EKSTREMNO PROGRAMIRANJE I PRIMJENA NA BALKANU

EXTREME PROGRAMMING AND APPLIANCE IN THE BALKANS

Borislav TADIĆ
Banja Luka

Ključne riječi: ekstremno programiranje, xp, agilni razvoj softvera, osiguranje i kontrola kvaliteta softvera, testiranje, komunikacija u razvojnom timu, kontakt s klijentom

REZIME

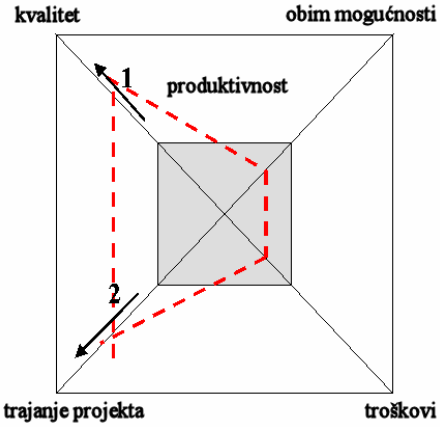
Osnovne karakteristike Ekstremnog programiranja (Extreme Programming, XP) kao metoda „agilnog razvoja softvera" su: programiranje u parovima, zajedničko vlasništvo nad kodom, prisustvo klijenta na mjestu razvoja, dnevna i klijentu vidljiva poboljšanja dijelova i integrisanje u cjelinu na kraju radnog dana, česta i temeljna testiranja, jednostavni dizajn, feedback, oslanjanje na metafore i 40-satna radna sedmica. XP je orijentisan na male timove (2 do 12 programera), pa imajući u vidu orijentaciju na elemente kvaliteta i mogućnost lake praktične primjene, predstavlja povoljnu metodu u razvoju softvera na prostorima Balkana. Tema rada su osnovni elementi, okvirna primjena, pregled nekoliko iskustva iz prakse i stručna literatura, te uočeni propusti XP-a.

SUMMARY

Main charateristics of Extreme Programming (XP) as a method of agile software development are: pair programming, collective ownership, on-site customer, simple design, short releases and continous integration, frequent testing, feedback, refactoring, metaphors and 40 hour workweek. XP is tailored for small teams (2 to 12 programmers) and having in mind its orientation on quality elements and possibility of simple practical implementation, it represents favorable method in software development in the Balkans. Theme of this paper are the basic elements, mainstream implementation, overview of several experiences and expert literature, as well as the contras of XP approach.

1. PROBLEMI U RAZVOJU SOFTVERA

U razvoju softvera i danas se suočavamo sa brojnim problemima. Informatika je relativno mlada disciplina, pa su pred nama često novi, nepovezani ili nepredvidivi problemi, za čije uklanjanje u drugim strukama postoje razrađeni mehanizmi. Vrijednost softvera se često potcjenjuje, a objektivne procjene napretka u razvoju softverskog projekta je teško napraviti. U ovim projektima 4 varijable se postavljaju kao elementarne: kvalitet, obim mogućnosti, trajanje i troškovi. H. M. Sneedov „đavolji" kvadrat kapaciteta projekta (Slika 1.) jasno ukazuje da četvrta varijabla zavisi od tri koje bira naručilac softvera, a razvojni tim mora predstaviti izvodivost četvrte varijable korisniku.



Slika 1. Sneedov kvadrat

U vezi sa ovim, 70-tih godina su postavljena četiri cilja softverskog inženjeringa: održavanje dogovorenih troškova, termina i specifikacije, te unaprijeđenje kvaliteta.
Koliko se u ovome uspjelo, najbolje svjedoči CHAOS Report [1] prema kojem samo 16,2% svih projekata u razvoju softvera postigne svoj cilj, dok se 52,7% završi bez ostvarenog cilja i 31,1% bude prekinuto tokom razvoja. Iako se ove cifre kreću u pozitivnijem smjeru u posljednjoj deceniji [2], značajno prekoračenje vremenskih rokova i predviđenih troškova, te manji isporučeni funkcionalitet od planiranog još uvijek redovno prati razvoj softvera. Osnovni problemi su promjene ili nejasnost poslovnih ciljeva, pogrešne funkcionalnosti, nedostatak podrške menadžmenta, česte korekcije softvera, nerealno postavljani termini, nivo tolerisanih grešaka, promjene angažovanog osoblja, nerentabilnost sistema na duži vremenski period i prekid projekta kada je već u kritičnoj fazi. Uzroci problema su to što dobar dio ICT subjekata još uvijek ne(dovoljno) prihvata projektne modele, praktikuje nizak stepen interakcije sa klijentom, rad bazira na nepotpunim procjenama, a projekte vode nedovoljno iskusni menadžeri. Subjekti koji su na ICT sceni duži niz godina sporo se rješavaju starih navika, ne shvatajući da je softver postao vrlo kompleksan i da njegove specifikacije ne mogu stati na jednu stranicu A4 formata.
Ekstremno programiranje je u 6 godina svog postojanja u porodici agilnog razvoja softvera pokazalo da se vrlo dobro nosi sa ovim, ali i drugim problemima u razvoju softvera, poput izgradnje kvalitetne komunikacije u razvojnim timovima kao i tima sa klijentom.

