MINIMIZIRANJE TROŠKOVA IZRADE SOFTVERA
SOFTWARE DEVELOPMENT COSTS MINIMIZATION

Stanišević Ilja, Marković Vesna, Petrović Đorđe
Visoka poslovna škola strukovnih studija VIPOS, Valjevo

Sadržaj – *Savremene metode projektovanja softvera, kako klasične tako i agilne, zahtevaju značajne resurse. Podrazumeva se visoko obučeno i stručno osoblje, upotreba naprednih softverskih alata, kao i raspoloživost značajnih finansijskih sredstava. Ovi zahtevi često nisu ispunjeni, posebno u društvima u razvoju, naročito tokom svetske ekonomske krize. Ovaj rad istražuje strategije za redukciju troškova u toku ciklusa životnog razvoja softvera, počevši od akvizicije korisničkih zahteva, pa do modifikacija i održavanja softverskog sistema.*

……

Abstract - *Both types of software development methodologies, classical as well as agile, require significant resources, i.e. highly educated man power, advanced software tools and significant funds are supposed. These requirements frequently are not met in developing countries, especially during the World Economic Crisis. This paper investigates costs reduction strategies in a software development lifecycle. Such a possibilities are investigated during each phase of the development, starting with the system requirements acquisition and ending with the system modification and maintenance.*

Ključne reči: *metode za projektovanje softvera, troškovi, upravljanje rizikom, agilne metode*
Keywords: *software development methodology, costs, risk management, agile methods*

1. UVOD

Rastuća potreba za izradom softvera za najraznovrsnije namene dovela je do pojave većeg broja raznovrsnih metoda za projektovanje i razvoj softvera. Većina ovih metoda je nastala u informaciono i ekonomski razvijenim sredinama, pa one pokazuju tendenciju da rešavaju probleme izrade softvera u tim sredinama. Tako su osnovni ciljevi ovih metoda efikasna kontrola nad odvijanjem procesa izrade softvera, standardizacija kvaliteta završnog proizvoda, minimiziranje rizika projekta, te efikasnost realizacije. Usled uslova koji postoje u društvima u kojim su nastale, ni jedna metoda kao primarni cilj ne postavlja minimiziranje zahteva za resursima. U njima se implicite pretpostavljla raspoloživost dovoljnog broja kadrova koji su dovoljno stručni za uloge koje u procesu razvoja softvera obavljaju. Pretpostavlja se i adekvatna hardverska i softverska platforma za razvoj. Ove pretpostavke su, konačno, dovele do toga da generisanje poslovnog softvera iziskuje značajna sredstva.

U manje razvijenim sredinama problemi sa kojima se projektanti i programeri susreću su drugačiji, pa je primena postojećih metoda često neadekvatna [1]. U ovim sredinama je primetno značajno odsustvo resursa, kako hardverskih i softverskih, tako i kadrovskih. Osnovni uzrok ovoj redukciji resursa je u nedovoljnim finansijskim sredstvima sa kojima se raspolaže za projekte realizacije softvera, budući da korisnici programskih sistema znatno manje sredstava mogu odvojiti za informatizaciju svog poslovanja [2]. Stoga je značajno ispitati kako se izbor metode za projektovanje softvera reflektuje na ukupne troškove, kao i koliko su određene metode primenljive u uslovima značajno smanjenih resursa, pre svega finansijskih. Takođe je značajno ispitati mehanizme i strategije koje mogu umanjiti zahteve razvoja softvera za resursima, te tako umanjiti krajnju cenu završnog proizvoda, a da to ne dovede do redukcije kvaliteta u neprihvatljivoj meri.

Projekti realizacije softverskih sistema u uslovima nedostajućih resursa su, po pravilu, mali projekti, u kojima se angažuju skromna finansijska sredstva i koje realizuje mali broj ljudi. Neadekvatnost postojećih metoda za ovakve uslove nosi u sebi opasnost da se projekat pretvori u realizaciju po „kodiraj i ispravljaj“ (*code-fix*) principu, što dovodi do nepouzdanog i nestandardnog završnog proizvoda. Usled toga je potrebno sistematizovati one osobine i strategije savremenih metoda izrade softvera koje su primenljive u ovakvim uslovima.

