

UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

SEMINARSKI RAD

Predmet: Menadžment
Tema: Waterfall model i agilni razvoj softvera

Mentor:
Dragan Bjelica

Studenti:	
Student 1	index1
Student2	index2
Student3	index3

Beograd, oktobar 2012

SADRŽAJ

1. UVOD	3
2. WATERFALL MODEL	3
3. AGILNI RAZVOJ SOFTVERA.....	5
3.1 EKSTREMNO PROGRAMIRANJE (XP)	7
4. POREĐENJE: KOJI JE PRAVI IZBOR?	8
5. ZAKLJUČAK	9
LITERATURA	10

1. UVOD

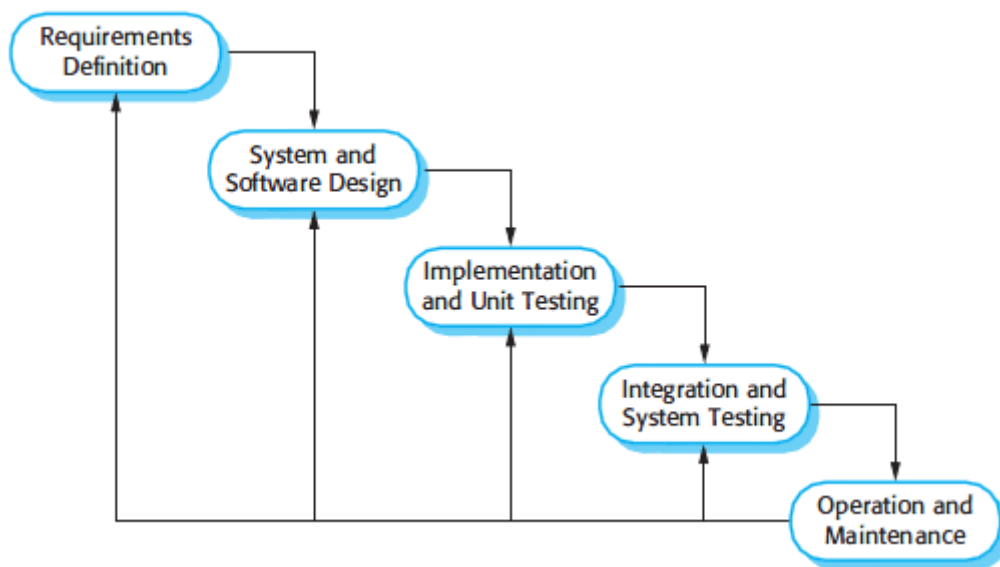
U ovom radu se govori na temu „waterfall“ (vodopad) modela i principa agilnog programiranja, trenutno dva jako popularna pristupa organizaciji tima programera. Razlike između pomenutih principa su velike, a oba imaju svoje prednosti i mane.

2. WATERFALL MODEL

Softver je vrsta proizvoda koja se tokom svog životnog veka menja, unapređuje i održava. S početkom razvoja softvera, javila se potreba da se organizuje tim programera kako bi se povećala efikasnost. Radi toga, bilo je neophodno da se precizno odredi životni ciklus programa. Tako je nastao Waterfall model. Godine 1970, dr. Winston Royce je objavio članak „*Managing the Development of Large Software Systems*“ u *Proceedings of IEEE WESCON* gde je prvi put formalno definisan ovaj model, iako Royce nije koristio termin „waterfall“ (Waterfall Model, Wikipedia).

U dotičnom članku, on kaže da „veruje u koncept ali implementacija je rizična i moguće je da ne uspe“ (Royce, 1970). Waterfall model se sastoji iz pet faza (Sommerville, 2010):

