UDC: 004.7:004.4 INFO M: str. 39-42

PRIMENA AGILNIH METODOLOGIJA RAZVOJA NA SPREDŠIT APLIKACIJE
APPLYING AGILE DEVELOPMENT METHODOLOGIES ON SPREADSHEET APPLICATIONS

Tamara Valok Radulović

REZIME: U ovom radu analizirana su dosadašnja istraživanja na temu metodologija razvoja spredšit aplikacija. Ova istraživanja ukazuju na to da još uvek ne postoji metodologija razvoja specijalno kreirana za ovaj tip aplikacija, što može biti jedan od uzroka uočenih ponavljanja sličnih grešaka u njihovim implementacijama. Kako bi se prevazišli pomenuti problemi, neki autori kao ideju predlažu primenu postojećih metodologija razvoja informacionih sistema na spredšit aplikacije. Nasuprot njima, u ovom radu predložena je ideja o primeni trenutno najpopularnijih netradicionalnih agilnih metodologija razvoja softvera, Ekstremno programiranje i Skram, na spredšit aplikacije. U radu je dat prikaz ovih metodologija, kako bi se uočila njihova primenljivost na spredšit aplikacije.

KLJUČNE REČI: Spredšit aplikacije, agilne metodologije, ekstremno programiranje, skram

ABSTRACT: This paper analyzes recent researches on the subject of spreadsheet development methodologies which indicate there is still no methodology specifically designed for this type of application. For that reason, in various applications similar mistakes are often repeating. Some authors have given the idea of applying existing development methodologies for information systems on spreadsheets in order to overcome the identified problems. As opposed to them, this paper presents the idea of applying currently the most popular non-traditional agile software development methodologies, Extreme programming and Scrum, on spreadsheet applications. The paper gives the overviews of these methodologies in order to realize their applicability on spreadsheet applications.

KEY WORDS: Spreadsheets, Agile methodologies, Extreme Programming, Scrum

1. UVOD

U današnje vreme, u preduzećima je prisutan trend porasta zastupljenosti integrisanih poslovnih rešenja (eng. Enterprise Resource Planning, skraćeno. ERP sistemi) kao krucijalnih stubova svakog poslovanja. Kompleksnost ovakvih sistema, njihovo uvođenje i održavanje, čini ih dosta skupim i dostupnim uglavnom većim ili bogatijim preduzećima. Za ostala, manja i srednja preduzeća, postoje alternativna rešenja, a neka od njih su spredšit aplikacije. U poređenju sa ERP sistemima, spredšit aplikacije su manje obimne, finansijski dostupnije, manje kompleksne, na osnovu čega se zaključuje da je njihov razvoj, uvođenje i održavanje jednostavnije i brže, a samim tim i jeftinije. Iako su spredšit aplikacije veoma rasprostranjene u poslovnom svetu, još uvek postoje problemi vezani za česte greške koje se u njima javljaju. Mnoga istraživanja zasnovana su upravo na ovom problemu, a neka od njih analizirana su u ovom radu. Smatra se da problem vezan za ove greške leži u činjenici da ne postoji metodologija razvoja specijalno kreirana za ovaj tip aplikacija. U svojim radovima, neki autori su pokazali kako tradicionalne metodologije za razvoj ERP sistema mogu da se primene na spredšit aplikacije. Ovaj rad prikazuje ideju o primeni netradicionalnih agilnih metodologija razvoja na spredšit aplikacije, koje bi mogle da spreče nastanak njihovih uobičajenih i čestih grešaka.

Rad se sastoji iz sledećih poglavlja: prvo, dat je uvod u spredšit aplikacije, a zatim prikaz prezentativnih radova iz ove oblasti. Dalje, dat je kratak prikaz agilnih metodologija razvoja. Na osnovu ovakve strukture rada, data je diskusija o primeni najpopularnijih agilnih metodologija razvoja na spredšit aplikacije i na kraju zaključak autora.

