



Funded by the
Erasmus+ Programme
of the European Union



This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Agilni razvoj softvera

Lekcija 09

PRIRUČNIK ZA STUDENTE

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Lekcija 09

AGILNI RAZVOJ SOFTVERA

- ✓ Agilni razvoj softvera
- ✓ Poglavlje 1: Nastanak agilnih metoda razvoja
- ✓ Poglavlje 2: Agilne metode
- ✓ Poglavlje 3: Agilni razvoj i planom vođen razvoj
- ✓ Poglavlje 4: Ekstremno programiranje
- ✓ Poglavlje 5: Primena agilnih metoda u većim projektima
- ✓ Poglavlje 6: Pokazni primeri
- ✓ Poglavlje 7: Agilefant softverski alat
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

✓ Uvod

UVOD

Nastavni cilj

Cilj ove Inastavne jedinice je da vas uvede u metode agilnog razvoja softvera (ili agilne metode), s ciljem da::

- razumete racionalnost agilnih metoda razvoja softvera, tzv. agilni manifest, i razliku između agilnog i planom-vodenim razvojem softvera;
- znate ključnu praksu u primenu ekstremnog programiranja i kakav je njihov odnos u odnosu na opšte principe agilnih metoda,
- razumete Scrum-ov pristup u upravljanje agilnim projektima;
- da budete svesni problema koji se javljaju pri primeni agilnih metoda pri razvoju velikih softverskih sistema.

▼ Poglavlje 1

Nastanak agilnih metoda razvoja

ZAŠTO JE DOŠLO DA RAZVOJA AGILNIH METODA RAZVOJA SOFTVERA?

Trka za vremenom zahvatila je i industriju razvoja softvera. S ciljem da se brzo dobije koristan softver, on se ne dobija kao jedna jedinstvena jedinica, već se dobija u seriji inkremenata.

Poslovanje firmi se sve više odvija u okruženju koje se brzo menja. Tržište postaje sve zahtevnije, konkurenčija sve veća, svi traže nešto bolje, a za manje novca, a naročito, svi žele da se nešto uradi što brže. Firme moraju da reaguju na te promene u što kraćem vremenu. Ko ne može brzo da reaguje, vrlo često gubi trku i nestaje sa globalnog tržišta, pa i iz poslovanja.

Ta trka za vremenom zahvatila je i industriju razvoja softvera. Klasičan postupak razvoja softvera obuhvata specifikaciju zahteve, projektovanje, pa razvoj softvera, a na kraju testiranje i isporuku softvera. Sve to je praćeno odgovarajućom dokumentacijom. Takva metodologija razvoja softvera je odgovarajuća za velika, a naročito tzv. kritične sisteme, koji moraju da rade vrlo pouzdano i rade u stabilnim uslovima poslovanja. Međutim, ova metodologija, zato što zahteve dosta vremena za razvoj, pa i dokumentovanje softvera, nije odgovarajuće za razvoj softvera namenjen poslovanju firmi, tj. pri razvoju tzv. poslovnih aplikacija. Kao što je rečeno, poslovno okruženje se brzo menja, te se i zahtevi koje softver treba da ispuni, a kojji su definisani danas, već za nekoliko meseci možda neće biti ispravni, jer su se uslovi u poslovnom okruženju promenile. Pa kako onda razviti softver, kada kupac stalno menja zahteve?

Krajem 90-tih godina, postavljene su osnove tzv. brzog razvoja softvera s ciljem da se brzo dobije koristan softver. Softver se ne dobija kao jedna jedinstvena jedinica, već se dobija u seriji inkremenata, pri čemu svaki softverski inkrement sadrži neku novu funkcionalnost sistema. Na tom principu, razvijeno je više metoda razvoja softvera koje dele sledeća zajednička svojstva:

1. **Procesi pripreme specifikacije, projektovanja i implementacije su izmešani.**
Ne postoji detaljna specifikacija, a projektna dokumentacija je minimizirana ili je automatski generisana softverskim alatima za razvoj softvera. Dokument o zahtevima korisnika definiše samo najvažnije karakteristike sistema.
2. **Sistem se razvija u vidu serije verzija.** Korisnici i druge zainteresovane strane definišu zahteve za svaku verziju. Oni predlažu promene softvera i nove zahteve, koje se onda primenjuju u kasnijim verzijama softvera.
3. **Korisnički interfejsi sistema se razvijaju primenom interaktivnih sistema** za njihov razvoj koji omogućavaju brzo kreiranje grafičkog interfejsa i raspoređivanje

ikona i drugih elemenata interfejsa. Ti sistemi generišu interfejse za prikazivanje na veb pretraživačima ili za posebne platforme kao što su MS Windows ili Mac.

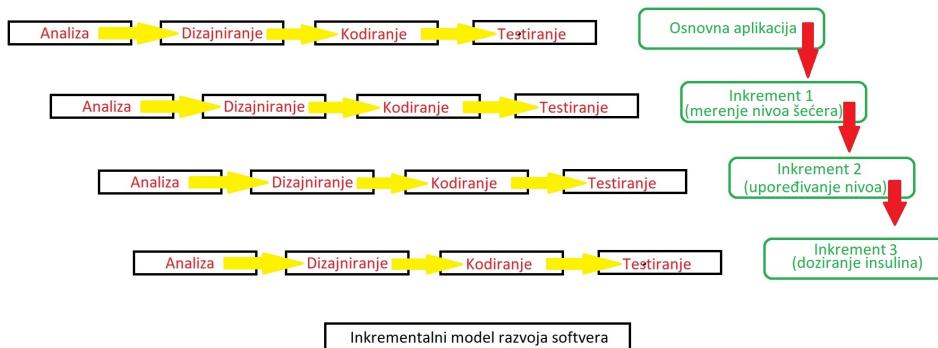
INKREMENTALNI RAZVOJ SOFTVERA

Agilne metode realizuju koncept integralnog razvoja softvera, u kome su inkrementi mali, i isporučuju se korisnicima na svake dve ili tri nedelje

Ove metode tzv. inkrementalnog razvoja softvera postale su popularne najviše u razvoju poslovnih softverskih sistema. Agilne metode realizuju koncept integralnog razvoja softvera, u kome su inkrementi mali, i isporučuju se korisnicima na svake dve ili tri nedelje. One podrazumevaju uključivanje kupaca u razvojni proces radi dobijanja brze njihove reakcije na promenu zahteva. Kod primene agilnih metoda koristi se minimalna dokumentacija, jer se dominantno koristi neformalna komunikacija učesnika u procesu razvoja softvera, umesto formalnih sastanaka i pisane dokumentacije

PRIMER INKREMENTALNOG RAZVOJA SOFTVERA

U primeru je dat inkrementalni model razvoja softvera sistema za upravljanje insulin pumpom.



