpuna kontrola nad evolucijom sistema, pošto se ne upravlja razvojem novih verzija korištenih delova. Ipak, pošto je broj gotovih rešenja sve veći, a zahtevaju se sve kraći rokovi, ovaj model se sve više koristi.

Model inkrementalnog razvoja podrazumeva da se sistem razvija u nizu iteracija, ali pojedina iteracija ne doteruje već realizovani deo sistema, već mu dodaje sasvim novi deo - inkrement. Izbor modela za razvoj pojedinog inkrementa unutar iteracije nije ograničen. Pristup je zasnovan na realizaciji inicijalno malih delova softverskog projekta koji dopuštaju da se greške i problemi rano otkriju. Ovakav proces dozvoljava da se ciljevi ostvare i ako korisnik ne zna kako da definiše šta hoće. Takodje je prednost što je proces dovoljno transparentan za menadžere – jasno je do kojeg inkrementa se stiglo. Korisnici ne moraju dugo čekati da bi dobili prvi inkrement koji zadovoljava njihove najbitnije potrebe i rano su u prilici da konkretno provere i verifikuju osnovnu koncepciju celog posla.

Osnovne mane ovog modela su u tome sto je često veoma teško podeliti korisničke zahteve u smislene inkremente i dodeliti im odgovarajući stepen prioriteta. Takodje, pošto svi zahtevi nisu dovoljno razradjeni na početku, teško je odrediti zajedničke module koji se koriste u raznim inkrementima i koji bi trebali biti implementirani u prvom. Ipak, ovo je veoma popularan i upotrebljiv model koji se intenzivno koristi u praksi. Na ovom modelu je zasnovan i Unified Software Development Process koji je u različitim modifikacijama od strane velikih proizvodjača postao defacto standard u softver inženjeringu. Osnovne karakteristike ovog procesa su da je on iterativan i inkremantalan, usmeren arhitekturom i upravljan slučajevima korišćenja. Njegovom primenom dobijaju se izvršne verzije sistema sa sve većim brojem realizovanih funkcija, čime se tokom trajanja razvoja sistema, polako smanjuje rizik. Prednost je što je ovaj proces dobro dokumentovan, tačno je definisano šta se u kojoj fazi dobija i u potpunosti je podržan softverskim alatima (sada vec raznih softverskih kuća) i templejtima.

Model agilnog razvoja softvera izgradjen je na osnovu iterativnog, ali se zasniva na pristupu koji je više okrenut direktnoj komunikaciji sa ljudima. Ovaj proces kao svoj primarni kontrolni mehanizam koristi povratne informacije iz regularnog testiranja svakog release-a softvera, a ne planiranje. Većina agilnih metoda pokušava minimizirati rizik razvojem softvera u kratkim vremenskim intervalima, u iteracijama koje traju 1 do 4 nedelje. Svaka iteracija je kao minijaturni softver projekat koji uključuje zadatke potrebne da se realizuje mini inkrement nove funkcionalnosti : planiranje, analizu zahteva, dizajn, kodiranje, test i dokumentovanje. Agilne metode kao primarnu meru napretka projekta uzimaju softver koji radi i usmerene su na brzu adaptaciju realnim promenama. Medjutim, zbog preference na lične komunikacije, ovaj metod obezbedjuje veoma malo pisane dokumentacije. Extreme programming je najpoznatiji agilni proces. Specifikacija, dizajn i kodiranje se kod ovakvog razvoja prepliću. Danas, kada se traže razni lightweight modeli upravljanja projektima, uvodi se cela disciplina extreme project managementa kao kombinacija modeliranja procesa i

principa upravljanja medjuljudskim interreakcijama. Takodje, smatra se da se agilni razvoj može podvesti pod pojam modifikacije Unified Software Development Process-a i definiše se tzv. AUP - Agile Unified Process. Prednosti agilnog procesa su u tome što izgleda efikasniji od ostalih, ali je ozbiljan nedostatak sa poslovne strane što ne obezbedjuje dugoročno planiranje. Agilne metode su primenjljive za male razvojne grupe (manje od 10) sa velikim iskustvom, na jednoj lokaciji, Treba ih primeniti za projekte koji nisu visoko kritični a imaju visoku frekvenciju mogućih izmena. Teško su primenjljive na timove veće od 20 ljudi, dislocirane i pogotovu ako ljudi nemaju dosta iskustva, kao i za visoko kritične projekte u relativno stabilnom, na strogi red naviknutom okruženju.