2. AGILNI RAZVOJ SOFTVERA

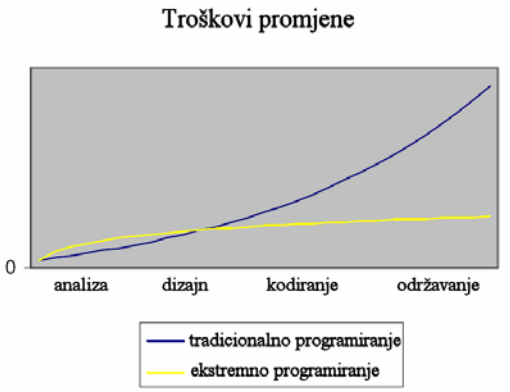
Agilni razvoj softvera (eng. agile software development, ASD) je porodica metodologija koja predstavlja konceptualni okvir za softverske projekte. Temelji se na "agilnom manifestu" iz 2001. koji preferira: pojedince i interakcije iznad procesa i alatki, softver koji funkcioniše iznad sveobuhvatne dokumentacije, saradnju sa klijentom iznad jednokratnog utanačenja ugovora i odgovore na promjene iznad praćenja plana.
Agilne metode su nastale kao kontrateža metodama "teške kategorije" (npr. Racionalni unificirani proces), pa su prepoznatljive kao metode "lake kategorije". Pogodan teren za primjenu agilnih metoda je nizak stepen kritičnosti, iskusni programeri, visok stepen promjene zahtjeva, mali broj developera i kultura koja počiva na haosu [3]. Neke od metoda su razvijene prije pisanja manifesta: Ekstremno programiranje, Scrum, Crystal clear, Adaptivni razvoj softvera i DSDM, a nove su: Feature driven development, Lean software development, Microsoft solutions framework, Agile documentation i dr.
Principi agilnog razvoja softvera [4]:

- Naš najviši prioritet je da zadovoljimo klijente kroz rane isporuke i kontinuirani razvoj vrijednog softvera
- Neka zahtjevi za promjene budu dobrodošli, čak i kasno u razvoju. Agilni procesi baštine promjenu konkurentne prednosti klijenta
- Isporučuj softver koji radi često, od par sedmica do par mjeseci, sa preferiranjem kraćeg vremenskog roka
- Poslovni ljudi i programeri moraju raditi dnevno zajedno kroz projekt
- Gradi projekte oko motivisanih pojedinaca. Daj im okruženje i podršku koju trebaju i vjeruj da će uraditi posao kako treba
- Najefikasniji i najefektivniji metod dolaska do informacija za i unutar razvojnog tima je razgovor lice-u-lice
- Softver koji radi je osnovna mjera napretka
- Agilni procesi promovišu održivi razvoj. Sponzori, programeri i korisnici moraju biti u mogućnosti da zadrže konstantan korak neodređeno dugo
- Kontinuirana pažnja na tehničku izvrsnost i dobar dizajn poboljšavaju agilnost

- Jednostavnost – umjetnost maksimizacije količine rada koji se ne mora uraditi - je esencijalna
- Najbolja arhitektura, zahtjevi i dizajn proizilaze iz samoorganizirajućih timova
- U redovnim intervalima tim saznaje kako postati efektivniji i respektivno prilagođava ponašanje
-

3. EKSTREMNO PROGRAMIRANJE

Ekstremno programiranje (eng. eXtreme Programming, dalje: skr. XP) je metod ili pristup softverskom inženjeringu i najpopularnija metodologija ASD-a. Formulisali su ga 1999. godine Kent Beck, Ward Cunningham i Ron Jeffries. XP je orijentisan na male timove i opisan kao "mehanizam za socijalne promjene, stil razvoja, put ka poboljšanju, pokušaj da se pomire humanost i produktivnost, te disciplina razvoja softvera" [5]. On omogućava proizvodnju dugotrajnog softvera i sposobnost da se reaguje na iznenadne i zahtjeve koji se mijenjaju. XP razvoja softvera zasniva na vrijednostima jednostavnosti, komunikacije, povratne informacije (dalje: pi.), hrabrosti i poštovanja. Cijeli tim se povezuje prateći 12 praksi i raspolaže sa dovoljno pi. koje mu omogućuju da vidi gdje je i prilagodi djelanje nastaloj situaciji. XP projekti stvaraju dnevno realnu ekonomsku vrijednost za klijente i mogu se u potpunosti upravljati njegovim zahtjevima. Riječ "ekstremno" u nazivu duguje činjenici što su one prakse programera koje su odavno prepoznate kao uspješne, spojene u jednu metodu i dovedene na intenzivniji nivo. XP pripada ASD-u jer se koncentriše na nekoliko ključnih aspekata procesa i daje programerima slobodu da ostalo prilagode formalnoj validaciji, konfiguraciji menadžmenta, prilikama u okruženju, vremenskim rokovima, pa i standardima kvaliteta (npr. ISO 9001).