Slika 1.1 Inkrementalni model razvoja softvera

Kako bi se što bolje prikazao model inkrementalnog razvoja softvera za primer je uzet softver koji se koristi u medicinske svrhe a to je softver za upravljanje insulin pumpom. Za početak potrebno je detaljno analizirati zahteve samih korisnika. U slučaju na slici 1. to bi bili zahtevi medicinskih radnika i bolesnika. Nakon toga se vrši dizajniranje dijagrama i pravljenje dokumenata koji bi pomogli samim programerima da što bolje shvate zahteve korisnika, takođe ovi dokumenti i dijagrami bi imali za cilj detaljno objašnjenje šta se očekuje od ovog dela aplikacije odnosno inkrementa i koji cilj bi inkrement trebao da ispunji. Jedan od glavnih

delova ovog modela jeste samo kodiranje i pravljenje dela aplikacije, nakon čega se vrši testiranje samog dela koda koji je kodiran. Kada se završi testiranje i ustanovi se da je deo aplikacije spreman i siguran za dalje korišćenje, on se priključuje aplikaciji i isporučuje krajnjem korisniku. Na slici 1. se može videti model koji se sastoji od 3 inkrementa, prvi se odnosi deo aplikacije koji će preko senzora da vrši merenje nivoa šećera u krvi i da ga smešta u bazu podataka, drugi inkrement bi treba da vrši upoređivanje izmerenog nivoa sa granicama koje su definisane a nakon toga bi treći inkrement trebao da deluje i dozira određenu količinu insulina kako bi se nivo šećera doveo u normalne granice.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju inkrementalni razvoj softvera.

- 1. Zadatak:** Modelovati inkrementalni model razvoja softvera za proizvoljnu aplikaciju. Opisati faze inkrementalnog razvoja i navesti konkretne aktivnosti u svakoj fazi za odabranu aplikaciju.
- 2. Zadatak:** Izvršiti izmenu korisničkih zahteva za odabranu aplikaciju u prvom zadatku. Izvršiti izmenu inkrementalnog modela razvoja softvera. Navesti sve izmene u procesu razvoja. Da li je izmena zahteva uticala na kompletan razvojni proces?

▼ Poglavlje 2

Agilne metode

PRINCIPI AGILNIH METODA

Agilne metode se filozofski oslanjaju na tzv. „agilni manifest“ koji je dokument oko koga su se složili mnogi proizvođači softvera.

Agilne metode primenjuju inkrementalni razvoj specifikacije, projektnog rešenja (dizajna), koda, testiranja i isporuke softvera. One eliminišu procesnu birokratiju, jer se usmeravaju ka razvoju softvera, a ne dokumentacije. Agilne metode se filozofski oslanjaju na tzv. „agilni manifest“ koji je dokument oko koga su se složili mnogi proizvođači softvera. Manifest definiše sledeće:

“Mi otkrivamo bolje načine razvoja softvera primenjujući ih i pomažući drugima da to isto urade. Radeći, mi smo došli do sledećih vrednosnih stavova:

- *Pojedinci i interakcije u odnosu na procese i alate*
- *Softver u radu u odnosu na odgovarajuću dokumentaciju*
- *Saradnja sa kupcem umesto pregovori oko ugovora*
- *Reakcija na promene umesto realizacije plana*

Ovde, u odnosu na vrednosti na desnoj strani, mi vrednujemo više vrednosti na levoj strani ovih iskaza.”

Na osnovu ovih principa, razvijeno više različitih agilnih metoda (ekstremno programiranje, Scrum, Crystal, i dr.). Došlo je i do njihove integracije sa tradicionalnim metodama koje se oslanjaju na modelovanje (agilno modelovanje, agilni RUP). Svi ovi metodi dele iste principe prikazane u prikazanoj tabeli (slik 1).:

Tabela: Principi agilnih metoda

Princip	Opis
Aktivna uloga kupca	Kupci treba da budu uključeni u proces razvoja. Oni treba da obezbede nove sistemske zahteve i da odrede prioritete, kao i da ocene iteracije sistema.
Inkrementalna isporuka	Softver se isporučuje u inkrementima za koje kupci određuju zahteve koje treba da zadovolji svaki inkrement.
Ljudi a ne proces	Veštine razvojnog tima treba da budu prepoznate i iskorišćene. Članovima time treba prepustiti da svoj softver razviju na sopstven način bez unapred obavezujućih procesa.
Zagrlji promene	Očekujte da se sistemski zahtevi menjaju, te i projektno rešenje sistema mora da se prilagođava tim promenama.
Održi jednostavnost	Usmerite se ka jednostavnosti i u softveru koji se razvija i u razvojnom procesu. Svuda gde je moguće, eliminisište složenost iz sistema.

Različite agilne metoda na različit način primenjuju ove principe. Najčešće se koriste dve: ekstremno programiranje i Scrum.

PRIMENA AGILNIH METODA

Zbog odsustva jasnog i kompletног dokumenta koji definiše zahteve, što je obično deo ugovore, softverske kuće imaju poteškoće da angažuju druge firme kao podizvođače.

Agilne metode uspešno se primenjuju pri razvoju softverskih sistema sledećeg tipa:

1. Razvoj proizvoda koji softverska kompanija razvija za prodaju, a primenom projekta male ili srednje veličine.
2. Razvoj sistema za kupca, pri čemu je kupac jasno opredeljen da aktivno učestvuje u razvoju i gde nema mnogo spoljnih pravila i ograničenja koji se odnose na softver.

Nije uvek lako u praksi sprovesti principe koje koriste agilne metode:

1. Zbog svojih drugih obaveza, predstavnici kupca softvera nisu uvek u mogućnosti da posvete dovoljno vremena radu sa razvojnim timom.
 2. Ponekad, članovi tima nemaju kapacitet da mogu intenzivno da rade, što je uobičajeno kod primene agilnih metoda, te ne ostvaruju dobru saradnju sa drugim članovima tima.
 3. Određivanje prioriteta u uvođenju promera nije jednostavno, jer različite zainteresovane strane imaju svoje, međusobno suprotstavljene prioritete.
 4. Ostvarivanje i održavanje jednostavnosti sistema zahteva dodatan napor i rad. Pod pritiskom rokova, razvojni tim nema uvek dovoljno vremena da traži najjednostavnija rešenja.
 5. Mnoge organizacije, a naročito vremena, uložile su velike napore i godine da bi navikli zaposlene da poštuju postavljenje procese. Sada im je teško da ih usmere da ne slede formalne, već neformalne procese definisane od strane samog razvojnog tima.
- Zbog odsustva jasnog i kompletног dokumenta koji definiše zahteve, što je obično deo ugovore, softverske kuće imaju poteškoće da angažuju druge firme kao podizvođače, u slučajevima kada primenjuju agilne metode. Zbog toga se u takvim slučajevima ugovara vremensko angažovanje ocenjeno da je neophodno za završetak nekog posla. Međutim, ako se u očekivanim rokovima ne završi taj posao, teško je onda proceniti ko je za to kriv, i na čiji račun treba da ide dodatan rad za završetak posla.

PRIMENA AGILNIH METODA U ODRŽAVANJU

Uključivanje korisnika sistema i česte promene u sastavu razvojnog tima – su dva najveća problema u primeni agilnih metoda u održavanju sistema.

Pitanje je logično, jer se razvoj odvija sa minimumom formalne dokumentacije, a ona je osnov za ljudе koji dobijaju zadatku da promene nešto u sistemu, a nisu na njemu radili. Međutim, poznato je da u praksi, i kada se koristi formalna dokumentacija, ona najčešće nije ažurna, tj. osvežena promenama koje su u međuvremenu vršene. Promoteri agilnih metoda zato tvrde da nije važna formalna dokumentacija i da na njoj ne treba gubiti vreme, već je važno napraviti visoko kvalitetan, dobro strukturisan kod, koji je čitljiv i jednostavan za održavanje. Oni smatraju da se napor treba usmeriti ka dobijanju takvog koda, a ne ka dokumentaciji. Smatraju da nedostatak dokumentacije, ne bi trebalo da bude problem u održavanju koda, ako je on dobro napravljen. Međutim, ipak se smatra da bez dobrog dokumenta koji definiše zahteve, teško je oceniti posledice predloženih promena u sistemu. Zato, primena agilnih metoda u održavanju softvera ipak otežava njegovo kasnije održavanje i čini ga skupljim.