2. KARAKTERISTIKE NISKOBUDŽETNOG RAZVOJA SOFTVERA

Značajna redukcija finansijskih resursa lančano izaziva redukciju i ostalih resursa. Osnovne tipove ograničenja možemo grupisati u kadrovska, hardverska i softverska ograničenja.

Kadrovska ograničenja izazvana nedostatkom kako brojnosti tako i obučenosti kadrova uslovljavaju da su projektni timovi veoma mali. Angažovano osoblje stoga mora obavljati više funkcija u okviru tima. Članovi tima su, takođe, prinuđeni da se angažuju na više različitih projekata koji su u različitim fazama životnog ciklusa istovremeno. U ovakvim uslovima je otežana bilo kakva specijalizacija kadrova, a nužan je i angažman neformalno i slabije obučenih informatičara. Može se očekivati i česta fluktuacija kadrova.

Hardverska ograničenja uslovljavaju da se informatička oprema nabavlja ne planski, već od prilike do prilike, kako uslovi dozvoljavaju. Stoga imamo situaciju da je hardver kojeg koristi kupac softverskog sistema izrazito neujednačen. Hardver je dugo u eksploataciji, te je samim tim verovatnoća otkaza veća.

Softverska ograničenja se ogledaju u tome da se minimalna sredstva investiraju u nabavku softvera, koji se takođe koristi značajno duže nego u razvijenijim sredinama. Ovo za posledicu ima raširenost besplatnog i piratskog softvera. Podrška za instalirane programe je stoga ili veoma oskudna ili nepostojeća.

Ovako detektovana ograničenja izazivaju efekte na razvoj softvera koji su drugačiji od onih u razvijenijim i bogatijim sredinama.

Angažovani kadrovi su ekstremno opterećeni. Oni obavljaju više uloga u timu i rade na više projekata istovremeno, što se negativno odražava na performanse njihovog rada, te je neophodno iskoristiti svaku mogućnost za njihovo rasterećenje.

Vitalno važna osobina je prilagođenost softvera stvarnim informatičkim potrebama korisnika. Od razvijanog softvera se ne zahteva da podrži celokupno poslovanje firme, već samo osnovne i kritične delatnosti. Često se sistem mora razvijati u skokovima, modularno, u zavisnosti od raspoloživih sredstava. Usled ovoga je poželjno da svaki modul sistema može biti u eksploataciji nezavisno od (ne)postojanja ostalih delova sistema.

Budući da su na raspolaganju skromna finansijska sredstva i rizik razvoja sistema je mali. Takođe su praćenje plana i vreme izrade sistema od sekundarne važnosti. Ovo su osobine kojima se u svakoj postojećoj metodi razvoja softvera pridaje velika važnost.

Komunikacija u timu je značajno olakšana usled njegove malobrojnosti. Najčešće nema stvarne potrebe ni za kakvom formalnom komunikacijom, bilo u pisanom obliku, bilo korišćenjem nekog softverskog alata za ovu namenu.

U ovako restriktivnim uslovima se menja priroda dokumentacije. Umesto da služi za izradu studije izvodljivosti, praćenje i kontrolu procesa razvoja sistema, te za kontrolu rizika projekta, osnovna funkcija dokumentacije postaje dvojaka. Prvi aspekt je određivanje obima i grubog termininiranja projekta, pri čemu je značajnije određivanje redosleda realizacije u zavisnosti od tehničkih zahteva i potreba naručioca, nego precizno određivanje vremena početka i završetka svake faze. Drugi aspekt dokumentacije je da olakša modifikaciju i održavanje sistema i omogući efikasno uključivanje novih kadrova u projekat.