5. OPŠTE STRATEGIJE PLANIRANJA

Uobičajeni, klasični pristup poslu planiranja projekta podrazumeva da se primeni **sekvencijalna tj. linearna tehnika planiranja projekta**: Ceo posao se strukturira po fazama koje se do detalja planiraju za sekvencijalno izvršenje. Jedna faza projekta mora da se završi da bi sledeća započela. Ovakva strategija osigurava da se pre prelaska na sledeću fazu ima detaljno uradjen plan svake faze posla. Nekako se podrazumeva da to obezbedjuje kvalitet čitavog procesa, ali to istovremeno zahteva da se svi detalji unapred znaju i-ili mogu predvideti, što veoma često nije slučaj. Provere uspešnosti projekta se obično obavljaju na kraju svake faze pri čemu se donosi odluka da se projekat nastavlja ili prekida.

Kod mnogih projekata koji su od samog pocetka planirani do najnižeg nivoa detaljnosti, protekom vremena javljaju se nepredvidjene aktivnosti koje prvobitni plan čine nevažećim pa se on iznova i iznova revidira i preispituje. Sa tačke gledišta rukovodstva stiče se utisak kao da tim od pocetka nije najbolje znao šta treba da radi (što ne mora uvek biti slucaj). Zbog toga je došlo do razvoja drugačijih, novijih strategija planiranja projekata, za koje, u slučaju softverskih projekata postoji analogija sa novijim modelima procesa razvoja.

Tzv. **strategija ROLLING VAWE** (talasanje) u planiranju prihvata činjenicu da će se stvari tokom realizacije projekta promeniti, tako da se detaljno planira samo prvi žtalas', a ostale faze koji slede, planiraju se na višem nivou detaljnosti. Kao pocetak 'talasa' se uzimaju odredene kritične tacke u projektu: ili početak kalendarskog perioda, ili jedna faza projekta ili dobijanje finansijskih sredstava ili potpis ugovora itd. Sa aspekta kontrole rukovodstva, ovako vodjen projekat može biti lako otkazan ili preispitan bez velikog gubitka vremena u detaljnom planiranju. Ukoliko je odluka da se projekat nastavi, tek se onda kreće u detaljno planiranje ali samo sledeće faze (talasa). Kako projekat napreduje, plan se često revidira, ali se planiranje ne odnosi na velik deo posla. Ideja je u tome što se na početku ne može sve unapred predvideti, tako da se prvi plan pravi samo opciono. Detaljno se planira deo posla koji se završava u sledećih par nedelja. Naravno da se u raspored ubacuju fiksni datumi ako su zahtevani, ali ovakvim načinom, iskustvo stečeno radom na projektu ugradjuje se u plan svake sledeće faze i često se ostvaruju mnogo bolji efekti u poštovanju rokova.