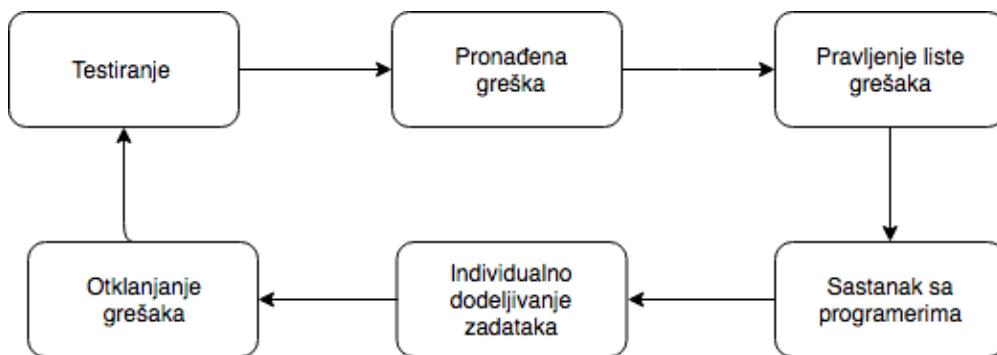
Praksa je ipak pokazala, da se agilni metodi mogu uspešno primeniti i u održavanju, jer inkrementalne isporuke popravki postojećeg sistema je prirodan način evolucije softvera. Glavni je problem uključiti korisnika sistema u rad na popravkama sistema, i u fazi evolucije softvera.

Drugi problem kod primene agilnih metoda u održavanju je u promenama razvojnog tima. Ako tih ljudi više nema, sa njima je otišlo i njihovo iskustveno znanje i poznavanje sistema, a novi ljudi, u odsustvu dokumentacije, imaju problem da se snađu.

Ima predloga da se koriste hibridne metoda, koje spajaju najbolje strane agilnih metoda i planom-vođenih metoda razvoja softvera.

PRIMER AGILNIH METODA U ODRŽAVANJU

Primer obuhvata primenu agilnih metoda u održavanju softvera na otklanjanju greške u toku izrade aplikacije.



Slika 2.1 Primer dijagrama za agilno otklanjanje grešaka pri izradi aplikacije

Kako bi se na što brži način otklonila greška u radu softvera, potrebno je što više skratiti proceduru i dokumentaciju oko detektovanja greške u softveru. Za početak potrebno je testirati aplikaciju i ukoliko se javi greška pri korišćenju ona se samo unosi u listu grešaka, kako se ne bi gubilo vreme za otklanjanje pojedinačnih grešaka, jer postoji mogućnost da su greške međusobno povezane i utiču jedna na drugu. Nakon napravljenje liste potrebno je napraviti sastanak na kome bi prisustvovali svi učesnici agilnog razvoja softvera kako bi bili upoznati sa greškama koje softver ima. Na sastanku se odlučuje koja greška se prva otklanja i

kome se zadatak otklanjanja dodeljuje. Kada se izvrši otklanjanje greške ili problema u sistemu, potrebno je ponovno prolaženje kroz aplikaciju i ponovno detektovanje svih grešaka. Ovaj proces se ponavlja u toku celog razvoja softvera.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju primenu agilnih metoda u razvoju softvera.

- 1. Zadatak:** Na proizvoljnu aplikaciju primeniti agilnu metodu u održavanju. Simulirati uključivanje korisnika u fazu održavanja. Identifikovati fazu u kojoj je moguće uključiti korisnika.
- 2. Zadatak:** Prikazati proces otklanjanja grešaka u fazi održavanja na proizvoljnem sistemu primenom agilnih metoda.

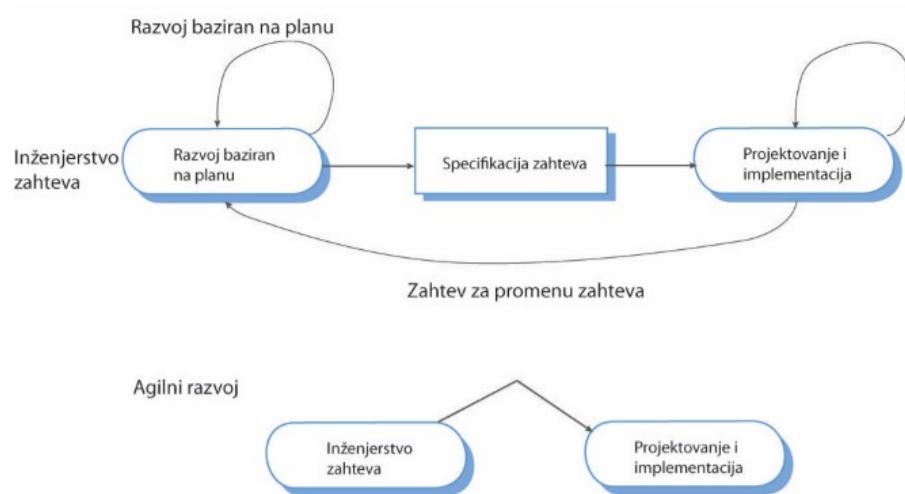
✓ Poglavlje 3

Agilni razvoj i planom vođen razvoj

UPOREĐENJE PLANIRANOG I AGILNOG METODA RAZVOJA

Kod planski vođenog razvoja, iteracije se javljaju unutar aktivnosti sa formalnom dokumentacijom. Kod agilnog pristupa, iteracije se dešavaju između aktivnosti.

Kod razvoja softvera vođenim planom, obuhvaćene su sve aktivnosti kao posebne aktivnosti, te izlazi iz jedne, predstavljaju ulaz u naredne, počev od prikupljanja zahteva, pa do testiranja. Kod agilnog razvoja, projektovanje i implementacija su centralne aktivnosti softverskog procesa, i one obuhvataju i druge aktivnosti, kao što su prikupljanje zahteva i testiranje. Slika 1 prikazuje te razlike između dva pristupa.



Slika 3.1 Planski vođen razvoj i agilni razvoj softvera

Kod planski vođenog razvoja, iteracije se javljaju unutar aktivnosti sa formalnom dokumentacijom koja se koristi za komunikaciju između faza procesa. Na primer, na osnovu prikupljenih zahteva, kreira se specifikacija zahteva, a na osnovu nje, vrši se projektovanje i implementacija softvera. **Kod agilnog pristupa,** iteracije se dešavaju između aktivnosti, te se definisanje zahteva i projektovanje realizuju zajedno, a ne posebno, kao kod planom vođenog razvoja.

Međutim, oba pristupa mogu da primenjuju metode koje su osnova drugog pristupa, te tako nastaju **hibridne metode**. Na primer, planom vođen proces može da primenjuje i inkrementalni razvoj i isporuku. Na taj način, se dodeljivanje zahteva, i planiranje faze

projektovanja i razvoja, realizuju preko serije inkremenata. Isto tako, nije nemoguće da agilni proces, koji je po prirodi usmeren na programiranje, proizvodi i neku projektnu dokumentaciju

ASPEKTI KOJI UTIČU NA IZBOR METODA RAZVOJA

Najveći broj projekata razvoja softvera koriste praksu i planom vođenim i agilnog pristupa razvoju softvera.