Slika 2. Troškovi promjena projekta u tradicionalnom i ekstremnom programiranju.

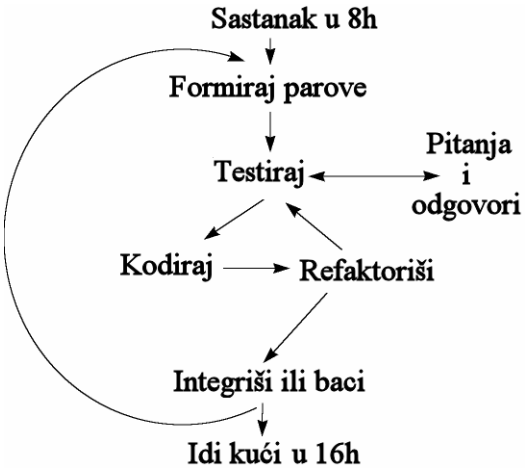
3.1. Preduslovi

XP tim se sastoji od 2 do 12 programera, supervizora/trenera sa strane menadžmenta, te jednog ili više klijenata. Uloge trenera i supervizora su bitne: prvi se stara da tim prati disciplinovano tehnike XP-a i podsjeća ih na kršenja, a drugi prati rashode i status projekta i evidentira ih. Radna sredina je otvorena: jedna velika ili više povezanih soba, sa oglasnom tablom, te po mogućnosti računari postavljenim u krug tako da se može programirati udvoje za jednom mašinom. Pred menadžmentom i klijentom su veliki izazovi. Prvi moraju da se naviknu da ne propisuju timu šta mora raditi, već da samo razvijaju metrike za praćenje projekta i prezentovanje uočenih problema klijentu, te prate prakse XP-a. Klijentova uloga nije pasivna - on mora biti spreman da aktivno učestvuje u razvoju projekta, postavljanjem konkretnih zahtjeva, razjašnjenjem nejasnoća i pripremom testova kojim bi se uvjerio da sistem radi ono što on od njega očekuje.

3.2. Tok

Osnovna ideja XP-a je sadržana u trouglu „kodiraj, testiraj, refaktoriši" (Slika 3.) (nasuprot dizajniraj, kodiraj, testiraj u tradicionalnim metodama). Radni dan izgleda ovako:

1. Desetominutni stojeći sastanak prije početka rada: daje odgovore na pitanja o statusu projekta i zadacima za tekući dan. Svaki programer bira zadatak(e) za danas ili nastavlja one koji nisu završeni – nijedan ne traje duže od tri radna dana.
2. Zatim se formiraju parovi – onaj ko je voljan da programira preuzima „upravljač“ i razmišlja taktički, drugi analizira kod koji prvi stvara, pravi strateške procjene i uklapa kod u nastajanju u „čitavu sliku“. Partneri moraju biti spremni na i do nekoliko smjena za tastaturom dnevno.
3. Pisanje unit testova u kojima se predviđaju mogući problemi, korisničke akcije i tok te komponente. Sve se dešava prije pisanja koda, a testovi se dalje automatski odvijaju.
4. Pisanje koda uz poštovanje standarda kodiranja je usko povezano sa prethodnim korakom – realizuje se najjednostavniji mogući kod da test uspije, a za nejasnoće u testovima ili programiranju obraća se direktno prisutnom klijentu.
5. Klijent je tu da odgovori na sva pitanja, donese brze odluke i kreira testove prihvatanja.



Slika 3. Radni dan iz XP ugla

6. Dalje se vrši refaktorisanje čiji se rezultat mora uklapati u sve postojeće testove.
7. Nakon završenog zadatka, pristupa se integraciji sa verzijom programa koja sadrži dosad obrađene komponente ostalih programera. Integracija mora da prođe 100% uspješno, zvanična verzija ne smije biti narušena, a ako to nije moguće zadatak se ignoriše i njegovo programiranje ponavlja drugi dan. Koraci 2-7 se ponavljaju u maksimalno polusatnim ciklusima uz formiranje novih parova (ako je to potrebno ili pogodno).
8. Zadaci su izvršeni, verzija za javnost je stoprocentno funkcionalna - nakon 8 radnih sati vrijeme je da se pođe kući. Obratite pažnju da programerima ništa ne "visi nad glavom" - sve je prošlo ili odbačeno, nema polovičnosti.