2. SPREDŠIT APLIKACIJE

Spredšit aplikacije su veoma rasprostranjene s obzirom na to da se koriste u malim i srednjim preduzećima koji čine veliki deo tržišta. Relevantni poslovni podaci perzistiraju posredstvom ovih aplikacija, a manipulacija njima vrši se zahvaljujući ugrađenim funkcijama koje omogućavaju izlaz u vidu značajnih informacija. Ove informacije koriste se u svrhu analize, planiranja i donošenja poslovnih odluka. Zahvaljujući pomenutim funkcijama, korisnici mogu da posmatraju i testiraju različite moguće scenarije kompleksnih poslovnih problema.

Uprkos svojoj velikoj rasprostranjenosti, greške u spredšit aplikacijama su veoma česte. One se smatraju preovlađujućim u spredšit aplikacijama, a postoje slučajevi kada su greške koštale preduzeća milione dolara (Powell, Lawson, & Baker, 2007). Neki autori (Knight, Chadwick, & Rajalingham, 2000) pokušavaju da objasne ovu situaciju kritikujući loš ili čak nepostojeći vid obuke programera za neku od metodologija razvoja. Takođe, isti autori u svom drugom istraživanju (Rajalingham, Chadwick, & Knight, 2001) razvili su klasifikaciju spredšit grešaka. Ovim istraživanjem oni pokušavaju da naglase koliko je važno razumeti karakteristike grešaka, kao i njihov nastanak. Kritične tačke prezentovane u njihovom istraživanju mogle bi se poboljšati primenom adekvatne metodologije za razvoj spredšit aplikacija.

Kako se poslovanje menja tokom vremena, menja se i njegov model. Spredšit aplikacije dozvoljavaju česte promene poslovnog modela, pa se upravo iz tog razloga smatraju veoma primenljivim. Spredšit inženjerstvo prilagođava učenje softverskog inženjeringa spredšit aplikacijama, obezbeđujući osam principa kao preporuke programiranju spredšita (Grossman, 2002). U sekciji 4, dato je poređenje ovih osam principa sa principima specifičnih metodologija razvoja. U ovom radu, ove specifične metodologije se odnose na agilne metodologije razvoja.

3. PREGLED OBLASTI

Mnogi radovi napisani su na temu grešaka u spredšit aplikacijama i autori pokušavaju da pronađu njihov uzrok. Greške se povezuju sa neshvaćenom i nepotpunom pokrivenošću poslovnog problema, lošom i nepotpunom dokumentacijom, nestandardnim dizajnom aplikacija, nedefinisanim kanalima ko-

munikacije između programera spredšit aplikacija i krajnjih korisnika i slično, što sve ukazuje na slabu organizaciju rada i nepostojanje adekvatne metodologije razvoja ovih aplikacija.

U svom radu, (Grossman, 2002) Grossman piše da se dobar softver ne razvija spontano, nego on mora biti projektovan. On tvrdi da su pomenutih osam principa softverskog inženjeringa takođe primenljivi i na spredšit aplikacije, barem kao polazna tačka, iako ovo još uvek nije potvrđeno.

Postoji zanimljivo istraživanje (Powell, Lawson, & Baker, 2007) vezano za klasifikaciju spredšit grešaka, gde autori pronalaze različite uzroke grešaka, kao što su: nedostatak domenskog znanja, nelogičan dizajn, ograničena ili nepostojeća dokumentacija, i slično.

Takođe, postoji istraživanje vezano za tradicionalne metodologije razvoja informacionih sistema i ideju da se one primene na spredšit aplikacije (Išljamović, 2012). Autor ne pokušava da uvede nove principe razvoja, nego da primeni postojeće, koji su se dokazali kao dobra praksa. Posmatrane su dve popularne metodologije: vodopad i spiralna metodologija. Zaključak autora je da metodologija vodopad može da se iskoristi barem kao polazna osnova za kreiranje dobre dokumentacije. Kada je reč o spiralnoj metodologiji, autor je predstavlja kao metodologiju sa velikim potencijalom zbog svojih karakteristika inkrementalnosti i stalnog napredovanja, razvoja i implementacije novih funkcija.

Sa druge strane, Grossman smatra da fleksibilni modeli životnog ciklusa imaju više prednosti u poređenju sa manje fleksibilnim razvojnim metodologijama, kao što je tradicionalni vodopad.

4. AGILNE METODOLOGIJE RAZVOJA SOFTVERA

U uslovima dinamičnog poslovnog okruženja, stabilnost je cilj kojem sva preduzeća teže. Ono što je potrebno da bi se taj cilj postigao je fleksibilnost.