Za planiranje realizacije projekta koji treba da se realizuje u zadatom vremenskom roku i sa zadatim resursima koristi se **strategija TIME BOXING** (vremenskog ogranicenja). Prethodne strategije ne odgovaraju najbolje projektima koji kao faktor uspeha imaju 'vreme-do-tržišta' ili striktno definisan 'rok i budžet'. Time boxing strategija planira realizaciju projekta SAMO na osnovu aktivnosti koje su neophodne da bi projekat profunkcionisao. U ovom scenariju vreme i resursi su glavne ulazne informacije planiranja, dok je tačnu definiciju svih koraka moguce i izostaviti. Precizno definisano vreme realizacije za projektni tim znači koncentrisanu timsku energiju, manje promena u planiranim poslovima, a samim tim i vecu produktivnost. Preduslov uspešnosti ove strategija je da klijent/investitor ima jasnu viziju svih rezultata projekta u finalnom obliku i da definiše sve svoje zahteve pre pocetka realizacije tih rezultata. Kod softverskih projekata ovaj uslov je nažalost gotovo nemoguće ostvariti, mada se veoma često nameće vremensko ograničenje

Na kraju, u primeni su i strategije plana koje kritične tacke u odlučivanju organizuju se tako da ih prate **planovi koji upravljaju rizicima**, a ne vremenom. To su najčešce planovi alternativnih scenarija a ne planovi aktivnosti. Od početka do kraja projekat se realizuje ciklično. U bilo kojoj fazi projekta revizija (i/ili promena) može da utice na prethodno realizovanu fazu. Ova strategija nam pomaže da se ne udubimo previše u redosled aktivnosti jer se fokusira na upravljanje rizicima i preduslova za ispostavljanje rezultata. Pomaže nam takode da baziramo planiranje kljucnih dogadaja na iskustvu više nego na predvidjanju.