1. *Definisanje i analiza zahteva* – Konsultovanje sa korisnicima softvera, definisanje zahteva korisnika i sistema. Ova faza predstavlja osnovu za dobro osmišljen i napravljen softver.
2. *Dizajniranje sistema i softvera* – Ovaj korak podrazumeva pisanje algoritama za softver, predviđanje kako će softver raditi na sistemu, problema i načina kako da se isti eliminišu.
3. *Implementacija i testiranje* – Tokom ove faze, vrši se programiranje softvera i realizacija predhodno definisanih algoritama i zahteva putem koda. Takođe, vrši se i redovno testiranje (bilo od strane samog programera ili testera) napisanog koda kako bi se izbegli problemi kada se dođe do implementacije softvera.
4. *Integracija i sistemsko testiranje* – Softver je iskodiran i istestiran, i počinje integracija na sistem. Testiranje softvera se vrši na sistemu kako bi se uverili da sistem funkcioniše onako kako je predviđeno. Nakon što je testiranje završeno, softver se dostavlja korisniku.
5. *Funkcionisanje i održavanje* – Najčešće (ali ne i uvek), ovo je najduža faza u životnom ciklusu softvera. Softver je instaliran i koristi se. Održavanje uključuje otklanjanje grešaka koje nisu otkrivene u predhodnim fazama životnog ciklusa, poboljšavaju se performanse i slično.



Slika 1. „Životni ciklus softvera / faze razvoja u waterfall modelu“ (Sommerville, 2010)

Na papiru, ovaj model izgleda idealno. Međutim, praksa je pokazala suprotno. Softver je jako dinamičan. Programski jezici se menjaju svakih par meseci, uvode se nove mogućnosti i poboljšanja.

Da bi smo shvatili mane waterfall modela, možemo postaviti dva pitanja:

1. Šta se dešava kada zahtevi nisu u potpunosti poznati, tj. kada klijent nije do detalja siguran šta tačno želi?
2. Šta se dešava kada moramo, na klijentov zahtev, da napravimo veliku izmenu u fazi 5, kada je softver funkcionalan i radi na sistemu?

Waterfall model podrazumeva da su svi zahtevi poznati, i da klijent zna tačno šta želi od softvera. On stavlja akcenat na ranije faze razvoja, što otežava implementaciju korisničkih zahteva za izmenu softvera (Sommerville, 2010). Klijenti u većini slučajeva nisu sigurni šta žele dok ne počnu da koriste softver. Kada počnu sa korištenjem softvera, u mogućnosti su da vide njegove mane i vrline bolje od programera.

Ovo nas dovodi do drugog pitanja: šta se dešava kada klijent želi da napravi veliku izmenu u softveru u fazi 5? To je momenat kada svi postaju nezadovoljni – programer, klijent i menadžer. Programer je nezadovoljan zato što je softver isplanirao u početnim fazama, i nije uzeo u obzir da će morati da pravi veće izmene u kasnijem životnom ciklusu. Promene koje je klijent naručio su teške za realizaciju i oduzeće puno vremena, a samim tim i novca.

Klijent je nezadovoljan jer smatra da nije dobio ono šta je platio, ili u krajnjem slučaju, očekivao je da će promena koju je zatražio biti realizovana u što kraćem roku.