Prema autoru Highsmith (Highsmith J. A., 2004), agilnost predstavlja sposobnost da se odgovori na promene, kako bi se postigao profit u turbulentnom poslovnom okruženju. Sve češće promene koje predstavljaju zahteve okruženja, pogodile su sve industrije. Kao rezultat ovih zahteva, da bi se redukovali troškovi, a povećao profit, uvedene su nove metodologije za upravljanje IT projektima, takozvane agilne metodologije. Ove metodologije prvenstveno su namenjene razvoju softvera, ali sa nekim manjim izmenama mogu se smatrati metodologijama za upravljanje projektima.

Ideja ovog rada je da se prikaže kako agilne metodologije mogu da se primene na razvoj spredšit aplikacija. Ovu ideju je takođe predstavio Panko. U svom radu (Panko, 2007) on kaže da postoji mnogo metodologija za razvoj softvera, posebno novih agilnih, i da razvoj spredšita može da se obavi na netradicionalni način, posebno agilnim metodama.

Agilne metodologije su bazirane na Agilnom manifestu (Fowler & Highsmith, 2001) koji obuhvata dvanaest principa agilnosti, koji propisuju vrednosti i principe koje metodologije moraju prihvatiti kako bi se smatrale agilnim. Imajući u vidu već pomenutih osam principa softverskog inženjerstva (Grossman, 2002) i dvanaest principa agilnih metodologija (Fowler & Highsmith, 2001) moguće je izvršiti njihovu komparaciju, kako bi se uvidelo koji agilni principi su u skladu sa principima softverskog inženjerstva, a koji nisu. Principi softverskog inženjerstva i agilnog razvoja nisu dati u ovom radu, nego su čitaoci upućeni na odgovarajuću literaturu.

Principi softverskog inženjerstva i agilnog razvoja teže da uvedu najbolje prakse prilagođene specifičnim potrebama softvera. Međutim, agilni principi nalažu da se izabrane prakse ponovo razmatraju u redovnim vremenskim intervalima i eventualno prilagode.

A priori zahtevima se pridaje velika pažnja u praksi softverskog inženjeringa, dok su za agilne principe na početku potrebni samo oni zahtevi relevantni za prvu radnu verziju softvera. Kasnije se ovi zahtevi proširuju do detalja i dodaju se novi, jer agilni principi podržavaju česte isporuke radnih verzija softvera.

Za razvoj spredšit aplikacija veoma je važno da agilni principi podržavaju održiv razvoj aplikacija, jer je upravo jedna od karakteristika spredšita česta promena zahteva.

Ono što je podjednako važno za softversko inženjerstvo i agilni razvoj je razvojni tim koji zahtevaju kontinuiranu pažnju. U okviru agilnih principa razvojni tim treba da bude samoorganizovan i sarađuje sa klijentima svakodnevno, kako bi se moguće greške svele na minimum.

5. PRIMENA EKSTREMNOG PROGRAMIRANJA I SKRAMA NA SPREDŠIT APLIKACIJE

Kako su najpopularnije agilne metodologije ekstremno programiranje i skram, one će biti posmatrane u smislu njihove primene na razvoj spredšit aplikacija.

5.1. *Ekstremno programiranje*

Ekstremno programiranje predstavlja agilnu metodologiju osmišljenu za male i srednje timove programera sastavljenih od 6 do 20 ljudi. Ova metodologija je verovatno najpopularnija agilna metodologija zbog svoje fleksibilnosti da odgovori na promene u softverskim zahtevima. Ona podržava menjanje životnog ciklusa razvoja softvera i uklanjanje grešaka u ranim fazama razvoja. Takođe, u ekstremnom programiranju izbegava se planiranje unapred i promovišu se brze iteracije koje daju klijentu neposredne rezultate. Ovo znači da se ekstremno programiranje koncentriše na trenutne potrebe klijenta, kreirajući najjednostavniju verziju sistema koja radi. Na ovaj način, kreirani sistem ima manje tačaka integracije i lakše se prilagođava novim zahtevima. Osnovna vrednost ove metodologije je komunikacija. Fokus je na oralnoj komunikaciji, a ne na dokumentima, izveštajima ili planovima. Ekstremno programiranje je bazirano na sinergiji uloga u timu. Ove uloge nisu fiksirane, pa tako jedna osoba u jednom trenutku vremena može imati ulogu programera, a u drugom može biti osoba koja upravlja razvojem. U svakom slučaju, izdvajaju se dve grupe u timu: prva se sastoji od klijenta i menadžera, a druga predstavlja tehnički deo tima koji čine programeri, testeri, kontrolori, treneri itd. Ekstremno programiranje sastoji se iz pet faza: istraživanje, planiranje, iteracije, produkcija i kontinualno održavanje.

