UNIVERZITET DŽEMAL BIJEDIĆ U MOSTARU
FAKULTET INFORMACIJSKIH TEHNOLOGIJA MOSTAR

ZAVRŠNI RAD br. 0000

Agilni *software development*

(Nacrt)

Student: *Ernad Husremović, DL 2792*
Mentor: *doc. dr. Jasmin Azemović*

ver: 0.1.0

Mostar, decembar 2012.

Sadržaj

1. Uvod

Šta znači 'biti agilan'?
Agilni razvoj software-a ne predstavlja specifični proces. Agilni razvoj je način na koji se razmišlja o razvoju software-a(J.Shore i S.Warden, 2008, str. 9).
Osnovna polazišta ovog pristupa opisuje "Agilni manifest"[1], koji je definisan kroz četiri vrijednosti i 12 principa:

1.1. Vrijednosti

- Ljudi i interakcije ispred procesa i softverskih alata
- *Software* koji funkcioniše ispred detaljne dokumentacije
- Komunikacija sa klijentima ispred pregovora
- Odgovor na promjene ispred slijeđenja plana

1.2. Principi

- Glavni prioritet je zadovoljiti zahtjeve klijente kroz ranu i kontinuiranu *isporuku* software-a
- Blagonaklono prihvatiti *promjene* funkcionalnih zahtjeva, čak i u kasnijim fazama razvoja.
- Funkcionalan software treba isporučivati *često*, nakon par hefti ili mjeseci, nastojeći da taj period bude što kraći.
- Najefikasniji način razmejene informacija unutar razvojnog tima je direktna - 'face-to-face' komunikacija.
- Software koji *funkcioniše* je primarna mjera uspjeha projekta.
- Agilni procesi promoviraju održivi razvoj. Finansijeri, developeri i korisnici trebaju biti u stalnoj koordinaciji, bez obzira na dužinu trajanja projekta.

[1] `http://agilemanifesto.org/iso/en/principles.html`

- Kontinuirano pažnja na *kvalitet* tehničkih operacije i dobar dizajn povećava agilnost.
- *Jednostavnost*, kao vještina postizanja maksimalnog učinka sa što manje rada, je krucijalni agilni princip.
- Najbolja arhitektura, funkcionalni zahtjevi i dizajn se postižu u *samo-organizovanim* timovima.
- Tim redovno analizira predhodne operacije u cilju bolje efektivnosti (*refleksija*). Na osnovu tih rezultata, tim utvrđuje buduće operacije.

Nijedan proces nije savršen. Svaki pristup razvoju software-a ima prostora za unapređenje. Konačno, cilj je ukloniti svaku barijeru između razvojnog tima i uspjeha projekta, pri čemu se pristup razvoju postupno adaptira novonastalim promjenama*uslova*. To je agilnost.(J.Shore i S.Warden, 2008, str 355)
Agilni development koristi "feedback" u cilju konstantnih prilagodbi djelovanja, u visoko kolaborativnom okruženju. (V.Subramaniam i A.Hunt, 2006)

1.3. Agilne metode

Predmet detaljnijeg istraživanja su sljedeće agilne metode[2]:

- XP - *Extreme programming*
- *Scrum*

Metoda	Dužina iteracije	Veličina tima	Distribucija tima[3]
XP	1-2 hefte	2-10	Nije moguće
Scrum	2-4 hefte	1-7	Moguće

Tablica 1.1: Karakteristike pojedinih agilnih metoda

1.4. *"Done Done"* princip

Objasniti "done done" princip: Feature je done-done tek kada se može isporučiti klijentu.

[2](D.Cohen et al.)
[3]Mogućnost da članovi tima budu geografski dislocirani, da ne sjede zajedno

1.5. Tri nepobitne projektne istine

Svaki projekat karakteriše sljedeće(Rasmusson, 2012)

- Nemoguće je prikupiti sve zahtjeve klijenata na početku projekta
- Koliko god dobro prikupljni, zahtjevi klijenata će se mijenjati
- U svakom projektu će uvijek biti više posla nego li raspoloživih resursa (novca i vremena)

1.6. Organizacija i uloge unutar tima

1.6.1. XP organizacija tima

- Product menadžer
- Couch
- Klijent
- Tester
- Developer

1.6.2. Scrum organizacija tima

- Product owner (PO)
- ScrumMaster
- The team

Note: Suštinske razlike u organizaciji između XP-a i Scrum-a ne postoje.