3. PRIMENLJIVOST KLASIČNIH METODA PROJEKTOVANJA SOFTVERA U USLOVIMA OGRANIČENIH RESURSA

Klasične metode projektovanja teže da budu opšte primenljive, što dovodi do toga da postaju preterano formalne i obimne. Metode koje se baziraju na praćenju procesa, kao što su kaskadna metoda [3], Behmova spiralna metoda [4] i Microsoftov okvir za rešenja (*Microsoft Solution Framework – MSF*) definišu veliki broj uloga u timu, što povećava broj angažovanih kadrova. Izlaz i ulaz svake faze predstavlja obimna dokumentacija čija je osnovna namena praćenje projekta i kontrola rizika, pa se značajni resursi troše na generisanje ove dokumentacije. U velikim timovima koje pretpostavljaju ove metode, i sama komunikacija između članova ili delova tima je problem, pa se često koriste posebni softverski alati za ovu namenu. Sve ovo utiče na povećanje cene završnog proizvoda, softvera.

Metode bazirane na prototipskom principu, kao što su metode za brzi razvoj softvera (*Rapid Aplication Development – RAD*) imaju značajno manje zahteva za angažovanim kadrovima, ali ti kadrovi moraju biti visoko obučeni, dakle skupi. One, takođe, zahtevaju i upotrebu posebnih softverskih alata za informacionu analizu i automatsko generisanje koda. Cene ovakvih alata mogu višestruko da nadmaše vrednost celokupnog projekta. U uslovima malog i siromašnog tržišta, broj potencijalnih korisnika čini njihovu nabavku neisplativom. Problem kod ovih metoda u uslovima značajno limitiranih resursa predstavlja i činjenica da često visoki hardverski zahtevi nisu određeni samim programskim sistemom, već softverskim alatom korišćenim za njegovu realizaciju.

Može se zaključiti da klasične metode projektovanja softvera značajne resurse troše na kontrolu procesa, kontrolu rizika projekta, formalnu komunikaciju (kako između članova tima, tako i između tima i korisnika), hardver za implementaciju sistema i razvojne softverske alate.

4. PRIMENLJIVOST AGILNIH METODA PROJEKTOVANJA SOFTVERA U USLOVIMA OGRANIČENIH RESURSA

Agilne metode projektovanja su se pojavile početkom XXI veka i baziraju se na četiri agilne principa:

- pojedinci i interakcije su važniji od procesa i alata;
- softver koji radi je važniji od sveobuhvatne dokumentacije;
- saradnja sa korisnikom je važnija od pregovaranja i ugovaranja;
- odgovor na promene je važniji od praćenja plana [5].

Iz ovih principa se razvio celi spektar novih, značajno fleksibilnijih metoda. Najpoznatije agilne metode su ekstremno programiranje (*Extreme Programming – XP*) [6], skram (*Scrum*) [7] i metodologija za dinamički razvoj softvera (*Dynamic Software Development Methodology – DSDM*). U osnovi ovih metoda je tretiranje kreiranja softvera kao kreativnog čina, a ne kao industrijskog proizvoda. Agilne metode vraćaju u fokus procesa projektovanja i izrade softvera čoveka i njegovu kreativnost, a ne primenjivani alat ili proceduru.

Agilne metode projektovanje su, usled svoje fleksibilnosti, daleko primenljivije u uslovima ograničenih resursa [1]. Ovo se pre svega odnosi na aspekte organizacije koja se zasniva na malim, multifunkcionalnim timovima,

neformalne i efikasne komunikacije, permanentne integracije sistema u radno okruženje kupca, svakodnevne raspoloživosti korisnika, te insistiranja na jednostvnosti sistema putem YAGNI (*You Ain't Gonna Need It* – "to Vam neće trebati") principa.