5.1.1. Faze ekstremnog programiranja

U fazi istraživanja, programeri definišu korisničke zahteve iz korisničkih priča koje se sastoje od 3-4 rečenice napisane u netehničkom jeziku. One odražavaju korisničke potrebe, a programeri procenjuju vreme i metode potrebne za njihovu implementaciju. U ovoj fazi, vizija i misija sistema postaje jasna.

Planiranje je zapravo donošenje odluka o implementaciji korisničkih priča u svakoj isporuci sistema. U ovoj fazi programeri ukazuju korisnicima na moguće rizike. Kada su sve is-

poruke isplanirane, one se dele na iteracije. Svaka iteracija treba da isporuči specifičnu vrednost, a svaka naredna iteracija da poveća prethodnu vrednost.

Tokom faze iteracija počinje najveći rad na sistemu. Prvo se uzimaju u obzir najrelevantnije priče. Priče mogu biti podeljene u programerske zadatke, pa programeri mogu da se prijavljuju za njih i daju procenu o potrebnom vremenu za njihovu implementaciju. Programiranje počinje sa pisanjem testova i nastavlja se sa programiranjem u parovima. Tokom iteracija programeri vrše integraciju sistema.

Produkcija, odnosno puštanje sistema u rad, može da se obavi na nekoliko načina u zavisnosti od potreba preduzeća i samog sistema: veliki prasak (big bang), fazni pristup, paralelni pristup.

Faza kontinuiranog održavanja započinje čim se faza produkcije završi. Definisanjem automatizovanih platformi za testiranje sistema omogućene su modifikacije sistema kada se za to ukaže potreba.

5.1.2. Primena ekstremnog programiranja na razvoj spredšit aplikacija

Imajući u vidu karakteristike ekstremnog programiranja, može se zaključiti da se ono može uspešno primeniti na razvoj spredšit aplikacija, posebno zbog mogućnosti prilagođavanja čestim promenama zahteva aplikacije. Takođe, naglasak na brze iteracije može umnogome pomoći razvoju spredšita, jer se eventualne greške mogu otkriti u ranim verzijama aplikacije.

Tokom faze istraživanja, krajnji korisnici aplikacije zaduženi su za pisanje korisničkih priča. Za njih to ne bi trebalo da predstavlja problem, jer nisu programeri što im olakšava opisivanje svojih zahteva. Iz ovih priča moguće je kreirati dokumentaciju koja se na taj način stvara paralelno sa razvojem. Korisnički zahtevi predstavljaju funkcije koje programeri treba da kreiraju, a to podrazumeva definisanje potrebnih ulaznih podataka, manipulaciju sa njima i očekivani izlazni rezultat u odgovarajućoj formi, koja može biti jednostavan tekstualni ili grafički prikaz. Nakon specifikacije korisničkih priča, na programerima je da ih interpretiraju i procene potrebno vreme za njihovu implementaciju.

U fazi planiranja, u okviru svake iteracije postavljaju se prioriteti za korisničke priče, na osnovu dogovora između korisnika i programera. Na osnovu svog iskustva, programeri treba da uvide eventualne probleme koji se mogu pojaviti, npr. u integraciji trenutne i naredne verzije spredšita, nedostatak nekog podatka u trenutnoj iteraciji, a koji se pojavljuje u narednoj, ili nekompletne korisničke priče koje kasnije rezultuju nekompletnim spredšitom. Takođe, trebalo bi da imaju dovoljno iskustva da uvide međuzavisne funkcije, da bi ih kreirali u istoj iteraciji. Ukoliko situacije poput ovih nisu adekvatno rešene, prouzrokovaće nezadovoljstvo korisnika.