2. Početak projekta

2.1. Planiranje

Generalno, agilni proces planiranja izbjegava detaljno ”head-on” planiranje.
Objasniti šta znači "Addaptive planning"

3. Analiza

3.1. *Stories*

Objasniti "Incremental requirements" princip.
Pragmatičan pristup analizi, analogno planiranju: analizira se do nivoa detalja koji se može postići u datom trenutku.
Definisanje detalja se vrši što je kasnije moguće, zato što tim kako projekat odmiče prikuplja sve više detalja - zato je razrada detalja kvalitetnija u kasnijim fazama projekta.

3.1.1. *Story card*

Njihova glavna osobina je da su ”customer-centric” - usmjereni na korisnika. Svaka *Story card* treba za cilj imati isporuku nove vrijednosti kljentu.

3.2. Početne iteracije

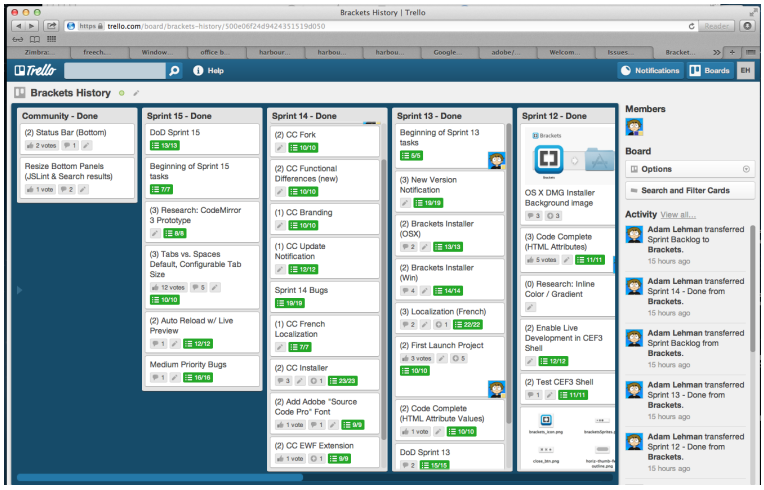
Početnim iteracijama se uspostavlja projekat, definiše toolset. U prvih par iteracija se takođe utvrđuje "velocity" tima. U prvih par iteracija "velocity" značajno varira, dok se u kasnijim fazama "velocity" stabilizira.

4. Praćenje projekta

4.0.1. Empirijska kontrola procesa

Predefinisana kontrola procesa naspram empirijske kontrole procesa(A.Tomasini i M.Kearns, 2012). Agilni pristup primjenjuje empirijsku kontrolu procesa.

4.1. *Trello* agilni projekt menadžment sistem



Slika 4.1: trello

4.2. Slack

Slack - "među-story" prostor(J.Shore i S.Warden, 2008, str. 275)

4.3. Estimating

Velocity, points, Consistent Estimates

4.3.1. Timeboxing, Iteration Timebox

4.4. Commitment

Objasniti kada tim daje "commitment" u smislu realizacije plana. Objasniti zašto je bitno da upravo tim vrši "estimating".

5. Komunikacija

5.1. Review meetings

Bez sastanaka nije moguće postići iterativnost. Stoga, je ovo obavezna praksa ASD-a.

5.2. *Issue management* (ISSUE)

Zadatak, Aktivnost, ticket
"Issue"[1] predstavlja određeni kokretni projektni zadatak:

- prijedlog za realizaciju (ideja)
- prijava programske greške (eng. bug)
- realizacija nove ili nadogradnja postojeće funkcije sistema (eng. new feature, feature upgrade)

5.2.1. Poznati alati za *issue management*

- gitlab issues[2]
- redmine issues[3]

Sve što je relevantno za projekat treba publikovati na ISSUE sitem.
Ako razvojni tim nije kolociran[4] većina komunikacija između članova obavlja se elektronskim putem.
"Issues management" sistem je po pitanju realizacije konkretnih projektnih zadataka najbolji način komunikacije.[5]

[1] developeri često koriste termin "ticket", ili "bug" (čak i kada se ne radi o programskog grešci)
[2] (Husremovič, 2012)
[3] `http://www.redmine.org`, `http://redmine.bring.out.ba`
[4] Nalazi se i djeluje na jednom mjestu, nije geografski distribuiran
[5] email kao sredstvo komunikacije u ove svrhe treba izbjegavati. Email komunikacija brzo postaje nepregledna za većinu projektnih zadataka.