Ipak, neki aspekti agilnih metoda nisu primenljivi u ovim uslovima. Najčešće napadana osobina agilnih metoda, posebno u slučaju ekstremnog programiranja, predstavlja programiranje u paru gde dva programera dele jedan računar, tastaturu i monitor. Istraživanja su pokazala da na ovaj način u približno istoj meri rastu i troškovi i kvalitet generisanog softvera [8], tj. smanjuje se broj grešaka u programiranju, te samim tim vreme i resursi potrebni za njihovu korekciju. Ipak, u uslovima značajno ograničenih kadrovskih resursa, ovaj aspekt metoda nije primenljiv, jer zahteva angažman većeg broja osoblja. S druge strane, vreme realizacije u ovakvim uslovima nije od vitalnog značaja, budući da se poslovi u velikoj većini ugovaraju unapred po fiksnoj ceni, nezavisno od stvarnog vremena realizacije i uloženog rada, koji je poznat tek po okončanju projekta. Jasno je da u ovim okolnostima nije prednost brža realizacija sistema, ako ona iziskuje veći broj angažovanog osoblja.

Karakteristika agilnih metoda je da se značajan naglasak stavlja na testiranje koje se sastoji od jediničnih i funkcionalnih testova. Ovi testovi treba da budu automatski, uz korišćenje adekvatnih alata, i da se pišu pre pisanja samog koda. Ovim se značajni resursi troše na proces testiranja, a ostvareni rezultati ne opravdavaju uvek taj trošak, pogotovo u jednostavnijim i rutinski realizovanim delovima sistema.

Posvećenost osoblja tekućem projektu, kao i insistiranje na 40-časovnoj radnoj nedelji, koji su sastvani deo svake agilne metode, su teško ostvarivi u uslovima hroničnog nedostatka kadrova.

Najveća mana agilnih metoda je neprikladnost dokumentacije. Težeći da što pre dođu do funkcionalno prihvatljivog softvera, ove metode u velikoj meri zanemaruju dokumentaciju, tako da ona ili ne postoji, ili se ogleda u skupu korisničkih priča koji daju tek grubi okvir projekta.

U poslednje vreme dolazi do približavanja klasičnih i agilnih metoda koje se ogleda u većem broju softverskih alata namenjenih razvoju programa bilo primenom neke od agilnih metoda, bilo modifikacijom neke od klasičnih metoda u skladu sa agilnim principima (pa na tržištu imamo MSF za agilni razvoj softvera i agilni RUP). Ovo približavanje će nužno voditi poskupljenju softvera realizovanog ovakvim pristupom uzrokovanim cenom novih alata i cenom obuke kadrova za njihovu primenu.

5. STRATEGIJE SMANJENJA TROŠKOVA IZRADE SOFTVERSKIH SISTEMA

Usled malog rizika, tj. skromnih uloženih sredstava u razvoj, sve aktivnosti kojima je cilj puka evidencija i praćenje projekta, te upravljanje i analiza rizika mogu biti potpuno izostavljene, budući da resursi potrebni za njih mogu premašivati cenu celokupnog projekta.

Postupak izrade plana projekta treba da bude minimiziran i da pokriva samo kratki opis potrebnih funkcionalnosti sistema i redosled njihove realizacije. Kod ovako malih projekata nema potrebe za pravljenjem studije izvodljivosti i detaljnog plana projekta.

Uslovima redukovanih resursa optimalno pogoduje organizacija projektnog tima u plitkoj hijerarhiji koja se sastoji od mastera i članova tima. Uloge nisu specijalizovane, već svi mogu raditi sve. Uloga mastera tima je koordinacija ostalih članova i rukovođenje odvijanjem projekta. U ovakvoj organizaciji tima nema potrebe za bilo kakvom formalizacijom i praćenjem komunikacijem. Neformalna verbalna komunikacija i običan e-mail potpuno zadovoljavaju potrebe ovakvog tima.