3.3. Temeljne vrijednosti, principi i aktivnosti

XP se definiše kroz pet vrijednosti [5-13]:

- Komunikacija. Umjesto komunikacije kroz dokumentaciju, XP tehnike omogućavaju širenje institucionalnog znanja između članova tima. Svi programeri zajedno stiču uvid u sistem koji ima i korisnik.
- Jednostavnost. XP započinje sa građenjem najosnovnije verzije sistema koja „radi“ i postepenom refaktorisanju u bolju. Potpuna koncentracija na ono što korisnik treba u ovom trenutku, izbjegava mogućnost ugradnje mogućnosti koje kasnije uopšte neće biti korištene. Jednostavan dizajn komponenti sistema, kao i pregledan kod mogu lako razumjeti svi članovi tima.
- Povratna informacija (Feedback). Povratne informacije u ekstremnom programiranju: (a) pi. sistema koja se ostvaruje pisanjem unit testova, (b) pi. klijenta – pomoću funkcionalnih testova periodično se sam klijent/korisnik uvjerava u napredak projekta i vraća dragocjene instrukcije timu, (c) pi. tima klijentu na njegove zahtjeve u igri planiranja je ocjena vremena koji je potrebno za implementaciju.
- Hrabrost. Hrabrost u kontekstu XP-a znači spremnost da se programira samo za potrebe klijenta danas, da se nema strah od prepravke koda ili kada treba eliminisati dio koda ili

čak sve početi iz početka, spremnost da programer ocijeni granicu produktivnog dnevnog rada, te sutra počne svježe glave.
- Poštovanje. Ono se odnosi na druge članove tima, jer programer bez obzira na tempo projekta ne krši dogovor i konvenciju. Tu je i poštovanje sopstvenog rada kroz veći kvalitet koda i poštovanje radne satnice. Poštovanje klijenta prema programerima, i u suprotnom smjeru poštovanje klijentovih želja i laičkog shvatanja sistema. Poštovanje menadžmenta prema developerima, kao i poštovanje programera za uvid menadžera. Nedovoljno poštovanja ukazuje na probleme u komunikaciji.

Četiri principa ekstremnog programiranja su bazirana na spomenutim vrijednostima i lako ih je transformisati u vodiče za važne odluke za projekt [5-13] :
- Rapidna povratna informacija. Pi. je najkorisnija ako se dobija i isporučuje rapidno, jer je vrijeme koje prođe između akcije i pi. kritično za učenje i promjene. Klijent pruža pi. jer je stalno uključen u proces, a unit testovi reaguju na promjene u kodu i ne propuštaju greške.
- Pretpostavi jednostavnost. Svaki problem može biti riješen ekstremno jednostavno, čime se odbacuju ideje o planiranju proširenja i koda za ponovnu upotrebu.
- Inkrementirajuće promjene. XP ne zagovara velike promjene, već male inkrementalne promjene - npr. izdavanje verzije svake tri sedmice. Mnogo, ali malih koraka daje klijentu veću kontrolu na sistemom koji se razvija.
- Prihvatanje promjene. Nasuprot klasičnom procesu u kojem se prva slika nastala na osnovu korisničkih zahtjeva nastoji održati što duže, u XP-u je razvojni tim potpuno spreman na promjene. Djeluje kao da je svejedno da li radi po starom planu ili po novorazvijenim okolnostima.

Četiri osnovne aktivnosti koje XP priznaje u okviru razvojnog procesa su [5-13]:
- Kodiranje. Jedini stvarno važan proizvod razvoja sistema je kod, a bez njega nema rezultata. Prema XP-u, kodiranje može biti crtanje dijagrama koji će generisati programski kod, skriptovanje web-baziranog sistema ili objektno orijentisanog programa koji se treba kompajlovati. Kodiranje određuje najpovoljnije rješenje: ako postoji nekoliko alternativa problemu, one bi se sve trebale isprogramirati, a onda unit testovi pokazuju koje je rješenje najbolje. Kroz kodiranje se misli o programskim problemima lako kanališu; programer pred složenim problemom u samom kodu kolegama pokazuje o čemu je riječ, a pi. dobija na isti način.
- Testiranje. Bez testiranja, ne možemo biti sigurni u bilo šta - da li je ono što je isprogramirano ono na šta ste na početku mislili (unit testiranje) i da li je ono što ste mislili ono što ste trebali misliti (testovi prihvatanja).
- Slušanje. Programeri ne moraju ništa znati o poslovnoj strani razvoja sistema - njegovu funkciju određuje klijent. Da bi isprogramirali potrebne funkcionalnosti oni moraju da saslušaju potrebe klijenta, pokušaju razumjeti njegove probleme i predstaviti mu rješenja.
- Dizajn. Izuzetno bitan, da nas primjena prethodne 3 aktivnosti ne bi dovela u ćorsokak, jer sistem postane prekompleksan, a veze između komponenti nevidljive. Dizajniranjem strukture koja organizuje logiku u sistemu nestaju nepotrebne zavisnosti između modula, a time i promjene na jednom modulu ograničavaju svoj uticaj.

3.4. Dvanaest osnovnih praksi

U XP-u je precizno definisano 12 usko vezanih praksi[5-13], koje su naslonjene na tri stuba softverskih projekata: programiranje, tim i procese. One nisu nove - samo su praktično prepoznate kao najbolje i izvedene iz višegodišnje prakse softverskog inženjeringa.