Komunikacija putem ISSUE sistema time obuhvata komunikaciju na temu novih ideja i prijedloga, realizacije ili nadogradnje funkcija sistema, kao i prijavu i otklanjanje programskih ”bug”-ova (grešaka).
Parcijalni pristup (npr. ograničite se samo na ”bug”-ove) stvara prostor da bitne informacije budu nedostupne svim članovima tima.
Redovno nakon gornje preporuke slijedi pitanje:
Kako ću znati da li je nešto relevantno ili nije ?
Ako ne znaš da li je relevantno - stavi na ”issue”.
Nije nikakav problem da se na ticketima u početku nalaze suvišne informacije. Problem je kada informacije nedostaju.
Takođe je vrlo bitno da se informacije na sistem publikuju bez kašnjenje, a ne retroaktivno u formi izvještaja.
Mnoge informacije nakon dan ili dva postaju beskorisne.
Issue management treba reflektovanti život i dinamiku projekta.

5.2.2. *Issues* i novi član tima

”Issues” su sredstvo komunikacije.
Kada se novi član uključi u ekipu, redmine komentari (ili nedostatak komentara) su odličan indikator kompetencija člana.
Iz njih se vrlo brzo dođe do informacija kojim vještinama član raspolaže, koje su njegove profesionalne navike. Na osnovu tih informacija se može djelovati:

- Organizvati fokusiranu edukaciju za novog člana
- Zajednički rad sa iskusnijim kolegama itd.

6. Agilni pristup dizajnu

6.1. Inkrementalni dizajn

Izbjegava se zasebna, dugotrajna faza dizajna ("up-front").
To znači da se kôdiranju (implementaciji) pristupa bez jasnog dizajna ?!
Zagovarači agilnih metoda developmenta na ovakva pitanja daju sljedeći odgovor:

Smatrate da je dizajn zanemaren u agilnom software developmentu ? Upravo suprotno - Mi dizajn smatramo iznimno bitnim. Zato ga ***prakticiramo*** *kroz* ***sve faze*** *razvojnog procesa !*
Suštinska razlika u odnosu na klasični "up-front" je ta da se dizajniranje realizira u onoj mjeri i u onom trenutku kada je to najpogodnije i najprirodnije za razvojni tim.
Prakticiranje inkrementalnog i jednostavnog dizajn zato spriječava "overengineering"[1]

6.2. Jednostavni dizajn

6.2.1. *You Aren't Gonna Need It* (YAGNI)

YAGNI[2] princip je bazni aspekt jednostavnog dizajna[3]

[1] http://en.wikipedia.org/wiki/Overengineering
[2] Ovo vam neće trebati
[3] (J.Shore i S.Warden, 2008, str. 318)

7. Development

7.1. *Source code management* (SCM)

Detaljno u materijalu "Agilni *software development*, GIT SCM"(Husremovič, 2012)

7.2. Testiranje

Testiranje i refactoring etaljno u materijalu "Agilni *software development*, Tehnike testiranja"(Husremović, 2012b)
Objasniti zašto je TDD ujedno i testiranje i implementacija(kodiranje) i dizajn !

7.3. Stalna integracija (CI)

Detaljno u materijalu "Agilni *software development*, *Continuous Integration* (CI)"(Husremović, 2012a)

7.4. *"Jednom i samo jednom"*

"Jednom i samo jednom" (eng. "Once and Only once") princip:

> Svaki koncept izrazi jednom. (i samo jednom !).[1]

7.5. *Spike* rješenja

Spike (bos. ekser, smeč) razjašnjavaju tehička pitanja koje susrećemo, pri čemu se izbjegava kompleksnost produkcijskog kôda.(J.Shore i S.Warden, 2008, str. 334)

[1] (J.Shore i S.Warden, 2008, str. 319)