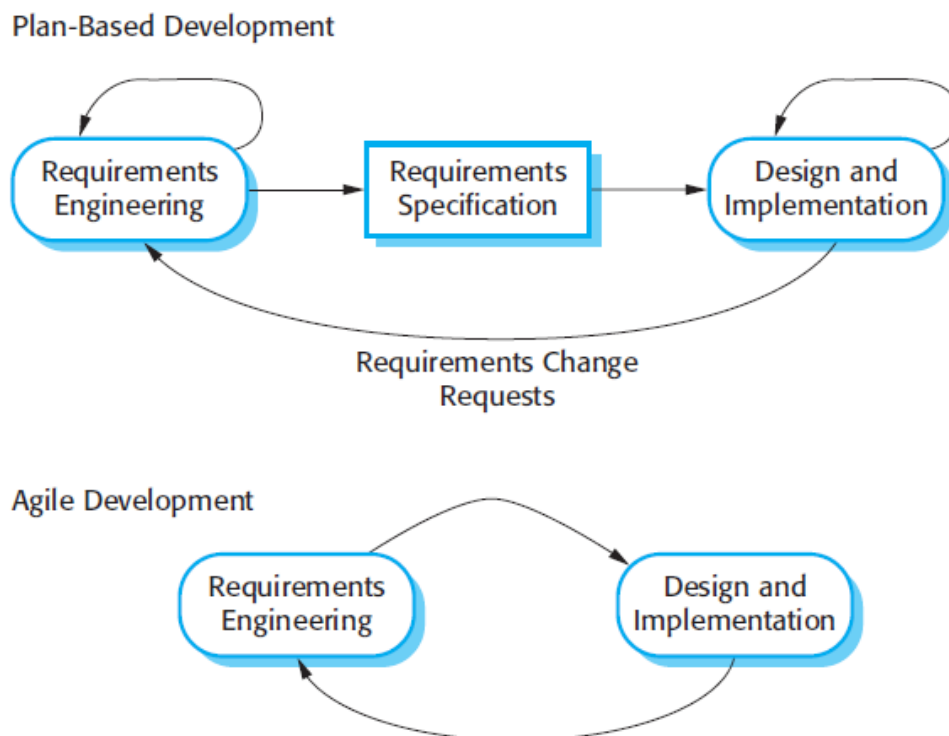
Menadžer se u ovoj situaciji nalazi u nezahvalnoj poziciji jer ne može da ispuni zahteve klijenta i programera. „Ako je vaš klijent nezadovoljan, svako je nezadovoljan. Svaki dobar softver počinje sa dobrom klijentovom idejom. Posao programera je da tu ideju ostvari. Ali pretvaranje nejasne ideje u softver koji radi i zadovoljava potrebe korisnika, je jako teško.“ (Pilone & Miles, 2008)

Međutim, waterfall model ima svojih primena i prednosti. On je konzistentan sa drugim modelima i dokumentacija se piše na kraju svake faze. Ovo čini proces razvoja vidljivim menadžerima koji imaju mogućnost da nadgledaju proces razvoja i da se pobrinu da isti teče po planu.

Kada koristiti ovaj model? Waterfall model se koristi samo kada su svi zahtevi dobro poznati i neće se bitno menjati tokom razvoja softvera. (Sommerville, 2010)

3. AGILNI RAZVOJ SOFTVERA

Okolina u kojoj se danas posluje je jako nestabilna i promenljiva. Softveri namenjeni takvoj okolini moraju da odgovaraju na potrebe koje se konstantno koriguju. Promenljiva ekonomska situacija i tržište novih, kompetentnih dobara doveli su do stvaranja agilnog modela. (Larman, 2004)



Slika 2. „Razvoj zasnovan na planu i agilni razvoj softvera“ (Sommerville, 2010)

Agilni razvoj softvera odlikuje se visokim stepenom prilagodljivosti. Veoma brzo se adaptira situaciji, čak i tokom samog razvoja softvera. On kao najvažniju komponentu razvoja ističe dizajniranje i implementaciju softvera.

Na prvo mesto, takođe, stavlja promene, kako bi postogao prilagodljivost, i jednostavnost, koja podrazumeva lišavanje svih nepotrebnih i nepraktičnih dokumentacija, kao i testiranja softvera.

Prednosti agilnog razvoja podrazumevaju:

1. Ne postoji detaljna specifikacija izgleda i funkcija sistema, a dokumentacija je svedena na minimum. Definisane su samo najvažnije karakteristike sistema.
2. Sistem se razvija kroz serije verzija. Budući korisnik sistema uključen je u proceduru- on ocenjuje svaku verziju softvera, komentariše i daje svoje predloge vezane za izmene na sistemu, kao i zahteve koje bi sledeća verzija sistema trebalo da ispunjava.
3. Nove verzije softvera se konstruišu i šalju korisniku svake dve do tri nedelje. (Larman, 2004)

Ovim putem, korisnik, iako objektivno lice, konstantno učestvuje u izradi sistema i daje programeru ideje o izmeni softvera.

Najpoznatiji metodi agilnog programiranja su Ekstremno programiranje (Extreme programming), još poznat kao XP, Skram (Scrum), Kristal (Crystal), Adaptivni softverski razvoj (Adaptive Software Development) i DSDM.