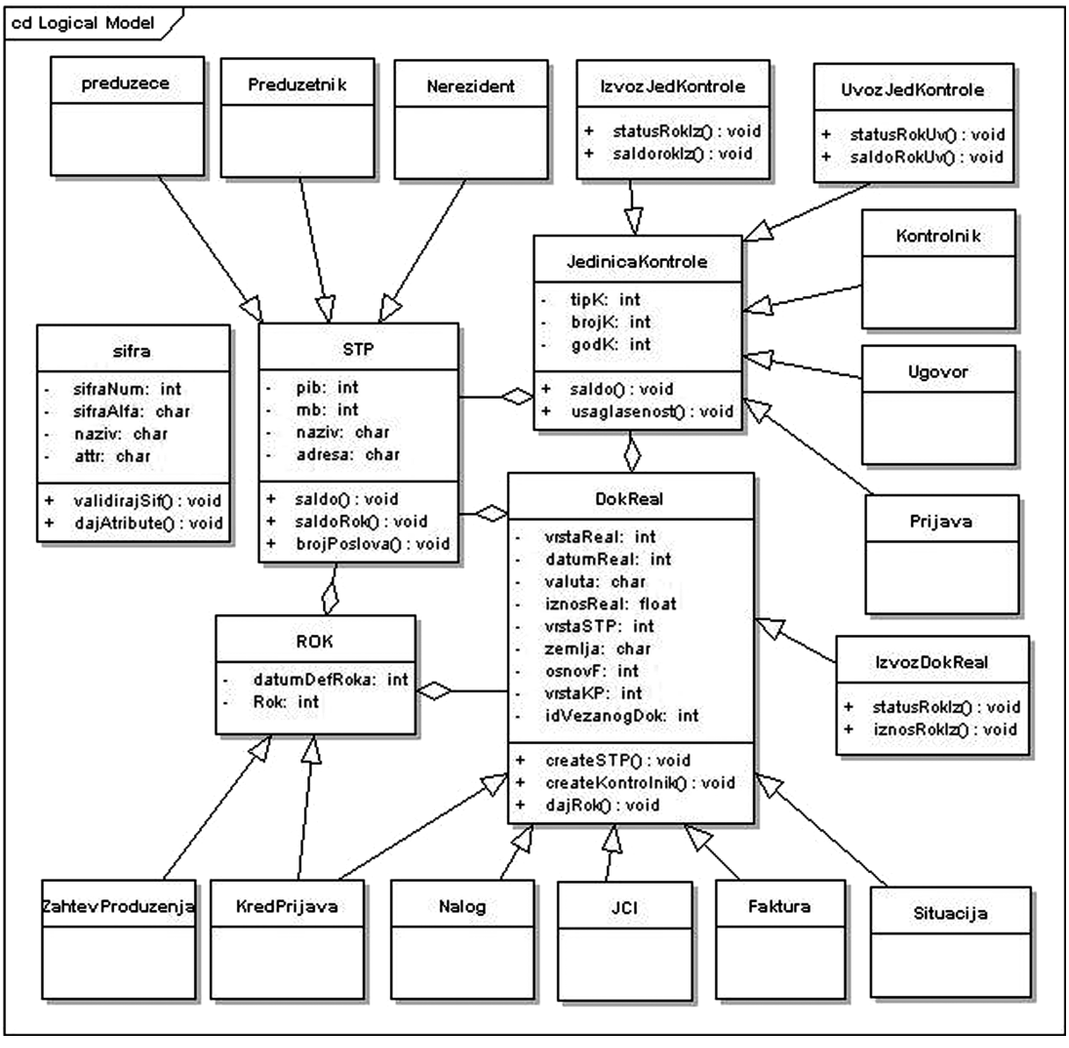
Glavno pitanje na početku izrade plana svakog projekta je - koju strategiju izabrati? Najbolje rešenje je najteže -birati strategiju prilagodjenu pojedinačnom projektu, a ne tražiti kalup koji bi se mogao usvojiti kao prihvatljivo rešenje za svaki projekat. U principu, linearna strategija mogla da bude od koristi kod dugackih projekata, kod onih gde je kvalitet i precizno dokumentovanje bitan faktor uspeha i gde je potrebna jednostavna, egzaktna kontrola projekta u svakoj fazi. Talasanje kao strategija je efikasno kod projekata koji se znatno menjaju i revidiraju i kada rukovodstvo ne želi previše da investira u detaljne planove, a želi da ima konstantan uvid u napredak projekta. Time boxing zahteva mogućnost razlaganja posla na manje segmente da bi se sklopila celina, održava energiju tima na visokom nivou i može da se koristi samo kada je finalni proizvod veoma precizno definisan i kada se raspolaže izuzetno kvalitetnim timom. Zasnivanje plana projekta na planu rizika kao slučajnih dogadaja može se primeniti u uslovima kada postoji veliko poverenje u sposobnost projektnog tima i kada je bitniji stvarni uspeh projekta nego rezultati zvaničnih supervizija od strane menadžmenta i njihovih revizora.