Ustvari, najveći broj projekata razvoja softvera koriste praksu i planom vođenim i agilnog pristupa razvoju softvera. Pri određivanja ravnoteže u primeni metoda i jednog i drugog pristupa, imaju se u vidu i sledeći tehnički, ljudski i organizacioni aspekti:

1. *Da li važno da se ima detaljna specifikacija i projektno rešenje pre nego što se počne sa implementacijom?* Ako je to tako, onda treba primeniti planom vođeni pristup razvoju.
2. *Da li je ostvarljivo da primenite inkrementalnu isporuku softvera kupcu, i da li od njega očekujete brz odgovor?* Ako je to tako, onda možete koristiti agilne metode.
3. *Koliko je veliki sistem koji se razvija?* Agilne metode su najefektivnije kada se sistem razvija sa malim timom koji mogu da neformalno komuniciraju. To nije moguće u slučaju velikih sistema koji zahtevaju veliki razvojni tim, tako da je u tom slučaju plom vođen pristup izgledniji.
4. *Koji se tip sistema razvija?* Sistemi koji zahtevaju puno analiza pre implementacije (npr. sistemi u realnom vremenu sa složenim vremenskim zahtevima) obično imaju potrebu za vrlo detaljnim projektovanjem radi izvršenja ove analize. U takvom slučaju, planom vođen pristup je najbolji.
5. *Šta je očekivani životni ciklus sistema?* Dugoživeći sistemi obično zahtevaju više projektne dokumentacije radi komunikacije sa originalnim namerama tima koji je razvio sistem i tima koji ga održava. Međutim, tim za održavanje često ima primedbe da je ta dokumentacija neažurna i da zbog toga nije od velike koristi za održavanje sistema.
6. *Koje su raspoložive tehnologije za podršku razvoja sistema?* Agilne metode često zavise od dobrih alata za održavanje verzija projektnog rešenja softvera. Ako se ne koristi dobar alat (IDE) za vizualizaciju programa i njegovu analizu, onda je potrebna obimnija projektna dokumentacija.
7. *Kako je projektni tim organizovan?* Ako su članovi tima na različitim lokacijama, ili ako u razvoju učestvuje i neka druga organizacija, onda je najčešće neophodno izraditi projektnu dokumentaciju radi komunikacije između udaljenih razvojnih grupa i pojedinaca. To se mora unapred planirati.
8. *Da li postoje kulturološki aspekti koji mogu da utiču na razvoj sistema?* Tradicionalne inženjerske organizacije obično imaju kulturu negovanja planom vođenog razvoja, jer je to norma u inženjerstvu. To obično zahteva obimnu dokumentaciju, što nije slučaj kod agilnih procesa kada se upotrebljava neformalno znanje.

ASPEKTI KOJI UTIČU NA IZBOR METODA RAZVOJA (NASTAVAK)

Pri izboru metodologije razvoja softvera treba uzeti u obzir niz faktora projekta

9. Koliko su dobri projektanti i programeri razvojnog tima? Obično primena agilnih metoda zahteva ljude većih sposobnosti nego primena plnom vođenih pristupa. Ako to nije slučaj, onda se najbolji rudi koriste za projektovanje, a ostali , za programiranje.

10. Da li je sistem zavistan od spoljne regulacije (zakona, propisa, standarda)? Ako sistem mora da bude prihvaćen od neke regulatorne agencije, onda je najčešće potrebno da se podnese detaljna dokumentacija sistema, radi provere njegove sigurnosti, odnosno, zadovoljenja pravila.

U praksi, mnoge firme koje tvrde da primenjuju agilne metode u razvoju, primenjuju samo neke elemente agilne prakse koju integrišu u njihove planom vođene procese

PRIMER AGILNOG RAZVOJA SOFTVERA

Primer prikazuje prednosti agilnog razvoja softvera u odnosu na standardne metode razvoja.

Agilna metoda razvoja softvera	Standarde metode razvoja softvera
Arhitektura je inkrementalna i postoji mogućnost naknadnog dodavanja novih funkcionalnosti	Arhitektura je dokumentovana i mora biti detaljno popunjena i analizirana pre početka samog kodiranja
Svaki individualni programer je odgovoran za kod cele aplikacije	Programer je odgovoran za samo jedan deo koda na kome je sam radio
Mogućnost unapređivanja aplikacije u svakom trenutku	Unapređivanje se radi nakon završetka softvera
Fokusirana na završetak manjih delova koda (istorija) u što kraćem roku	Fokusirana na završetak celih modula (koji se sastoje od više istorija)
Jednostavnija dokumentacija	Komplikovana dokumentacija i svaki deo aplikacije je detaljno analiziran
Glavnu ulogu u razvoju ima: Programer	Glavnu ulogu u razvoju ima: Arhitekta softvera, Programer
Postepeno puštanje delova aplikacije u rad	Tek nakon završetka projekta, aplikacije se pušta u rad

Slika 3.2 Razlike između Agilne metode i standarnih metoda za razvoj softvera

Na slici 3. se mogu videti razlike između agilne metode razvoja softvera i standardnih metoda. Agilna metoda razvoja softvera je dosta jednostavnija, nije potrebna komplikovana dokumentacija, izrada softvera se odmah započinje i nema dodatnog gubljenja vremena na planiranje i analizu. Dodavanje funkcionalnosti se može izvršiti bez planiranja i nije potrebno završiti projekat kako bi se dodavale nove funkcije kao kod standardnih metoda. Ušteda vremena na razvoju softvera agilnom metodom ubrzavanje procesa unutar softvera koji će

regulisati nivo šećera u krvi a samim tim bolje korisničko iskustvo. Agilnom metodom bi se sam softver dosta brže razvijao i testirao, a nove funkcionalnosti bi mogle da se dodaju velikom brzinom u slučaju nekih dodatnih potreba pacijenta.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju primenu agilnih metoda razvoja softvera.

1. Zadatak: Za proizvoljni sistem osmisliti planski vođen razvoj softvera i agilni način razvoja softvera. Predstaviti korake u razvoju i uporediti jedan i drugi način. Navesti koji način razvoja softvera više odgovara odabranom sistemu.

2. Zadatak: Podeliti agilni metod razvoj softvera na manje delove proizvoljne aplikacije. Da li je moguće te delove razviti uz pomoć planski vođenog razvoja softvera?

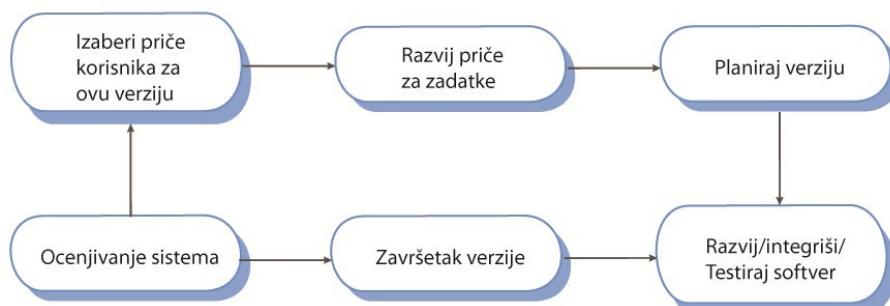
✓ Poglavlje 4

Ekstremno programiranje

PRINCIPI PRIMENJENI KOD EKSTREMNOG PROGRAMIRANJA

Kod ekstremnog programiranja, zahtevi se izražavaju u vidu scenarija a programeri rade u parovima i razvijaju testove za svaki zadatak pre nego što napišu kod.

Ekstremno programiranje (XP) je najčešće primenjivan i najbolje poznat agilni metod. Kod ekstremnog programiranja, zahtevi se izražavaju u vidu scenarija (koji se nazivaju korisničkim pričama) koji se direktno primenjuju u vidu serije zadataka. Programeri rade u parovima i razvijaju testove za svaki zadatak pre nego što napišu kod. Ovi testovi moraju da se uspešno izvrše kada je novi kod integrisan sa sistemom. Postoji vrlo kratko vreme između susednih verzija sistema. Slika 1 ilustruje XP proces koji proizvodi jedan inkrement sistema koji je u razvoju.