8. Release management

8.1. Testno vs produkcijsko okruženje

Detaljno u materijalu "Agilni *software development*, *Test* & *deploy* infrastruktura"(Husremović, 2012c)

9. Ostalo

9.1. ASD vodi ka haosu ?!

Striktne granice između faza dizajna, implementacije i isporuke rješenja korisniku nestaju.
Prakticiranjem TDD-a developer stalno mijenja dizajn[1] u malim koracima u toku implementacije.
Breakthrough faze obezbjeđuju veće promjene. Bitno je uočiti da se te promjene dešavaju u najbolje, najproduktivnije vrijeme - onda kada developer "sazrije" po pitanju konkretnog problema, kada dobro ovlada postojećim stanjem - njegovim ograničenjima i dobrim stranama.
Što ne trebaš (čega nema u *story*-jima) - ne implementiraj !
Nemamo potrebu anticiparti dizajnerska i arhitektonska rješenja na duge staze.
Nedostatak anticipacije će voditi "siromašnim", "kratkovidnim" rješenjima ?
ASD svakako nije pogodan za svaki tim. ASD doista može uzrokovati paralizu tima koji nije pripremljen za ASD.

9.2. Rizici i ograničenja ASD-a

Agilni *software development* pretpostavlja visoko motiviran tim, tim koji je spreman da uči.
Članovi tima moraju posjedovati bazne vještina iz oblasti koje pokrivaju.
Ako se te vještine i uvriježene prakse u slučaju programera svode na očekivanja prema vodećim članovima tima na:

> *Dajte vi meni šta trebam uraditi (izkôdirati), to je sve što me interesuje (dalje od toga nije moj posao) !*

onda ovaj tim (još) *nije spreman* za agilni development.

[1] inkrementalni dizajn TODO: referenciraj se na poglavlje

Agilni pristup jednostavno nudi i očekuje od svakog pojedinca da maksimalno doprinosi projektu u cjelini. Kolokvijalnim riječnikom, neprihvatljivo je da članovi tima probleme koji nisu direktno vezani za njihovu trenutnu aktivnost i zaduženja adresiraju kao:

To nije moj "rejon" ![2] *?*

Naravno da je i u agilnom timu neophodno poslove raporediti na najbolji način, te da je svaštarenje potrebno smanjiti na namanju moguću mjeru. Međutim, treba uvijek imati na umu da je agilni tim multi-funkcionalan, usmjeren na ciljeve projekta.
Stoga je sve što ima bitan uticaj na ciljeve i tok projekta "rejon" svakog člana.

9.3. Komptencije i vještine

Potreban (ali ne i dovoljan) uslov u primjeni agilnih metoda su odgovarajuće kompetencije i veštine članova tima. Ako one ne postoje, agilni pristup će doživjeti potpuni fjasko.
Ovdje treba naglasiti da se prvenstveno misli na bazne kompetencije i vještine, ne kompentencije koje se stiču dugogodišnjim iskustvom. Naravno, iskusniji članovi, članovi koji posjeduju višegodišnjim iskustvom iz problemskog domena projekta su bitan faktor.[3] Međutim, tim koji je spreman da uči može nadoknaditi nedostatak iskustva. Takav tim zasigurno neće moći brzo realizovati ciljeve projekta, ali će pravilnim pristupom doći do cilja.

9.3.1. Eksperti i izvršioci

U nedostatku kompetencija se pribjegava klasičnom hijerarhijskom sistemu[4] u kome unutar tima postoje jedan - dva eksperta[5], dok ostatak tima (i većinski dio tima) čine krajnji izvršioci[6]
Note: Ovdje je bitno razgraničiti da se od junior programera ili novog člana tima naravno ne može očekivati previše, ali je ključna stvar postići visoku dinamičnost njegove pozicije od početnika do člana koji daje značajan doprinos projektu.