Agilne metode najuspešnije se koriste za određene tipove razvoja sistema:

1. U slučaju kada softverska kompanija radi na razvoju malog proizvoda ili proizvoda srednje veličine, koji je predviđen za prodaju.
2. Pri razvoju unutar organizacije u kojoj se ne primenjuju spoljna pravila koja bi uticala na softver i kada je korisnik spreman da se u potpunosti uključi u proces razvoja softvera.

Međutim, agilni softver ima i svojih mana. Zahvaljujući fokusiranju na male, usko povezane timove, jako je teško primeniti istu metodu na velike i razvijene sisteme. Takođe, karakteri članova tima neće se u svakoj situaciji slagati, pa će često dolaziti do razmirica unutar tima. (Sommerville, 2010)

Agilne metode daju prednost promenama i zagovaraju prilagodljivost. Međutim, svaki član tima se ne mora deliti mišljenje ostalih po pitanju važnosti određenih promena. Neke promene će pojedinac smatrati bitnijim, dok će se drugi fokusirati na različite.

Iako ideja uključivanja korisnika u proces razvoja na početku zvuči primamljivo, takođe dovodi u pitanje uspeh softvera. Korisnik ne mora biti potpuno uključen u razvoj i time može ometati celokupan proces. Isto tako, on ne mora biti validan predstavnik svih budućih korisnika. (Larman, 2004)

Još jedna mana agilnog razvoja softvera ogleda se u pisanju ugovora između kompanije koja je zatražila uslugu pravljenja softvera i firme koja ga razvija. S obzirom da su od početka zahtevi razvoja jako promenljivi, teško je sastaviti ugovor s toliko nepoznatog materijala. Zbog toga, ugovor jedino garantuje okvirno vreme razvoja sistema, a ne njegove karakteristike i dizajn. U koliko se znatno prekrši vreme, problemi prouzrokovani nepoznatim zahtevima se teško rešavaju. (Sommerville, 2010)

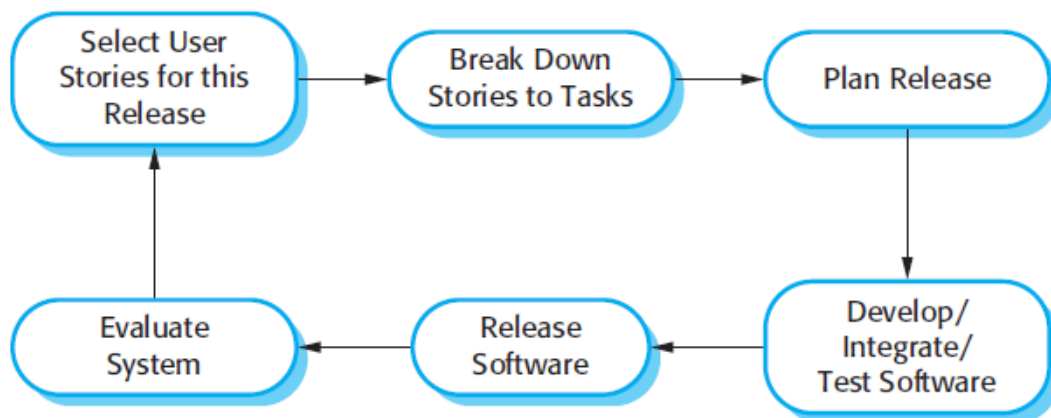
3.1 EKSTREMNO PROGRAMIRANJE (XP)

Ekstremno programiranje (XP) je tip agilnog razvoja softvera u kome veliki broj programera razvija nove verzije sistema, kombinuje ih i testira u toku jednog dana. Programeri funkcionišu u parovima ili manjim grupama i dok jedan programer radi na sistemu, drugi ga ispravlja i koriguje kodove sistema. Vremenski period između puštanja u prodaju predhodne i najnovije verzije sistema je mali, a nove verzije sistema su česte.