6. KARAKTERISTIčNIPROBLEMI KOD VELIKIH SISTEMA

Svrha svakog plana je da definiše okvir za vrednovanje posla tj. merljive parametre prema kojima se može proceniti i meriti napredak posla. Za velike, složene softverske projekte, da bi bili ocenjeni kao uspešni, moraju se u samoj fazi plana adekvatno razrešiti karakteristični problemi koji proističu iz same njihove veličine, odnosno složenosti.

Naravno da struktura posla dobrim delom proizilazi iz strukture i definicije izabranog modela samog procesa razvoja i strategije planiranja, ali samo u kombinaciji sa strukturom poslovnih zahteva nad definisanim područjem rada. U uslovima ekspanzije informacione tehnologije razvijena je i tendencija sveobuhvatnosti softverskih rešenja, bez razlike da li se ona odnose na jednostavne ili na složene i velike realne sisteme. Kako za velike sisteme nema mogućnosti da se u kratkom vremenu inicijalizacije projekta kompletan sistem dovoljno upozna, u specifikacijama se uglavnom oslanjamo na definicije budućih korisnika kao poznavalaca. Takve definicije su obično neprecizne, pa se često umesto definicije granice sistema pribegava definiciji namene i ciljeva budućeg projekta, podrazumevajući da se kontrola nad sistemom ostvaruje uzročno - posledičnim odnosno ciljnim usmeravanjem aktivnosti. činjenica je da unutar nekog velikog sistema različiti akteri imaju različite, čak suprotstavljene ciljne vrednosti. Takodje se ispostavi situacija da se ne može ustanoviti uzrok ili cilj neke aktivnosti a vrlo često i situacija da se npr. zna zašto je nešto počelo ali ne i čemu vodi, ili obrnuto, da se zna šta je cilj ali ne i šta je uzrok. Znači, kako su realni sistemi živi u smislu da su podložni promenama, sav trud oko analize i razvoja velikih integrisanih softverskih sistema obično rezultira time da se takav informacioni sistem, ako se realizuje, praktično postavlja iznad svojih kreatora i svojim korisnicima ograničava mogućnost drugačijeg shvatanja realnog sistema, a samim tim i kreativan doprinos u smislu njegovog razvoja. Jer ciljevi, odnosno namena celog velikog sistema ne moraju, niti mogu, biti jasni svakom akteru sistema, a sa druge strane upravo ta neodredjenost krije razvojne šanse odnosno mogućnost inovativnog doprinosa pojedinca. To je najčešći razlog zbog kojeg se žsveobuhvatno' projektovani softveri odbacuju kao neuspešni, a parcijalna, privremena rešenja se lakše i duže eksploatišu.

Zato je za uspešnost nekog softverskog projekta u okviru velikog i složenog realnog sistema, ključna odluka ona koja se odnosi na namenu, glavne ciljeve i delokrug projekta, tj. na granicu budućeg softverskog sistema. Ambiciozni projekat se ovim definicijama u startu mora strukturirati u smislu izbora onog sto je **glavno i prioritetno** jer je za takve projekte koji obično dugo traju, važno ostvariti početnu prednost, odnosno



Slika 1. – *Model sistema obrade različitih dokumenata*

dobro definisanim planom što pre obezbediti povoljnu ocenu uspešnosti. Takva ocena ne obezbedjuje apriori konačni uspeh, ali u startu deluje veoma podsticajno na dalji razvoj projekta. Precizne i dobro skrojene definicije moraju obezbediti efikasnu validaciju i otvorenost za eventualnu nadogradnju odnosno dopunu i izmenu sistema. Zapravo ovde se krije esencija veštine uspešnog planiranja - u delokrugu posla izdiferencirati centralne i marginalne delove. Sistem se primenom tehnika analize i dekompozicije strukturira na podsisteme, ali za izdvajanje delova koji su od ključne važnosti za celinu projekta potrebno je primeniti postupak generalizacije u meri u kojoj se ceo složeni sistem svodi na relativno jednostavan i jasan esencijalni zahtev, koji se detaljno analizira i kojem se daje prioritet u redosledu realizacije a mnoštvo pojedinačnih detalja se marginalizuje i ostavlja za naknadne specijalizacije i inkrementalnu nadogradnju u narednim iteracijama. Ovakav postupak zasniva se na objektnim modelima i metodama. čak i u slučaju da ovakav pristup nije prihvaćen od strane korisnika, veoma je korisno na startu napraviti bar jedan ovakav model sistema, da bi se za dalji posao detaljne analize i dizajna procenila prava težina i značaj pojedinih modula. Na slici 1 dat je jedan model sistema koji obuhvata pojedinačne obrade različitih dokumenata u svrhu njihovog klasifikovanja po preduzećima i jedinicama posla i uparivanja iznosa sa dokumenata prema odredjenim kriterijumima. Model apostrofira proces osnovne kontrole iznosa generalizujuci sve tipove dokumenata u jedan osnovni i sugerise da se taj deo realizuje prvim inkrementom a sve specifike obrada pojedinačnih dokumenata i izuzetaka u kontroli planiraju se kao sekundarne specijalizacije.