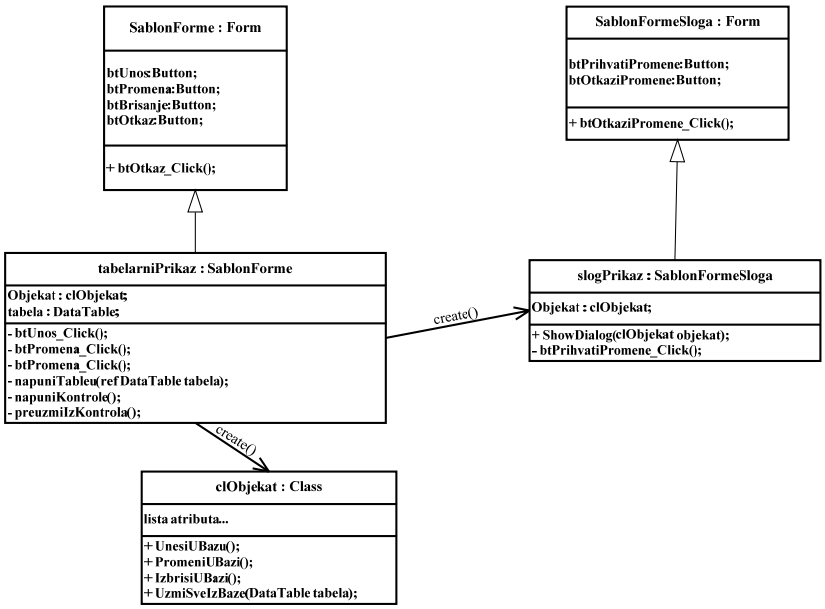
Dizajn sistema ne bi trebao da postavlja nikakve dodatne hardverske zahteve, već bi se trebao osloniti na postojeći hardver korisnika u što je to moguće većoj meri, čak i ako to podrazumeva korišćenje nešto starije tehnologije. Sistem se treba sastojati od međusobno što manje zavisnih modula koji bi omogućili faznu implementaciju.

Mogućnost redukcije troškova i smanjenja opterećenja angažovanih kadrova je i u aktivnom uključivanju krajnjih korisnika u ceo proces. Kod klasičnih metoda, korisnik je uključen na početku razvoja, kada daje zahteve koje sistem treba da ispuni, i na kraju, kada prihvata gotov proizvod. Korisnička uloga je značajnija kod agilnih metoda usled čestih isporuka manjih delova programa koje korisnik permanentno evaluira. Korisnik se može u značajnijoj meri angažovati i u testiranju sistema. Umesto da se troše resursi na pisanje automatizovanih testova, krajnji korisnik u okviru svog posla može vršiti i jedinično i funkcionalno testiranje svakog isporučenog dela sistema. Ovako definisana korisnička uloga, gde on nije puki naručilac, već deo tima, značajno pomaže i u implementaciji sistema. Korisnik sistem počinje doživljavati kao svoj, što eliminiše uobičajene otpore prilikom nužnih promena koje implementacija novog softverskog sistema iziskuje.

Posebno veliki potrošač resursa je postupak generisanja dokumentacije projekta [9]. Upotrebljivost standardno generisane dokumentacije je često upitna. U istraživanju sprovedenom pri Visokoj poslovnoj školi strukovnih studija VIPOS u Valjevu, čak 90% od ispitane 74 radne organizacije ili nemaju ili nikad ne koriste dokumentaciju svojih programa. U skladu sa svojom izmenjenom funkcijom, dokumentacija ne treba da bude detaljna i sveobuhvatna, već treba da obezbedi jezgrovit i efikasan pregled programa. Jedno moguće rešenje je intenzivna upotreba UML dijagrama koji na jednostavan način predstavljaju potrebne informacije [10]. Za komunikaciju sa korisnikom i određivanje potrebnih funkcionalnosti sistema su se veoma pogodnim pokazali dijagrami slučajevi korišćenja. Za komunikaciju i evidenciju unutar projektnog tima pogodni su dijagrami klasa i dijagrami baze podataka. Ovi dijagrami se veoma lako i brzo mogu generisati, a na tržištu je i veći broj paketa za njihovo generisanje koji se mogu slobodno koristiti, te ne iziskuju dodatna sredstva, a njihova upotreba ne zahteva posebno obučene i specijalizovane kadrove.