U fazi iteracije programiranje konačno počinje. Nakon podele korisničkih priča na zadatke, uočavaju se potrebni podaci i funkcije. Kada je reč o podacima, programeri treba da specificiraju formate podataka, način njihovog unošenja u aplikaciju, njihov prikaz i način izvoza. Sa druge strane, kada je reč o funkcijama, programeri imaju zadatak da napišu potrebne kalkulacije u formi funkcija, koje će prihvatati korisničke podatke i kreirati odgovarajući izlazni rezultat. Ovi rezultati mogu biti prikazani u tekstualnoj, tabelarnoj ili grafičkoj formi. Ukoliko se prikazuju grafički, i dizajneri i krajnji korisnici treba da se uključe u osmišljavanje načina prikaza podataka, da bi on sigurno prikazivao sve potrebne podatke i bio prilagođen krajnjem korisniku. Funkcije bi trebalo da se kreiraju na takav način da budu modularne, kako bi mogle da se koriste na više različitih mesta u aplikaciji, nezavisno od korisničke priče. Pisanje funkcija na ovakav način takođe pomaže pisanje testova. Testovi se koriste za validaciju funkcija i celog spredšita, pa ukoliko se validacija uspešno završi, programeri mogu da se prijave za sledeći radni zadatak.

U fazi produkcije, kao način za uvođenje nove aplikacije, preduzeće može da izabere veliki prasak, pod uslovom da su korisničke priče implementirane kako je zamišljeno i da je validacija uspešno završena, jer se onda greške ne očekuju ili se retko dešavaju. Sa druge strane, u slučaju uvođenja obimne spredšit aplikacije, adekvatnije rešenje može biti fazni ili paralelni pristup. Paralelni pristup takođe može da posluži kao još jedan način da se aplikacija testira, jer se svakodnevno paralelno radi na dva sistema: stari, koji treba da se zameni, i novi. Na ovaj način, krajnji korisnici imaju priliku da ocene spredšit u smislu brzog unosa podataka, brzog i čitljivog prikaza rezultata, dizajna, integracije i uopšteg utiska.

Na kraju, u fazi kontinualnog održavanja, automatizovane platforme za testiranje dobijaju na važnosti, iz razloga što su spredšit aplikacije poznate kao aplikacije u kojima su greške česte, a zahtevi se često menjaju. Ove platforme mogu da podrže promenu izvora ulaznih podataka, prikaza podataka, načina izvoza podataka, broja ulaznih parametara u funkciju ili logiku funkcije. Održavanje takođe treba da podrazumeva korisnički trening, da bi se korisnici obučili za korišćenje aplikacije na predviđen način. Korišćenje aplikacije na neadekvatan način može prouzrokovati nepredvidive greške, pa iz tog razloga programeri mogu da se odluče da postave dodatne restrikcije na spredšit ili kreiraju korisničke privilegije.

5.2. *Skram*

Skram predstavlja agilnu metodologiju za upravljanje i kontrolu složenih softverskih projekata koristeći iterativno inkrementalni pristup. Karakterišu ga jednostavni procesi sa jasnim ciljevima i brzim povratnim informacijama. Baziran je na ideji empirijskog modela upravljanja procesima, koji je idealan za projekte sa promenljivim i nepredvidivim zahtevima. Skram ne definiše šta projektni tim treba da radi ili kako nešto da uradi. Za to je zadužen sprecijalizovan projektni tim. Akcenat je na samoorganizovanim i autonomnim timovima koji vrše implementaciju softverskog inkrementa paralelno. Skram tim broji od pet do deset članova: vlasnik projekta koji predstavlja kupca, skram master, i ostatak tima koji je specijalizovan za različite oblasti razvoja. Skram se izvodi u tri faze: priprema, razvoj i završna faza.

5.2.1. Faze skram metodologije

Faza pripreme uključuje definisanje projektnih planova i ciljeva. Krajnji cilj je transformacija projektne vizije u radnu verziju proizvoda, tj. softvera. U ovoj fazi formira se tim, definišu se potrebni resursi i specificira inicijalna lista zahteva klijenta i članova tima koji definiše aplikaciju. Sastoji se od svih potrebnih karakteristika i funkcionalnosti, modifikacija, tehnologija i nadogradnja u budućim verzijama. Svaka od ovih stavki liste ima svoj opis, prioritet, i procene, poređane po prioritetu, imajući u vidu njihove koristi i moguće rizike. Skram tim prvo radi na stavkama sa najvišim prioritetom. Svaka promena u poslovnim zahtevima, tržišnim uslovima, tehnologiji i povratnim informacijama od strane korisnika, uzrokuje promene u listi zahteva.