Slika 4.1 Ciklus ekstremnog programiranja

Ekstremno programiranje odražava sledeće principe agilnih metoda:

1. *Inkrementalni razvoje je podržan preko malih, ali čestih verzija sistema.* Zahtevi nastaju iz jednostavnih priča korisnika ili scenarija koji se koristi kao osnova za odlučivanje o tome koja funkcionalnost treba da se uključi u inkrement sistema
2. *Korisnik sistema je uključen njegovim stalnim angažovanjem u razvojnom timu.* On je u timu odgovoran za definisanje testova prihvatanja sistema.
3. *Ljudi, a ne procesi, su podržani programiranjem u paru,* kolektivnom pripadanju koda sistema, i održivom razvojnim procesom koji ne zahteva vrlo dugo vreme rada.
4. *Promene su podržane regularnim izdavanjem novih verzija sistema kupcima,* razvojem u kome su testovi na prvom mestu, restrukturiranjem sistema da bi se izbegla njegova degeneracija, i stalna integracija novih funkcionalnosti (funkcija).

5. Održavanje jednostavnosti koje poboljšava kvalitet koda i u kome se upotrebljava jednostavna projektna rešenja koja ne ograničavaju buduće promene sistema.

PRAKSA EKSTREMNOG PROGRAMIRANJA

Zahtevi nisu specificirani u obliku lista zahtevanih funkcija sistema, već u vidu scenarija koji nastaju kao rezultat diskusije predstavnika korisnika i razvojnog tima.

Kod XP procesa, korisnici si vrlo uključeni u specifikaciju zahteva sistema i u određivanje prioriteta. Zahtevi nisu specificirani u obliku lista zahtevanih funkcija sistema., već u vidu scenarija koji nastaju kao rezultat diskusije predstavnika korisnika i razvojnog tima. Oni zajedno razvijaju „**kartu sa pričom**“ (engl. **story card**) koja sadrži potrebe korisnika sistema. Razvojni tim onda ima zadatak da primeni ovaj scenario u sledećoj verziji softvera.

Sledeća tabela ilustruje kako XP proces proizvodi jedan inkrement sistema u razvoju.

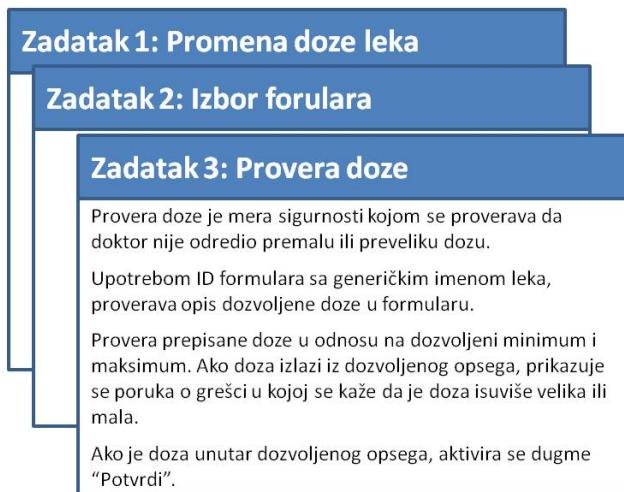
Princip ili praksa	Opis
Inkrementalno planiranje	Zahtevi se upisuju na Karti sa pričom, tako da se priča uključuje u novu verziju sistema, zavisno do potrebnog vremena razvoja i od prioriteta. Razvojni inženjeri dele ove priče u vidu zadataka (slike)
Male verzije	Prvo se razvija minimalno koristan skup funkcionalnosti koji obezbeđuje poslovnu vrednost. Nove verzije sistema su česte i dodaju novu funkcionalnost inkrementalno.
Jednostavno projektno rešenje	Traži se projektno rešenje koje zadovoljava tekuće zahteve, i ništa više od toga.
Razvoj oslonjen na testiranja	Koristi se automatizovani sistem za testiranje jedinica i pisanj etestova za novi deo funkcionalnosti, pre nego što je taj deo funkcionalnosti i применjen.
Restrukurisanje	Svi razvojni inženjeri bi trebalo da kontinualno rade na prestruktursanju koda čim se izvrše promene u kodu. To čini jednostavnim i održivim.
Programiranje u paru	Razvojni inženjeri rade u paru, proveravajući rad jedan drugome, i obezbeđujući uzajamnu podršku da bi uradili dobar posao.
Pripadnost kolektivu	Par razvojnih inženjera radi u svim oblastima sistema, tako da se ne javljaju „ostrva ekspernosti“ i što omogućava da svi razvojni inženjeri preuzimaju odgovornost za ceo kod. Svako može da menja bilo šta.
Stalna integracija	Čim se završi posao na jednom zadatku, on se integriše u sistem. Posle svake takve integracije, svi testovi jedinica se moraju ponovo uspešno realizovati
Održivost	Ne podržava se masivni prekovremeni rad jer to dovodi smanjivanje kvaliteta koda i smanjivanje produktivnosti u bliskoj budućnosti.
Uključenost korisnika	Predstavnik korisnika sistema bi trebalo d abude uključen sve vreme u rad sa razvojnim timom. U XP procesu, korisnik je član razvojnog tima i on je odgovoran za definisanje sistemskih zahteva koje daje razvojnom timu.

Slika 4.2

PRIMENA KARATA SA PRIČAMA

Kada se karte sa pričama razviju, razvojni tim svaki kartu podeli na zadatke i onda procenjuje potreban rad i resurse za implementaciju

Karte sa pričama su glavni ulaz u XP proces planiranja. Kada se one razviju, razvojni tim svaki kartu podeli na zadatke i onda procenjuje potreban rad i resurse za implementaciju (realizaciju) svakog zadatka (slika 3)..



Slika 4.3 Primer karata sa zadacima za recepte za lekove

Ovo obično podrazumeva razgovor sa korisnicima radi poboljšanja zahteva. Korisnik onda određuje prioritete implementacije, izabirajući one koje se

mogu odmah primeniti i koje obezbeđuju korisnu podršku poslovanju. Namera je da se odredi korisna funkcionalnost koja se može primeniti u roku oko dve nedelje, kada je vreme za izbacivanje nove verzije sistema, i koja se šalje korisniku

Ako dođe do promena zahteva, odbacuju se ne primenjene krte. Ako se zahtevane promene odnose na već primenjen sistem, razvijaju se nove karte, i opet, kupac odlučuje o prioritetima uvođenja novih funkcionalnosti.

U slučajevima kada razvojni time ne može lako da dođe do rešenja, određuje se dodatno vreme za traženje rešenja. Za to vreme, moguće je i razvoj prototipa rešenja da bi se vršile provere mogućih rešenja.

Ekstremno programiranje se naziva „ekstremnim“ pristupom inkrementalnom razvoju jer se nove verzije softvera mogu izrađivati i nekoliko puta u toku dana, a konačna verzija, tj. Izdanje koja se šalje kupcu, isporučuje s ekupcu svake dve nedelje. *Rokovi se nikada ne menjaju*, a ako ima problema, uz konsultaciju sa kupcem, sporna funkcionalnost, tj. sporni deo softvera se izostavlja iz planiranog izdanja softvera, a ono što ostaje, šalje se kupcu.