[2] Legendarna šala iz serije "Nadrealisti"
[3] Ako projekat ima kratke rokove za realizaciju, najčešće i neophodan
[4] "vojni" sistem
[5] ekspert = "guru"
[6] izvršioc = "hamal"

9.4. Agilni software development u *Opensource* projektima

Most of the analyzed projects adapted a fixed cycle and others are in the process of transition. Fixed cycles can also be found in agile software engineering methods, such as Extreme Programming or the Scrum method. While not equal to those methods, the analyzed projects seem to benefit from such an approach. The release schedule with fixed cycles is often already fixated before development starts and each project tries to stick to it as close as possible.(Siegel, 2012, str. 95)
As shown by Mockus, Fielding and Herbsleb the role of the release manager is vital for every project.(Siegel, 2012, str. 95)

10. Zaključak

Agilni *software development* svakako nije *Silver Bullet*[1]
Međutim, on iz temelja mijenja uvriježene inžinjerske prakse. Agilni pristup unosi "životnost" u razvojni proces software-a.
U toku razvoja sofware-a[2] se od članova razvojnog tima u znak rezignacije mogu čuti konstatacije slijedećeg tipa:

> *Hah, sve bi bilo u redu da živimo u idealnom svijetu ... Da imamo dovoljno vremena i/ili developera ... Da smo prije implementacije "xyz" funkcije dobili sve potrebne informacije ...*

Agilni pristup kao temeljno polazište uzima realni svijet i realne potrebe korisnika software-a.
On razvojni tim stalno podsjeća i usmjerava da je glavni cilj sofverskog projekta ostvariti vrijednost za *korisnika software*-a.[3] Ta vrijednost (upotrebljivost, korisnost) je sama po sebi vremenski *dinamična* kategorija. Agilni *software development* nastoji razvojni ciklus usaglasiti sa tom činjenicom.
Agilni pristup zahtjeva visok nivo kompetencija unutar tima, te postojanje kulture učeće organizacije[4].

[1] `http://en.wikipedia.org/wiki/No_Silver_Bullet`

[2] ali i sveukupnog životnog ciklusa

[3] Prvi slogan moje firme je bio: *Iskoristite računar*. Međutim, slogan se nije puno koristio. Još gore, firma u svom djelovanju u mnogim segmentima odstupila od principa ovog slogana. Nakon 16 godina, rad na ovoj temi me je podsjetio na taj slogan. Iz ove perspetkive mogu konstatovati da je to bio sjajan slogan. Šteta što smo ga zaboravili - i slovom i djelom.

[4] `http://en.wikipedia.org/wiki/Learning_organization`

Literatura

A.Tomasini i M.Kearns. *Agile transition, What you Need to Know Before Starting*, 2012.

D.Cohen, M.Lindvall, i P.Costa. *Agile Software Development, (DACS State-of-the-Art/Pratice Report)*.

Ernad Husremović. *Agilni software development, Continuous Integration (CI)*, 2012a. URL `https://github.com/hernad/agile_dev_env/raw/master/agile_ci.pdf`.

Ernad Husremović. *Agilni software development, Tehnike testiranja*, 2012b. URL `https://github.com/hernad/agile_dev_env/raw/master/agile_test.pdf`.

Ernad Husremović. *Agilni software development, test & deployment infrastructure*, 2012c. URL `https://github.com/hernad/agile_dev_env/raw/master/agile_test_deploy.pdf`.

Ernad Husremovič. *Agilni software development, Git SCM*, 2012. URL `https://github.com/hernad/agile_dev_env/raw/master/agile_git.pdf`.

J.Shore i S.Warden. *The Art of Agile Development*. O'Reilly, 2008.

Jonanthan Rasmusson. *The Agile Samurai (P4.0)*. The Pragmatic Programmers, 2012.

Daniel G. Siegel. Typical development processes of free and open source software projects. Magistarski rad, Technische Universität München, Maj 2012. URL `http://www.dgsiegel.net/writing/2012_master_thesis.pdf`.

V.Subramaniam i A.Hunt. *Pratices of an Agile Developer*. The Pragmatic Programmers, 2006.

Dodatak A
Riječnik termina

– Coach - mentor razvojnog tima, onaj koji se brine o primjeni praksi, moderira tok procesa i komunikaciju unutar tima
– Velocity - sposobnost, kapacitet tima u toku iteracije

Dodatak B
Software toolset

1. Mac OS X 10.8.2
2. mvim, vim tekst editor ver 7.3
3. MacTex (TeX Live 2012)

Dodatak C
Software repozitoriji

– Agilni developerski environment - `https://github.com/hernad/agile_dev_env`