Kod ovakvog pristupa u razvoju velikih i složenih sistema nije glavni problem u definisanju tehnike analize nego problem smislene, postupne sinteze pojedinačno razvijenih i implementiranih delova. Takodje, ovakav pristup pretpostavlja kontinuitet u razvoju sistema pa je provera kompletnosti suvišna i dobro postavljen sistem se praktično uvek ostavlja otvorenim. Usudjujem se ovaj pristup, u odnosu na neki žklasičan', uporediti sa razvojem filozofske misli od racionalističkog determinizama do egzistencijalizma. Zapravo u praksi smo uvek i bili u situaciji da tražimo esenciju u već egzistirajućim delovima sistema i mogli bismo reći da smo, i u ovom poslu, ipak, "osudjeni na slobodu".

7. ZAKLJUČAK

Na kraju treba reći da na konačnu odluku o varijanti plana za konkretan softverski projekat ne utiču samo usko stručni kriterijumi, nego se moraju uvažiti i mnogi drugi, koji u realnom okruženju projekta mogu biti i jači i važniji. A efikasno planiranje podrazumeva prepoznavanje i pravilno rangiranje svih kriterijuma uspeha u celokupnom kontekstu projekta. Kako praksa dokumentovanog planiranja praktično daje šansu da se deo odgovornosti za eventualne propuste prebaci na korisnike, ili naručioce, danas se u odnosima naručilac-isporučilac projekta, ispostavlja kao najvažnija veština dobre komunikacije i formulacije ugovornih i planskih dokumenata. Zapravo, uspešan menadžer softverskih projekata ne može praviti specifikacije merila uspešnosti posla samo na osnovu poznavanja procesa razvoja, softverskih arhitektura i tehnologija nego mora imati mnogo šira konkretna znanja iz različitih oblasti (psihologije, sociologije, ekonomije...) ili bogato iskustvo u saradnji sa različitim korisnicima na različitim projektima, da bi pažljivo izbalansirao sve realne faktore i dobrom formulacijom plana obezbedio povoljnu ocenu uspešnosti projekta.

Literatura

[1] Projekat posredne Kontrole Deviznog Poslovanja (KDP) - Projektna dokumentacija, Beograd, 2005. i 2006.

[2] http://msdn.microsoft.com/vstudio/teamsystem/msf/, juni 2006.

[3] http://www.microsoft.com/technet/itsolutions/cits/mo/mof/default.mspx, juni 2006.

[4] http://www.codeproject.com/gen/design/optimize_develop.asp?print=true, juni 2006.

[5] http://www.phptr.com/articles/article.asp?p=27371&seqNum=3&rl=1, juni 2006.

[6] http://sunset.usc.edu/publications/dissertations/aaadef.pdf, juni 2006.

[7] http://sern.ucalgary.ca/courses/seng/621/W97/johnf/TOP#TOP, juni 2006.

[8] http://www.sei.cmu.edu/cmmi/, juni 2006.

[9] http://www-128.ibm.com/developerworks/rational/library/oct05/kroll/rate#rate, juni 2006.

[10] http://www.rm-odp.net/, juni 2006.

[11] Schulte,P. : Complex IT project management : 16 steps to success, Auerbach Publications, 2003.

[12] Cassidy A. Guggenberger K.: A practical guide to information systems process improvement, Auerbach Publications, 2000.

[13] Johnson B. Woolfolk W. Miller R. Johnson C. : Flexible Software Design Systems Development For Changing Requirements, Auerbach Publications, 2005.

[14] Rothman J. : Iterative Software Project Planning And Tracking, 1997.

[15] Si Alhir S. : The Agile Unified Process (AUP) , PMP, 2005.

[16] Ambler S.W. : A Manager's Introduction to The Rational Unified Process (RUP), Ambysoft, 2005.

Biljana Ribić, dipl. mat., savetnik za Informacione tehnologije, Narodna banka Srbije Beograd

email: biljana.ribic@nbs.yu

Oblasti interesovanja: biznis analiza i projektovanje softverskih sistema, project management,internet tehnologije i elektronsko poslovanje