Slika 3. „Programiranje u paru“ (Pair Programming, Wikipedia)

Korisnik u potpunosti učestvuje u razvoju softvera, on je deo tima programera i njegov zadatak je da testira softver. Konsultuje se sa drugim članovima, diskutuje scenarije se njima i izlaže potrebe drugih korisnika. Tim u sledećoj verziji teži da zadovolji sve te potrebe. Takođe, od korisnika se očekuje da napravi prioritet između promena koje bi trebalo primeniti na sistem. Tako se najvažnija promena implementira prva itd.



Slika 4. „Grafički prikaz ekstremnog programiranja“ (Sommerville, 2010)

U praksi, mnoge kompanije ne koriste u potpunosti sve funkcije ekstremnog programiranja, već biraju one koje im najviše odgovaraju. (Sommerville, 2010)

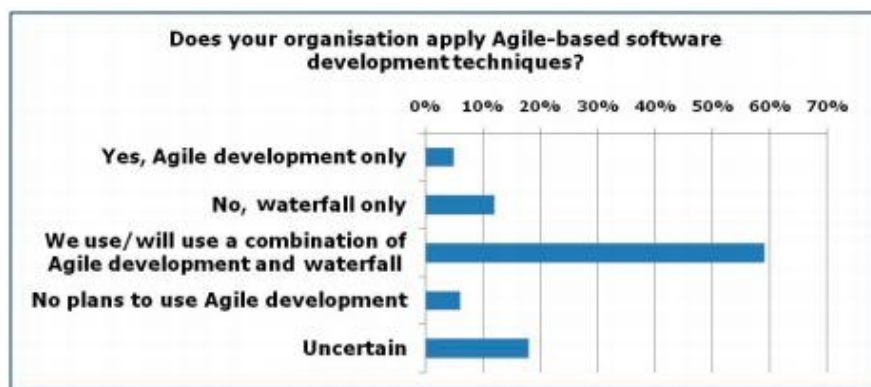
4. POREĐENJE: KOJI JE PRAVI IZBOR?

Poređenje ova dva pristupa najlakše je izvršiti kada izdefinišemo njihove osnovne karakteristike (prednosti) a nakon toga mane.

Waterfall:	Agile:
• Očekivanja su unapred definisana, mogućnost planiranja	• Konstantna dostava proizvoda i uzimanje povratnih informacija (iterativni i inkrementalni razvoj)
• Zahtevi moraju biti realizovani pre ulaska u sledeću fazu.	• Promena zahteva tokom razvoja je dobrodošla.
• Dok se proizvod na završi, korisnik može da se fokusira na druge stvari.	• Rano testiranje koda i konstanta integracija i menjanje.
• „Dva puta meri, jednom seci“ – manja šansa da softver mora da se menja.	• Razmenjivanje mišljenja sa korisnikom i čekanje njegovog odobrenja nakon svake implementirane mogućnosti softvera.

Tabela 1. „Prednosti svakog modela“ (Sliger, 2007)

Na osnovu Tabele 1, možemo zaključiti da su oba pristupa korisna u praksi, ali za različit tip projekata. O manama oba pristupa, bilo je reči u predhodnim poglavljima. Sada, problem se svodi na pitanje – koji pristup odabrati?



Slika 5. „Da li vaša organizacija primenjuje tehnike razvoja softvera bazirane na Agilnom principu?“ (CA Technologies, 2010)

Pitanje „Da li vaša organizacija primenjuje tehnike razvoja softvera bazirane na Agilnom principu?“ postavila je kompanija za CA Technologies koja se bavi IT menadžmentom. Na pitanje, većina ispitanih kompanija je odgovorila da koriste kombinaciju agilnog razvoja i waterfall modela - čak 59% (CA Technologies, 2010).

Nijedan od ova dva modela nije savršen, i ne može da se uklopi u svaku situaciju i svaku kompaniju. Većina firmi, kao što možemo primetiti, bira kombinaciju oba pristupa.