- Igra planiranja – Brzo odredite opseg i svrhu sljedećeg izdanja/isporuke kombinovanjem poslovnih prioriteta i tehničkih procjena. Klijent piše "korisničke priče", tj. osobine koje program treba da sadrži. Razvojni tim potom procjenjuje koliko se truda i vremena ulaže u implementaciju te osobine, nakon čega klijent ponovo procjenjuje koje će priče biti implementirane, u kojem redoslijedu i u kojim intervalima. Usvojeni plan u daljem toku ažurirajte ako situacija nalaže korekciju.
- Kratka isporuka softvera – Pustite jednostavan sistem sa minimalnim brojem osobina u proizvodnju rano, a zatim isporučujte nove verzije sa nekoliko dodatnih osobina u veoma kratkim ciklusima (iteracijama). One traju od 1 do 3 sedmice, a na kraju svake stoji potpuno funkcionalan, testirani sistem sa novim i za klijenta vrijednim funkcionalitetom.
- Metafora – Vodite sav razvojni proces sa jednostavnom zajedničkom pričom kako sistem radi. Programeri, klijent i menadžment razvijaju zajednički rječnik, da bi mogli raspravljati o sistemu - otvoreno i iskreno.
- Jednostavni dizajn – Uvijek koristi najprostiji mogući način da završiš posao - sistem treba biti dizajniran što je moguće jednostavnije u svakom datom momentu. Dodatna kompleksnost se uklanja čim se otkrije, jer ako se zahtjevi klijenta promjene, tek se onda pojavljuje potreba za dodatnim opcijama ili promjenom postojećih mogućnosti.
- Testiranje – Prije dodavanja nove osobine programu, prvo napiši testove. Programeri kontinualno pišu automatizovane unit testove za svaku najmanju komponentu, koji se moraju izvršavati besprijekorno da bi se razvoj nastavio. Klijenti pišu testove prihvatljivosti cijelog sistema ili njegovog većeg dijela da bi se uvjerili da su osobine ugrađene i da softver ispunjava njihove zahtjeve.
- Refaktoring – Programeri poboljšavaju dizajn sistema bez promjene njegovog ponašanja da uklone duplikate u kodu, poprave komunikaciju, pojednostave kod ili mu dodaju fleksibilnost. Ovo je lako uraditi jer postoje testovi koji osiguravaju da se ništa ne "pokvari".
- Programiranje u parovima – Sav proizvedeni kod pišu dva programera za jednom mašinom ("upravljačem") i rotiraju se za tastaturom. Oni komentarišu alternative razvoja. U suštini, tako se kod evaluira dok se stvara, a znanje konstantno poboljšava kod svih članova tima.
- Zajedničko vlasništvo koda – Niko ne posjeduje modul. Bilo koji programer može promijeniti kod bilo gdje u sistemu u bilo koje vrijeme i na to mora biti spreman. Rezultat je viši kvalitet koda i širenje znanja.
- Standardi kodiranja – Svi pišu program po predefinisanom standardu i u skladu sa pravilima koji naglašavaju komunikaciju kroz kod. U idealnom slučaju, niko ne bi trebao biti u mogućnosti na osnovu pregleda koda zaključiti ko ga je pisao.
- Kontinualna integracija – Integriši i "gradi" sistem nekoliko puta u jednom danu, svaki put kada je neki zadatak završen. Sistem mora ispuniti testove 100% nakon svake integracije.
- 40-satna radna sedmica – Programer ide kući na vrijeme i ne radi više od 40 sati sedmično. Nikad ne radi prekovremeno dvije sedmice zaredom, a ako se to desi, to upućuje na jasne probleme u projektu i procesu.
- Klijent na licu mjesta – Uključi živog realnog korisnika u tim, koji je stalno dostupan da odgovara na pitanja. Za komercijalni softver sa više kategorija korisnika, morao bi biti prisutan zastupnik (npr. produkt menadžer) ili više korisnika. Klijent nije onaj ko plaća softver, niti stvarni krajnji korisnik.

4. ISKUSTVA U PRAKSI – XP PRO I CONTRA

Brojni IT subjekti u svijetu koji ispunjavaju preduslove su uspješno primjenili XP tehnike. Četiri tima američkih i evropskih kompanija naveli su pro i contra argumente:

- Firma FST koja se bavi dizajnom, izradom i outsourcingom Internet usluga, primjenila je XP [16]. Njena iskustva na dva projekta govore o potpunom neprihvatanju metafore u korist UML-a. Polovično su prihvaćene prakse standarda kodiranja jer ih je bilo teško ukomponovati sa postojećim modulima, kao i kontinuirane integracije, jer se ona dešavala sedmično, a ne dnevno. FST je posebno zadovoljan kvalitetom koji je postignut i koji je mjeren prema ISO 9001 standardu, navodeći da XP poboljšava koliko programersku, toliko i efikasnost menadžmenta, te činjenicom da je zbog svoje agilnosti i davanja tek ključnih principa dozvolio timu da ostalo potpuno prilagodi svojim potrebama.
- ThoughtWorks je u stisci sa vremenom na web-aplikaciji zasnovanoj na Enterprise Java Beans [17] odlučio da primjeni tehnike ekstremnog programiranja. 40-satna radna sedmica nije usvojena, jer se zbog kašnjenja i zavisno od iteracije radilo i do 60 sati. Programiranje u parovima je primjenjeno na samo 1/3 radnog vremena, metafora je odbačena (bez pokušaja implementacije), a ni kratka isporuka nije našla primjenu. U ulozi klijenta bili su projekt menadžer i timski analitičar. U procesu se vodilo računa o standardima, no zbog kašnjenja u startu i ova praksa je polovična.
- Veliki američki programerski tim koji je radio na multiprocesorskoj NT aplikaciji kritičnoj za bezbjednost iz [18] navodi da nije usvojio metaforu, jer su umjesto nje korišteni UML dijagrami i tekstovi objašnjenja, kao ni otvoreni radni prostor, jer njega fizički nije bilo moguće organizovati. U igri planiranja odlučili su se za use cases umjesto korisničkih priča, ali su potpuno proveli ograničenje obima, podjelu na zadatke i prijavljivanje za zadatke. Ulogu klijenta je preuzeo sistemski inženjer, kojega je tim nekoliko puta preduhitrio izvršenjem zadataka u odnosu na njegovu pripremu testova prihvatanja.
- U projektu za TransCanada Pipelines Limited [19] takođe se metafora ne primjenjuje u potpunosti (koriste modele i riječi za opis sistema), zatim pisanje testova su prvo praktikovali samo iskusniji programeri, a od dva reprezentativna klijenta (web usluge i legacy aplikacija), samo je prvi bio prisutan kao on-site customer.

Sve spomenute prakse koje eksplicitno nisu navedene su potpuno uspjele; one koje su neosporno prihvaćene u svim datim slučajevima su refaktoring, kolektivno vlasništvo nad kodom, jednostavni dizajn i intenzivni „prvo-testiraj“ metod. U svim slučajevima, koji su se po obimu i tipu razlikovali, 2/3 praksi je potpuno prihvaćeno i ocijenjeno kao značajno poboljšanje u odnosu na ranije primjenjivane tehnike uz unapređenje efikasnosti za 30-80%.
[20] navodi unit testove, kontinuiranu integraciju i jednostavnost kao najbolje, baš kao i limit na 40 radnih sati i prisan rad sa klijentom. Kritikuje se sljedeće: refaktoring se ne smije pretvoriti u stihijski dizajn, planiranje (i pored za tim naporne analize) se ne smije dešavati „u hodu“, a programiranje u parovima se smatra nemogućim zbog nespojivosti istovremenih konsultacija sa kolegom i programiranja u naletu kreativnosti i inspiracije.
XP u mnogočemu podsjeća na naučni metod [21]. Programiranje u paru potvrđuje da je istraživanje provedeno u naučno-urednom maniru. Razvoj korisničkih priča je ključno za kolekciju zahtjeva i razumijevanje domena problema, a klijent na licu mjesta nezamjenjiv faktor poboljšanja kvaliteta softvera, naročito one komponente koja se odnosi na njegovu subjektivnu primjenu. Testovi određuju stanje isporuke u bilo kojoj tački vremena - na isti način na koji se naučni metod oslanja na eksperimente. Ako se razvije dio koda (nova teorija) ona se mora reevaluirati da bi bila u saglasnosti sa postojećim teorijama. XP zapravo izuzetno praktičnu disciplinu programiranja, dovodi na novi teorijski nivo. Puni kapacitet dostiže u oblastima dijeljenja znanja, komunikacije i razumijevanja razvojnog tima.