U situacijama kada je to, radi lakšeg razumevanja primenjenog rešenja, potrebno, određeni specifikumi programa se mogu definisati i komentarima u samom kodu. Korisna praksa je da se uz svaku netrivijalnu klasu i metodu u kodu priključi, u vidu standardizovanih komentara, informacija o ulazu, izlazu, uslovima za odvijanje i o posledicama koje metoda izaziva. Ovakvi komentari u značajnoj meri olakšavaju održavanje i modifikaciju sistema, te omogućavaju efikasnije uključivanje novih kadrova u projekat.

Značajne uštede se mogu ostvariti i uključivanjem slabije i neformalno obučenih kadrova. Ovo je moguće ako se u realizaciji sistema koriste kodni obrasci. Za razliku od projektnih obrazaca koji pružaju strukturu, ali ne i implementaciju rešenja, kodni obrasci u sebi uključuju i implementacioni kod u meri u kojoj je to moguće. Ovakve šablone mogu koristiti i osobe sa minimalnim poznavanjem programiranja. Pored ovoga, kodni obrasci omogućavaju i standardizaciju generisanog koda, te njegovu brzu realizaciju. Kao primer kodnog obrasca, UML dijagram klasa za tabelarni prikaz i ažuriranje podataka dat je na slici 1.



sl.1: UML dijagram klasa kodnog obrasca za tabelarni prikaz i ažuriranje podataka.

Pri primeni prikazanog šablona, sav programerski deo posla se sastoji od definisanja stringova koji sadrže adekvatne SQL upite prilikom implementacije metoda UnesiUBazu, PromeniUBazi, IzbrisiUBazi i UzmiSveIzBaze klase clObjekat i prilagođevanja korisničkog interfejsa u okviru klase tabelarniPrikaz povezivanjem grafičkih kontrola sa konkretnim podacima instance klase clObjekat.

6. ZAKLJUČAK

U ovom radu su ispitane i predložene neke strategije za redukciju potrebe za resursima pri projektovanju i realizaciji softverskih sistema. Potreba za niskobudžetskim, a ipak funkcionalnim i pouzdanim poslovnim softverom je evidentna, pogotovo u društvima u razvoju. Od velike koristi bi bio razvoj jedinstvene i sistematične metode za projektovanje softvera koja bi u sebi objedinjavala sve mogućnosti za umanjivanje troškova projekata izrade softvera. Postojanje takve metode bi zasigurno olakšalo i ubrzalo informatizaciju manje razvijenih i manje bogatih sredina.

LITERATURA

[1] Stanišević I., Pavlović V., Petrović Đ.„Komparativna analiza metoda projektovanja IS u domaćim uslovima*", Proc. 52nd ETRAN Conference, RT 4.3 1-4,* Palić, June 08-12, 2008.

[2] Stanišević I., „Karakteristike informatičkih uslova u našem okruženju", *INFOTEH-JAHORINA Vol. 7, Ref. E-II-3, p. 424-428*, March 2008.

[3] Royce W.W., "Managing the development of large software systems: Concepts and techniques", *Proc. of WESCON, VOL XIV, pg. A1-A9*, August 1970.

[4] Barry W. Boehm, „A Spiral Model of Software Development and Enhancement", *IEEE Computer, vol. 21, no. 5, pp. 61-72, doi:10.1109/2,* May 1988.

[5] Beck K. et al., "Manifesto for Agile Software Development", [on-line], http://agilemanifesto.org/

[6] Beck K, *Extreme Programming Explained: Embrace Change,* Reading, MA, Addison-Wesley, 2000.

[7] Schwaber K., Bedlee M., *Agile Software Development with Scrum*, Upper Saddle River, NJ, Prentice-Hall, 2002.

[8] A. Cockburn, L. Williams, The Costs and Benefits of Pair Programming, M.Marchesi, Succi G. *Extreme Programming Examined.* Eds. Boston MA: : Addison-Wesley, 2001.

[9] Stanišević I., Marković V., Petrović Đ., Pavlović V., „Low Budget Software Development", *UNITECH, ISSN 1313-230X,* Gabrovo, Bulgaria 2009.

[10] Booch G., Rumbaugh J., Jacobson, I., *The Unified Modeling Language - User Guide*, Boston, MA : Adison Wesley Longman Inc., 1999