Na primer, mnoge kompanije prave softver za internu upotrebu, koji koriste za upravljanje projektima, kao razvojno okruženje za pravljenje svog softvera i tome slično. Primer ovakvog jednog alata bi bio „WoWEdit“ korporacije Blizzard. (BattleNet Forums, 2011)

Kod odabira modela koji ćemo koristiti, primenjuje se osnovna logika. Postavljamo sebi pitanja:

- Ko je klijent? Da li je to naša firma ili neko drugi?
- Da li želimo da klijenta uključimo u sam proces razvoja?
- Da li će se projekat bitno menjati tokom vremena?
- Da li klijent zna tačno šta želi?

Odgovor na ova pitanja daje predlog koji model da koristimo. Ako pravimo softver koji će da procesira sve transakcije jedne banke, koristićemo waterfall model. Za ovakav sistem pretpostavljamo da se neće bitno promeniti tokom vremena (on će pratiti sve transakcije i eventualno neku statistiku – koliko će biti uplata, isplata i slično; što ne ostavlja mogućnost za velike izmene). On mora da bude bez grešaka i jako pouzdan. Klijent (banka) u ovom slučaju neće imati potrebu da se detaljno uključi u razvoj softvera, ali zato daje sve specifikacije koje su potrebne kako bi se napravio softver kakav oni žele.

Sa druge strane, ako pretpostavimo da pravimo web sajt za manju kompaniju, pristup će se bitno promeniti. Klijent je mala kompanija, koja nema mnogo resursa i želi da dobije ono što je platila (često i više). Ona ima vremena da se uključi u razvoj sajta, jer joj je to bitna stavka u marketingu, i konsantno može da nam da informacije da li joj nešta odgovara ili ne. Web sajt je softver koji se stalno menja i napreduje u skladu sa trendovima. Od početka razvoja, pa do kraja, korisnik će dovoljno puta promeniti mišljenje tako da sajt uopšte neće ličiti na onaj iz prvobitne ideje. U ovom slučaju, koristi se agile pristup, gde će korisnik biti uključen u svaki aspekt razvoja sajta, kako na kraju razvoja, kada je projekat završen, korisnik ne bi bio nezadovoljan i kako se ne bi menjao veći deo koda.

5. ZAKLJUČAK

Informacione tehnologije se menjaju iz dana u dan, a sa njima i menadžment u IT-u. Na izazove organizovanja tima programera, ne postoji jednostavan odgovor i nije moguće odabrati samo jednu predefinisanu opciju. Dok waterfall model može da obezbedi veliku pouzdanost, lako definisanje rokova, i „dva puta meri jednom seci“ pristup, agilni model je prikladniji za jednostavnije i manje projekte koji se konstanto menjaju.

U dinamičnom svetu IT menadžmenta, na menadžerima je odluka šta je najprikladnije za dati projekat i tim.

LITERATURA

1. BattleNet Forums. (2011). *What does Blizzard use*. Retrieved October 2012, 29 from Forums - World of Warcraft: <http://us.battle.net/wow/en/forum/topic/2456451706#4>
2. CA Tehnologies. (2010, October). *A survey of portfolio management for agile IT*. Retrieved October 16, 2012 from CA Technologies: <http://www.ca.com/~media/files/supportingpieces/ppm-summit-agile-research-oct-2010.aspx>
3. Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition*. Addison Wesley Professional.
4. Pair Programming, Wikipedia. (n.d.). *Pair Programming*. Retrieved October 29, 2012 from Wikipedia: http://en.wikipedia.org/wiki/Pair_programming
5. Pilone, D., & Miles, R. (2008). *Head First Software Development*. O'Reilly Media.
6. Royce, W. (1970). Managing the Development of Large Software Systems. *IEEE Wescon*, (pp. 328-338).
7. Sliger, M. (2007, April 20). *Agile vs. Waterfall - Which is the Ethical Choice?* Retrieved October 29, 2012 from Sliger Consulting: <http://140.99.29.241/images/uploads/AgilevsWaterfall.pdf>
8. Sommerville, I. (2010). *Software Engineering (9th Edition)*. Addison Wesley Professional.
9. Waterfall Model, Wikipedia. (n.d.). *Waterfall model*. Retrieved October 16, 2012 from Wikipedia: http://en.wikipedia.org/wiki/Waterfall_model