5. KONTROLA I OSIGURANJE KVALITETA SOFTVERA I XP

Danas su svi softverski timovi pod stalnim pritiskom da se kvalitet njihovih projekata poboljša. Prakse testiranja koje se koriste da se proizvod ocijeni igraju ključnu ulogu u osiguranju kvaliteta. Kako je XP orijentisan na ljude on utiče na još jedan segment kvaliteta: kvalitet komunikacije unutar tima i tima sa klijentom, čime opet unapređuje kvalitet procesa, a tim i proizvoda. Baziran na „lightweight“ specifikaciji, testiranju prije kodiranja i testiranju cijelog sistema, kao i uključivanju klijenta u svaki korak procesa, proizvedeni kod sadrži značajno manji broj grešaka. XP oficijelno ne preporučuje nijedan standard kvaliteta, poput IEEE, SEI, ISO900x, tj. formalni mehanizam pisanja zahtjeva ili snimanja stanja. XP zato usmjerava klijente, programere i testere na razumijevanje konačnog cilja projekta [6,7].
Proizvodnja softvera zahtijeva formalno osiguranje kvaliteta, što se izvršava odg. programima osiguranja i kontrole kvaliteta. Prvi se obavlja na početku i zahtjeva ranu investiciju, dok se kontrola vrši nakon razvijanja proizvoda, kroz funkcionalne i sistemske testove. Proces kontrole kvaliteta u razvoju softvera se čini efikasnijim od osiguranja kvaliteta. Ipak, problem leži u činjenici, bez obzira na broj testova koje su razvili programeri ili tim za osiguranje kvaliteta, nije moguće predvidjeti sve slučajeve i uslove koji čekaju softver nakon isporuke. Uparivanje agilne metodologije i osiguranja kvaliteta daju najbolje rezultate. Troškovi se povećavaju zbog ugrađivanja kontrole kvaliteta u proces od početka, ali proces održavanja softvera u koji za klijenta ide preko 90% ukupnih troškova softvera [2] košta značajno manje.
Korisničke priče su još jedan temelj kvaliteta softvera koji se dobija primjenom XP-a. U klasičnom procesu isporuke softvera, klijent na početku izražava zahtjeve budućeg sistema, ali vrlo rijetko ih konkretizuje. Tek po isporuci, klijent uviđa razliku između njegovih zamišljenih funkcionalnosti i načina na koji ih je programer shvatio. Korisničke priče u procesu povećanja kvaliteta ne dopuštaju dvosmislenosti. Ono što korisnik želi, on to jasno definiše u velikom broju priča koje piše na kartice, a dalje ih kako proces odmiče redefiniše i kroz testove prihvatanja kontroliše koliko se slažu sa napisanim pričama. Programer date priče dijeli u nekoliko malih cjelina, pogodnih za transformaciju u komponente programa, zadatke. U XP procesu ne samo da je potrebno manje testera (programer je ujedno i tester), već je mali tim testera upravo onaj koji podržava programera, podupire stvaranje dobrog koda i podiže svijest o važnosti testiranja i kvaliteta.
Bolji kvalitet proizvoda u XP-u osiguravaju i konstantna interakcija sa klijentom i redovno testiranje kroz sve faze razvoja [14]. Dva tipa testova određuju softverski projekt u XP-u: testove prihvatljivosti razvijaju klijent i inženjeri kvaliteta, a funkcionalne testove programskih modula razvijaju programeri. Oba tipa testova se obavljaju često. Funkcionalni testovi se fokusiraju na komponentu u razvoju, a nakon njene finalizacije i uklapanja u cjelinu, mogu se ukloniti ili uključiti u neki veći sistemski test. Ukupna baza testova je svakim danom sve veća i sveobuhvatnija, što testerima i inženjerima kvaliteta daje potpuni uvid u status razvoja. Testovi prihvatanja, na kojima je najbolje da rade oni zaduženi za kvalitet, testiraju svaku opciju koju sistem nudi pojedinačno. Stojeći sastanci su važan segment kvaliteta u XP-u. Jutarnje izlaganje svakog člana tima koje traje svega par minuta pomaže da se problemi uoče i prevaziđu, svi informišu o trenutnom napretku, kao i da se svakom pruži prilika za komentare. Menadžerima je ovaj segment bitan, jer u tradicionalnim metodama oni pi. dobijaju (ali i traže) često prekasno.

6. XP NA BALKANU

Ekstremno programiranje je idealno za timove do 12 ljudi koji se nalaze fizički u jednom poslovnom prostoru i koji rade na malim projektima čiji je rizik takođe mali. Naš je teren

izuzetno pogodan za implementaciju XP-a. U prvom redu, na području Bosne i Hercegovine, Srbije i Crne Gore i Hrvatske dominiraju mali poslovni subjekti (po standardima EU do 50 zaposlenih) u ICT sektoru. Naš ICT sektor rijetko potpisuje ugovore za projekte veće od 200 MD (man-days), a većina subjekata nije upoznata sa metodologijama, ni projektnim modelima u razvoju softvera. Reprezentativni uzorak balkanskog ICT subjekta je preduzeće sa desetak zaposlenih na jednoj geografskoj lokaciji, koje je angažovano na kratkoročnim po obimu malim do srednjim projektima u direktnom kontaktu sa klijentom/naručiocem posla za lokalno tržište. S obzirom da je naša softverska industrija u razvoju, time je teren pogodniji za uvođenje XP-a. Najbolje rezultate će postići timovi i firme koji sa njim počnu raditi od formiranja, a njih će opet najbolje prihvatiti mlade kompanije kao klijenti.

Prvo direktno unapređenje koje bi XP mogao unijeti softversku produkciju na Balkanu je uvođenje radne discipline i pozitivnih navika u razvojne timove. Standardi kodiranja i jasna pravila rada u mnogim organizacijama gotovo da ne postoje, što u integraciji koda nerijetko predstavlja veliki problem. Komunikacija između članova razvojnog tima je ograničena. Programiranje u parovima bi, pored izgradnje povjerenja i višeg morala u radnim jedinicama, značajno doprinijelo discipaciji znanja u timovima. Bolja komunikacija sa klijentima u svim fazama razvoja uz povećanje kvaliteta donijela i bi proizvode koji su prilagođeniji klijentovim željama. Najčešći problem je ipak prekoračenje rokova, a primjena kratkih ciklusa izdavanja softvera bi ovome stala u kraj. Za ova dva segmenta, neophodno je doobrazovanje programera, ali i transparentnije upućivanje klijenata u osnove procesa isporuke softvera. Opšti je utisak da se malo pristupa praksama testiranja na Balkanu. Uz rijetke izuzetke, programira se stihijski, a rezultati rada tek vide po isporuci i korekciji grešaka. Otvoreni radni prostor omogućava da jedan prog. par indirektno čuje/vidi na čemu radi drugi, što dodatno poboljšava komunikaciju. Dodatna prostorija za klijenta, postavlja se gotovo idealnom za njegovo obavljanje paralelnih aktivnosti i optimizaciju utrošenog vremena. Prava programera kao radnika bila bi unaprijeđena 40-satnom radnom sedmicom. Motivacija bi svakako drugačije izgledala u tim uslovima rada, otvorene radne sredine i povećanog stepena interakcije sa kolegama i klijentom.