Faza razvoja se vrši u iteracijama nazvanim sprint. Sprint se pažnjivo planira biranjem stavki iz liste zahteva. Svaki član tima dobija zadatke iz liste zahteva. Rezultat sprinta je inkrement sof-

tvera, nazvan demo, koji se prezentuje klijentu. Tokom sprinta, svakodnevno se održavaju sastanci nazvani dnevni skram, tokom kojih se vrši pregled stanja razvoja, potpomaže komunikacija u timu, i pronalaze načini za izbegavanje eventualnih prepreka tokom razvoja. Takođe, tokom ovih sastanaka, svaki član tima ima zadatak da ispriča šta je uradio od prethodnog sastanka, a šta planira da uradi do sledećeg, kao i da li ima nekih problema vezanih za njegove zadatke. Ovakav način rada omogućava da svaki član tima tačno zna do koje faze je projekat stigao.

Završna faza počinje kada su sve stavke iz liste zahteva implementirane i kada se skram tim i klijent slože da je softver završen. Tokom ove faze, vrše se završne integracije, testiranje i dokumentovanje sistema.

5.2.2. Primena skrama na razvoj spredšit aplikacija

Na osnovu opisanih karakteristika skram metodologije, može se zaključiti da se ona može uspešno primeniti na razvoj spredšit aplikacija, jer podržava promenljive i nepredvidive korisničke i druge zahteve, koji su karakteristični za spredšite. Za primenu skrama na razvoj spredšita za početak potrebno je imati ideju o osnovnim funkcionalnostima spredšita. Kasnije se ova ideja proširuje u detalje dok se ne postigne konačna verzija. Iako skram master upravlja timovima za razvoj spredšita, timovi su samoorganizovani i rade po svom rasporedu. Specijalizovani tim je u stalnom kontaktu sa vlasnikom proizvoda, tj. klijentom i krajnjim korisnicima kako bi dobio sve neophodne informacije.

Faza pripreme za cilj ima kreiranje liste zahteva. Na početku, ova lista treba da sadrži minimalne zahteve koji se tiču dizajna i organizacije prozora spredšita u smislu mesta za unos podataka, vizuelne prezentacije najvažnijih informacija, standarda za pisanje funkcija i prikaza njihovih izlaza. Kasnije se inicijalna lista proširuje novim zahtevima ili izmenom postojećih. Na osnovu opisa stavki definišu se njihovi prioriteti koji pomažu boljoj organizaciji tima.

U fazi razvoja, timski rad dostiže svoj vrhunac. Na dnevnim sastancima članovi tima vrše pregled dodeljenih zadataka iz liste zahteva, kako bi ih modifikovali ukoliko je potrebno. Takođe, kolaboracijom programeri mogu dobiti nove ideje koje se tiču npr. čitljivosti spredšita, ili modularnosti funkcija. Ovakvom organizacijom timskog rada, jedan zadatak zapravo obavlja grupa programera upućena u njegov sadržaj. Ovo je od posebnog značaja u situacijama kada postoje nedoumice, jer će se timskim radom one ukloniti i sprečiti stagnacija u razvoju.

Završna faza skrama može da se posmatra kao neka vrsta testiranja prethodne faze. U ovoj fazi, svi završeni zadaci se integrišu u jednu aplikaciju koja je spremna za puštanje u rad. Ukoliko je integracija uspešno izvršena, programeri dobijaju zadatak da napišu testove. Testovi treba da provere da li spredšit aplikacija podržava potrebne standarde formata podataka, da li definisane funkcije rade precizno kako za granične, tako i za obične slučajeve korišćenja, da li su formati izlaznih podataka adekvatni i slično. Ovim testovima se proverava i da li na spredšitu postoje neke neiskorišćene funkcije ili podaci, jer oni predstavljaju nepotrebne delove spredšita. Takođe, ukoliko je spredšit nepotrebno velikog obima, potrebno je proveriti ga ponovo. Ukoliko se pokaže da je potrebno kreirati dodatne restrikcije nad spredšit aplikacijom, programeri mogu definisati korisničke privilegije ili zaključati određene tabele ili ćelije, kako bi se sprečio neautorizovan pristup podacima. Najzad, završno dokumentovanje sistema je ostavljeno za sam kraj kada se vrši objedinjavanje svih inkremenata dokumentacije nastalih iz korisničkih priča.