PROBLEM DEGRADACIJE STRUKTURE SOFTVERA

Opšti problem kod inkrementalnog programiranja je da on dovodi do degradacije strukture softvera, tako da se buduće promene sve teže i teže implementiraju

Kada programer razvija novu verziju, on mora da izvrši automatske testove i za sve postojeće softverske jedinice, kao i za nove, koje obezbeđuju novu funkcionalnost. Nova verzija se

prihvata samo ako svi testovi daju pozitivne rezultate. Takva verzija postaje onda osnova za sledeći ciklus promena softvera.

U ekstremnom programiranu se ne primenjuje princip po kome se projektovanje vrši tako da se omoguće lake buduće promene, jer se smatra da to zahteva dodatno vreme u razvoju, koje se ne želi potrošiti za tu namenu. Kod XP pristupa, promene sistema se planiraju samo kada se te promene zahtevaju, tj. one se unapred ne planiraju.

Opšti problem kod inkrementalnog programiranja je da on dovodi do degradacije strukture softvera, tako da se buduće promene sve teže i teže implementiraju. To je posledica traženja rešenja za konkretnu promenu, ne vodeći računa o optimizaciji koda i njegovoj strukturi, te se desi da dođe i dupliranja koda, a i do kopiranja delova softvera na ne odgovarajući način, te dolazi do degradacije strukture kode kada se on dodaje u sistem.

Taj problem ekstremno programiranje rešavanja **stalnim restrukturiranjem softvera**. Razvojni tim razmatra načine da se poboljša struktura softvera i nađeno rešenje odmah primenjuje.

Kada neki član tima primeti da se kod može poboljšati, on to odmah radi, iako ta promena nije povezana za zahtevom za promenama na kojoj radi. Na primer, tako se može promeniti struktura klasa, da bi se uklonio dupliran kod, ili promeniti nazivi atributa i metod i zamjeniti kod sa pozivima metoda koji su definisani u programskoj biblioteci. Programske alati, kao što je Eclipse, podržavaju restrukturiranje koda jer nalaze zavisnost promena opšteg koda izabranog koda.

Iako bi trebalo da se na opisan način softver uvek održava u dobrom stanju, tako da je lako čitljiv i strukturisan, u praksi nije uvek tako. Pod vremenskim pritiskom da se primene nove funkcionalnosti, inženjeri razvoja ne posvećuju dovoljno vremena usavršavanju samog koda, sem kada nova funkcionalnost zahteva promenu strukture i arhitekture sistema.

U praksi, mnoge kompanije ne primenjuju ekstremno programiranje u skladu sa principima datim u tabeli, već primenjuju samo neke od njih, u skladu sa svojim načinom rada. Ali, ono što najčešće primenjuju, to su česte i malo promenjene verzije sistema, definisanje najpre testova, i stalna integracija.

TESTIRANJE U EKSTREMНОM PROGRAMIRANJU

Prvo se definiše test, pa se onda razvija kod. To omogućava testiranja programa još kada je u fazi razvoja.

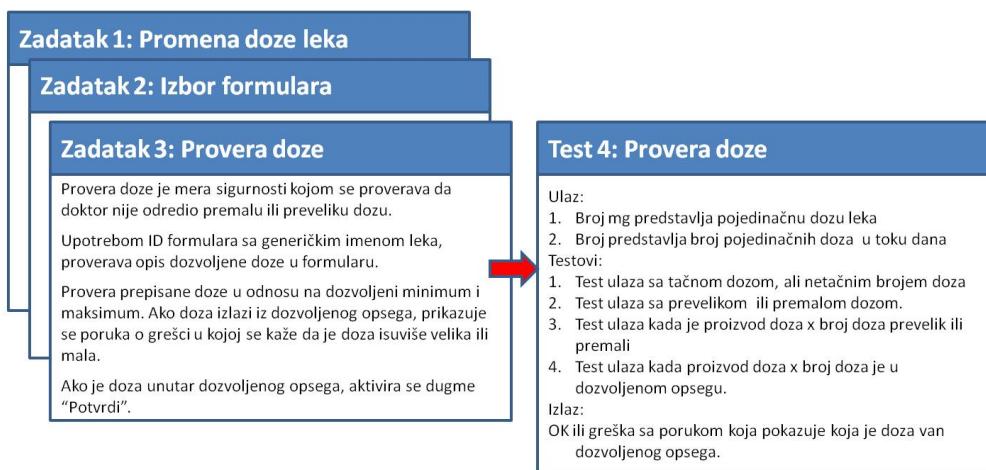
Ekstremno programiranje daje značaj testiranju. Osnovna svojstva testiranja u ekstremnom programiranju su sledeća:

1. Prvo se pripremaju testovi.
2. Razvijaju se inkrementalni testovi na osnovu scenarija.
3. Korisnik učestvuje u razvoju i validaciju testova.
4. Koriste se sistemi za automatsko testiranje.

Umesto da se test definiše posle pisanja koda, kod ekstremnog programiranja je obrnuto. Prvo se definiše test, pa se onda razvija kod. To omogućava testiranja koda još kada je u fazi razvoja.

Pisanjem testa se implicitno definišu i interfejs i specifikacija ponašanja za funkcionalnost koja se razvija. Ovaj pristup se može da primenu svuda gde postoji jasan odnos između sistemskih zahteva i koda koji ih primenjuje. Ta veza se kod ekstremnog programiranja jasno vidi jer karta sa pričom (scenarijom) predstavlja zahteve koji se razbijaju na zadatke, a zadaci su glavne jedinice implementacije.

Pri pisanju testova otklanjaju se i sve nejasnoće oko zahteva, tj. pre nego što implementacija počne. Na slici 4 je prikazan primer testova jedinica određenih za jedan od zadataka koji su definisani na osnovu prethodno definisanog scenarija. Za svaki zadatak se na taj način definišu jedan ili više jediničnih testova, pre nego što implementacija počne.



Slika 4.4 Primer testova za zadatak: "Proveri dozu leka"

AUTOMATIZACIJA TESTOVA

Da bi se testiranje ubrzalo, izvršenje testova se automatizuje. Testovi se pišu kao izvršne komponente pre nego što se implementira zadatak

Da bi se testiranje ubrzalo, izvršenje testova se automatizuje. Testovi se pišu kao izvršne komponente pre nego što se implementira zadatak. To su samostalne programske komponente koje obrađuju ulaze i proveravaju ulaze u skladu sa definisanim testom. Postoje alati za pisanje automatskih testova, tako da oni izvršavaju grupe testova. (na primer, Junit).

Uvek kada se doda nova funkcionalnost, pokreće se izvršenje svih (i starih i novih) testova jedinica, tako da se može uočiti eventualno negativna posledica uvođenja novog koda u sistem.

Iako se na ovaj način programi mogu vrlo detaljno i pouzdano testirati, u praksi nije uvek tako.

1. Programeri više vole programiranje nego testiranje, te pri definisanju testova koriste skraćene postupke testiranja, što nije dobro. Nekompletni testovi ne vrše proveru svih izuzetaka koji se mogu javiti u radu sistema
2. *Neki testovi se teško mogu inkrementalno napisati.* Na primer, složeni korisnički interfejsi se teško proveravaju testovima jedinica. Jedinični testovi nisu pogodni za proveru „logike prikazivanja“ i za radne tokove.
3. *Teško je oceniti kompletност izvršenih skupa testova.* Možda i vrlo važan segment sistema nije obuhvaćen nekim testom.

Ako se ne vrši provera napisanih testova, i ako se ne radi na dopuni nedostajućih testova, onda se mogu javiti neotkrivene greške (tzv. bagovi) u sistemu.