Iako se radi o zreloj metodi, ekstremno programiranje je tek sad u pionirskoj fazi prihvatanja u evropskim firmama. Tu ujedno leži i šansa firmi iz jugoistočne Evrope. Osim u SAD i Kanadi, primjetnu podršku programerske komune XP ima u Francuskoj, Irskoj, Švedskoj i Ukrajini[15], a njene prakse se sve intenzivnije izučavaju na evropskim visokoškolskim ustanovama IT usmjerenja. Popularna je i u Kini, Indiji, Japanu i Brazilu. Iskustvo sa XP-om bi se moglo pojaviti kao vrlo povoljan faktor u saradnji sa inostranim partnerima u IT branši.

Pored pogodnog tla, pred implementacijom stoje i problemi. Među mlađom populacijom su prisutni brojni solo-programeri (“one-man-team”) za koje ekstremno programiranje jednostavno nije primjenjivo. U velikim preduzećima direktan kontakt sa klijentima razvojni tim najčešće nema. Mali broj klijenata je spreman uložiti čovjeka u projekt, tj. učiniti ga stalno dostupnim razvojnom timu. Ako se XP pogrešno interpretira, može dovesti i do loše dokumentacije, koja se zbog prisustva klijenta i bliske saradnje tima često zapostavi. Prepreka je još dosta, ali znatno manje od navedenih pro argumenata.

Ako prakse nije moguće provesti u potpunosti, primjena 80% praksi pokazaće samo 20% rezultata [5]. Istovremeno je XP agilna metoda koja ne bi smjela biti praćena doslovno i bez odstupanja, posebno u periodu uvođenja u firmu i prelaska sa ustaljenih metoda. Dosljedna primjena ekstremnog programiranja u firmama koje zadovoljavaju preduslove za njeno uvođenje, znatno bi povećala konkurentnost balkanskih IT subjekata na međunarodnom, ali i lokalnom tržištu, te poslužila kao čvrsta karika u lancu proizvodnje softvera višeg kvaliteta.

7. REFERENCE

[1] Standish Group: CHAOS Report, http://www.standishgroup.com/sample_research, 1994.
[2] J. Koskinen: Software Maintenance Costs, http://www.cs.jyu.fi/~koskinen/smcosts.htm, 2003
[3] B. Boehm, R. Turner: Balancing Agility and Discpline: A Guide for the Perplexed, Addison-Wesley, 2004.
[4] A. Cockburn: Agile Software Development, Addison-Wesley, 2000
[5] K. Beck, C. Andres: Extreme Programming Explained:Embrace Change, 2. izdanje, Addison-Wesley, 2004
[6] W. C. Wake: Extreme Programming Explored, Addison-Wesley, 2001.
[7] L. Williams, Robert Kessler: Pair Programming Illuminated, Addison-Wesley, 2002
[8] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts: Refactoring: Improving the Design of Existing Code , Addison-Wesley, 2000.
[9] K. Beck and M. Fowler: Planning Extreme Programming, Addison-Wesley, 2000.
[10] R. Jeffries, A. Anderson, C. Hendrickson: Extreme Programming Installed, Addison-Wesley, 2000.
[11] M. Marchesi, G. Succi, D. Wells, L.Williams: Extreme Programming Perspectives, Addison-Wesley, 2002
[12] K. Auer, R. Miller: Extreme Programming Applied: Playing To Win, Addison-Wesley, 2001.
[13] Anon.: http://en.wikipedia.com i http://de.wikipedia.com, 2005.
[14] Anon.: http://www.c2.com/cgi/wiki?ExtremeProgramming, 2005.
[15] R. E. Jeffries: http://www.xprogramming.com, 2004
[16] O. Murru, R. Deias, G. Mugheddu: Assessing XP at European Internet Company, IEEE Software, 5/6 2003
[17] P. Schuh: Recovery, Redemption, and Extreme Programming, IEEE Software, 11/12 2001
[18] J. Grenning: Launching Extreme Programming at Process-Intensive Company, IEEE Software, 11/12/2001
[19] J. Rasmusson: Introducing XP into Greenfield Projects: Lessons Learned, IEEE Software, 5/6 2003
[20] R.L. Glass: Extreme Programming: The Good, the Bad and the Bottom Line, IEEE Software, 11/12 2001
[21] Anon.: What's So Extreme About Extreme Programming?, SDTimes, 2004
[22] G. Mintchell: Improve competitive quality with extreme programming, Automation World, 2003