6. ZAKLJUČAK

Nakon uvida u stručnu literaturu iz oblasti razvoja spredšit aplikacija, uviđa se da problem čestih i sličnih grešaka u spredšit aplikacijama zaslužuje veliku pažnju. Pošto neki autori smatraju da je uzrok ovog problema neadekvatna ili čak nepostojeća metodologija razvoja ovih aplikacija, ovaj rad prikazuje moguće rešenje u vidu primene netradicionalnih agilnih metodologija na razvoj spredšita. Nakon prikaza dve najpopularnije agilne metodologije, ekstremnog programiranja i skrama, zaključuje se da one mogu da ispune sve zahteve za uspešan razvoj spredšit aplikacije. Ideja ovog rada je da se pronađe brzo rešenje, a da se ne uvodi nova metodologija, nego pokušaju da se primene postojeće. Ekstremno programiranje i skram su izabrane iz razloga što podržavaju promenljive zahteve, timski rad i iterativan pristup. Poredeći ove metodologije, zaključuje se da ekstremno programiranje više odgovara preduzećima sa višim nivoom kontrole, dok skram ostavlja izvesnu slobodu u organizovanju tima. Takođe, ekstremno programiranje veći naglasak stavlja na komunikaciju između programera i krajnjih korisnika spredšit aplikacije nego skram. Takođe, obe metodologije naglašavaju potrebu da se testiranje i dokumetovanje aplikacije vrši uporedo sa razvojem. Na ovaj način postiže se ušteda u vremenu i preciznija dokumentacija. Veliki akcenat je na timskom radu, što je veoma važno kada se greške otkriju u ranim fazama, jer tim radi na njima, pa se one dalje ne prenose. Budući pravci istraživanja odnose se na demonstraciju primene ekstremnog programiranja i skrama na praktičnom primeru spredšit aplikacije. Ovo će biti od važnosti za polje istraživanja vezano za razvoj spredšit aplikacija, jer je ovaj rad napisan samo sa teorijskog stanovišta.

LITERATURA

[1] Fowler, M., & Highsmith, J. (2001). The Agile Manifesto.*Software Development*,28-32

[2] Grossman, T. A. (2002). Spreadsheet Engineering: A Research Framework. *European Spreadsheet Risk Interest Group Symposium.*

[3] Highsmith, J. A. (2004). *Agile Project Management.* Boston: MA: Addison-Wesley.

[4] Highsmith, J. (2002.). *Agile Software Development Ecosystems.* Addison-Wesley Professional.

[5] Išljamović, S. (2012). Mogućnost razvoja spredšit inženjerstva po uzoru na metodologiju razvoja informacionih sistema. *Strategijski menadžment.* Bor: Univerzitet u Beogradu, Tehnički fakultet u Boru, Odsek za menadžment.

[6] Knight, B., Chadwick, D., & Rajalingham, K. (2000). A Structured Methodology For Spreadsheet Modelling. *Spreadsheet Risks, Audit and Development Methods. 1*, pp. 43–50. EuSpRIG.

[7] Panko, R. R. (2007). Recommended Practices for Spreadsheet Testing. *CoRR* .

[8] Powell, S. G., Lawson, B., & Baker, K. R. (2007). Impact of Errors in Operational Spreadsheets. (pp. 57-67). Eusprig.

[9] Rajalingham, K., Chadwick, D. R., & Knight, B. (2001). Classification of Spreadsheet Errors. *EuSpRIG.*

Tamara Valok Radulović
Fakultet organizacionih nauka
Kontakt: tamara.valok@fon.bg.ac.rs
Oblasti interesovanja: Razvoj zasnovan na modelima, Savremene softverske arhitekture, Projektovanje informacionih sistema, Metodologije razvoja informacionih sistema