PROGRAMIRANJE U PAROVIMA

Podržava ideju o osećanju kolektivnog autorstva, realizuje neformalna recenzija procesa razvoja i omaže rad na usavršavanu koda, tj. restrukturiranje, tokom razvoja izmena sistema

Programiranje u parovima je česta praksa pri primeni ekstremnog programiranja. Programeri rade u paru te zajedno na istoj radnoj stanicici razvijaju softver. Međutim, članovi para ne mora uvek da programiraju zajedno. Parovi se dinamički planiraju, te ne mora celo vreme da parovi budu isti. Raspored članova razvojnog tima, po parovima, se može menjati s vremena na vreme.

Rad u parovima obezbeđuje sledeće povoljnosti:

1. *Podržava ideju o osećanju kolektivnog autorstva nad svakom linijom koda.* Tim, a ne pojedinac, preuzima odgovornost za ceo kod i svaki njegov deo, i zajedno radi na rešavanju problema.
2. Na ovaj način se realizuje neformalna recenzija procesa razvoja, jer svaku liniju koda gleda najmanje dva člana tima. Provera i recenzija koda se uspešno primenjuje na ovaj način. Međutim, zbog dodatnih vremenskih zahteva za ovakav način rada, rad u parovima zahteva duže vreme, i može da malo uspori projekat razvoja. Mada rad u paru nije tako efikasan kao formalna inspekcija, on je mnogo jeftiniji za realizaciju od formalne inspekcije.
3. Pomaže rad na usavršavanu koda, tj. restrukturiranje, tokom razvoja izmena sistema, što donosi koriste na duže staze.

Gledano po članu tima, rad u parovima za određeno vreme proizvede za 50% manje koda nego ako oni radi samostalno. Međutim, to ne mora da bude tako. Njihova produktivnost može biti slična produktivnosti ako rade samostalno, ako zajedničkim diskusijama na vreme otkriju moguće greške i koje mogu kasnije nastati kada počnu da kodiraju. To smanjuje broj kasnijih grešaka, a takođe i ubrzava otklanjanje kasnije otkrivenih grešaka.

Praksa ipak pokazuje da rad u parovima smanjuje produktivnost tima, ali povećava kvalitet rada. Takođe, daje stabilnost timu, jer ako ga napusti neki član tima, drugi član može bez problema da radi poslove na kojima je radio član koji je otišao.

PRIMER: LEĆENJE PACIJENATA SA MENTALNIM POREMEĆAJIMA

Na osnovu određenog scenarija (karte sa pričom), razvojni tim scenarijo zamenjuje nizom zadataka, koji treba da realizuju scenario.

Na slici 5 je dat kratak opis scenarija određivanja neophodnih lekova za pacijenta.

Određivanje lekova terapije pacijentu
Kate je doktor koji treba da odredi terapiju i lekova za jednog pacijenta klinike. Istorijat bolesti pacijenta je već prikazan na njenom monitoru kompjutera tako da klikom na dugme „terapija“ dobija izbor opcija „sadašnja terapija“, „nova terapija“ ili „formular“. Ako izabere „sadašnja terapija“, sistem je traži da proveri dozu. Ako želi da promeni dozu, onda može da unese novu dozu i da potvrdi terapiju sa izmenjenom dozom. Ako izabere „nova terapija“, sistem predspostavlja da ona zna koju terapiju treba da odredi. Ona unosi nekoliko prvih slova naziva terapije. Sistem prikazuje listu mogućih terapija koje koriste ista početna slova. Ona bira jednu od tih terapija i sistem je pita da potvrdi tačnost izabrane terapije. Ona unosi dozu i potvrđuje terapiju. Ako izabere „formular“, sistem prikazuje boks pretraživanja formulara za odobrenje. Onda ona traži zahtevan lek. Ona bira lek a sistem od nje traži da potvrdi izbor leka. Ona unosi dozu i potvrđuje odabran lek sa dozom. Sistem uvek proverava da li je doza u natar dozvoljenog intervala vrednosti. Ako nije, on zahteva od lekara da promeni dozu. Kada je Kate potvrdila terapiju, ona će biti prikazana na ekranu radi dodatnog potvrđivanja. Ona može da pritisne „OK“ ili „Promena“. Ako klikne „OK“, nova terapija se beleži u bazu revizije. Ako klikne „Promena“, onda se obnavlja proces „Određivanje terapije“.

Slika 4.5

Na osnovu ovog scenarija (karte sa pričom), razvojni tim scenarijo zamenjuje nizom zadataka (prikazanih na slici) i procenjuju potreban rad i resurse za primenu svakog zadatka. Obično o tome diskutuju sa kupcem/korisnikom sistema. Kupac određuje prioritet..

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju proces ekstremnog programiranja.

- 1. Zadatak:** Dati primer ekstremnog programiranja i prikazati prednosti takvog načina razvoja softvera.
- 2. Zadatak:** Za proizvoljnu aplikaciju podeliti razvojni proces na karte sa pričama. Identifikovati minimum pet karata i detaljno ih objasniti i predstaviti.

▼ Poglavlje 5

Primena agilnih metoda u većim projektima

SPECIFIČNOSTI VELIKIH SOFTVERSkiH SISTEMA

Veliki softverski sistemi imaju svoje specifičnosti koje ometaju primenu agilnih metoda u njihovom razvoju.

Agilne metode su razvijene za upotrebu od strane malih programerskih timova koji mogu da rade u istoj prostoriji i da neformalno komuniciraju. Zbog toga se najčešće koriste za projektima razvoja malih i srednjih sistema. Kako i kod velikih projekata postoji pritisak da se sistem što pre isporuče kupcu, postavilo se pitanje da li, i kako, agilne metode mogu da se primene i u projektima razvoja velikih sistema.

Specifičnosti razvoja velikih softverskih sistema:

1. *Veliki sistemi su najčešće skupovi posebnih sistema koji međusobno komuniciraju, a koje razvijaju različiti timovi. Ti timovi često rade na različitim, pa i vrlo udaljenim lokacijama. Ti timovi nemaju vidljivost celog sistema. Njihov prioritet, zbog toga, je da završe svoj deo sistema ne obazirajući se na ostale delove.*
2. *Veliki sistemi komuniciraju i uključuju mnoge druge postojeće sisteme. Mnogi od njih služe za obezbeđivanje integracije i integracije ovih sistema, i ne obezbeđuju direktno neku posebnu funkcionalnost celog sistema, niti obezbeđuju fleksibilnost i inkrementalni razvoj.*
3. *Kada su više sistema integrirani u novi sistem, značajan deo razvoja se bavi konfiguracijom sistema, a ne razvojem originalnog koda. To nije kompatibilno sa inkrementalnim razvojem i čestim integracijama sistema.*
4. *Veliki sistemi i njihovi procesi razvoja su često ograničeni spoljnim pravilima i regulacijom koji ograničavaju način njihovog razvoja, jer zahtevaju proizvodnju određenih dokumenata.*
5. *Veliki sistemi imaju dugo vreme naručivanja i razvoja. Teško je održati isti razvojni tim na okupu celo vreme trajanja razvoja, te postoji nepoznavanje celine sisteme od strane novih članova.*
6. *Veliki sistemi često imaju različite zainteresovane aktere (stakeholderse). Zato je teško sve njih uključiti u razvojni proces softvera.*

KAKO KORISTITI AGILNE METODE KOD VELIKIH SISTEMA

Postoji mišljenja da se agilne metode mogu koristiti i kod razvoja velikih sistema, ali uz izvesne modifikacije metoda.

Postoje dve perspektive primene agilnih metoda u razvoju velikih sistema:

1. Perspektiva „scaling up“ upotrebe agilnih metoda za razvoj velikih sistema koji se ne mogu razvijati od strane malih timova.
2. Perspektiva „scaling out“ koja se bavi kako agilne metode se mogu koristiti u velikoj organizaciji sa višegodišnjim iskustvom u razvoju softvera.

Sommerville (2009) je mišljenja da se agilni metoda mogu prilagoditi za primenu u razvoju velikih sistema na sledeći način:

1. **Kod velikih sistem se ne može tim da usmeri samo na razvoj koda.** Mora da se pripremi najpre projektna i sistemska dokumentacija. Mora de projektovati arhitektura sistema i mora da postoji dokumentacija koja opisuje kritičke aspekte sistema, kao što je šema baze podataka, podela poslova članovima tima i dr.
2. **Mora se uspostaviti mehanizam komunikacije unutar razvojnog tima.** To može da obuhvati regularne telefonske i video konferencije između članova tima, kao i česte, kratke elektronske sastanke na kojima timovi upoznaju jedne druge sa napretkom svog posla. U tu svrhu koriste se komunikacioni kanali kao što su: e-pošta, razmena poruka, wikis, i socijalne mreže.
3. **Stalna integracija, pri čemu se ceo sistem konfiguriše uvek kada se unosi neka promena, je nemoguća kada se mora se mora integrisati nekoliko sistema koji čine sistem.** Međutim, neophodno je održavati čest razvoj sistema i proizvoditi nove verzije sistema. Zbog ovoga je potrebno koristiti nove alate za upravljanje konfiguracijom sistema a koji podržavaju rad više timova.

TEŠKOĆE U PRIMENI AGILNIH METODA KOD VELIKIH SISTEMA

Primena agilnih metoda u velikim organizacijama je proces promene kulture organizacije. Za to je potrebno dosta vremena i za hteva upravljanje organizacijskim promenama.

Teškoće koje prate pokušaje primene agilnih metoda u velikim kompanija su sledeće:

1. Menadžeri projekta koji najčešće nemaju iskustva u primeni agilnih metoda *nisu voljni da prihvate rizik novog pristupa*, jer ne znaju kako će to uticati na njihove projekte.
2. Velike organizacije imaju procedure osiguranja kvaliteta i standarde koje svi projekti treba da primenjuju, te zbog birokratske prirode, nisu voljni da ih prilagode agilnim

metodima. Ponekad su u obavezi i da primenjuju određena softverske alate (kao na primer, alat za upravljanje zahtevima) koji nisu za upotrebu kod agilnih metoda.

3. *Agilne metode najbolje rade kada članovi tima imaju vrlo dobre veštine.* U velikim organizacijama, postoji velika varijacija nivoa veština i sposobnosti članova tima, te članovi tima sa nižim nivoima sposobnosti i veština često ne mogu da budu efektivni u primeni agilnih metoda.
4. *Postoji i kulturološki otpor primeni agilnih metoda,* naročito u organizacijama koje imaju dugu tradiciju upotrebe konvencionalnih inženjerskih procesa za razvoj sistema.

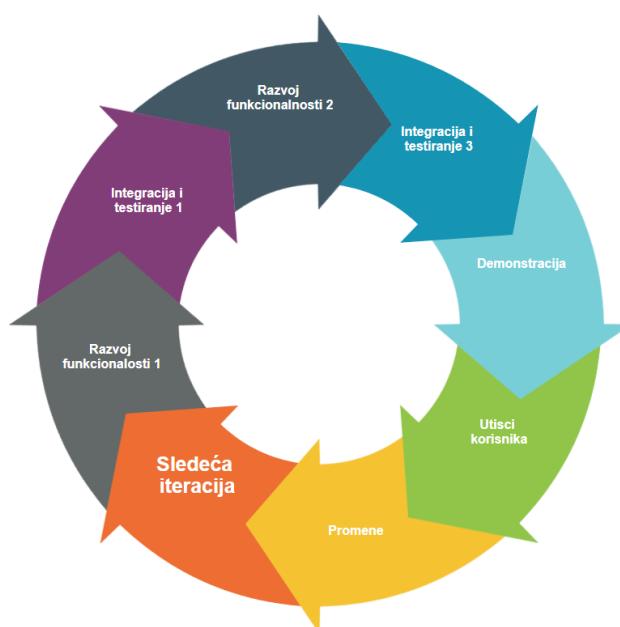
Upravljanje promenama softvera i procedure testiranja su primeri procedura koje nisu kompatibilne sa agilnim metodinima. Sistemi za upravljanje promenama softvera kontrolišu promene sistema, tako da omogućuju predikaciju uticaja promena, kao i troškova tih promena. Sve promene moraju da budu prihvачene unapred. Pri XP, svaki inženjer razvoja poboljšava kod bez saglasnosti nekoga drugog. Za velike sisteme, standardi testiranja definišu način primopredaje softvera spoljnim timovima razvoja. To je u konfliktu sa principom primarnog postavljanja testova i čestim testiranje koje se primenjuju kod XP.

Primena agilnih metoda u velikim organizacijama je proces promene kulture organizacije. Za to je potrebno dosta vremena i za hteva upravljanje organizacijskim promenama.

PRIMER PRIMENE AGILNIH METODA RAZVOJA SOFTVERA NA VEĆIM PROJEKTIMA

Primer obuhvata primenu agilnih metoda razvoja na sistem za upravljanje insulin pumpom.

Agilna metoda razvoja na velikom projektu



Slika 5.1 Agilna metoda razvoja softvera na velikom projektu

Kako bi se uopšte mogla koristiti agilna metoda razvoja softvera na velikom projektu, iako se ne preporučuje, potrebno je da se projekat podeli na što manje delove - storije. Nakon toga potrebno dosta testiranja kao i prikupljanje utiska od krajnjih korisnika. Na datom primeru se može videti da se za početak, implementiraju i testiraju dve funkcionalnosti, nakon čega se vrši demonstracija aplikacije i prikupljaju se utisci korisnika, zatim se ispisuje dokument u kome se nalaze sve primedbe koje se odnose na aplikaciju. Ukoliko su primedbe opravdane, potrebno ih je ispraviti a samim tim se dijagram ponavlja samo se razlikuju funkcionalnosti koje se kodiraju. Kao primer na kome se može prikazati ovaj dijagram može se uzeti i sistem za upravljanje insulin pumpom. Za početak ceo softver je potrebno podeliti na što više manjih storija, napraviti plan rada, podeliti zadatke u okviru malih timova, nakon čega se pristupa izradi funkcionalnosti. Integracija i testiranje se vrše odmah nakon razvoja funkcionalnosti i ukoliko testovi prođu uspešno, potrebno je da se sam napredak u aplikaciji demonstrira stručnim licima koji će davati svoje utiske u vezi aplikacije. U zavisnosti od reakcije stručnih lica, nastavlja se proces razvoja softvera sa nekim novim funkcionalnostima ili se radi revizija i izmena već urađenih funkcionalnosti. Agilnom metodom se dobija dosta vremena za izradu i testiranje aplikacije, a sa utiscima korisnika se dobija povratna informacija koja je od ključnog značaja za razvoj same aplikacije.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci za samostalni rad koji obuhvataju agilni proces razvoja softvera.

- 1. Zadatak:** Dati primer scaling up perspektive upotrebe agilnih metoda na konkretnom primeru.
- 2. Zadatak:** Dati primer scaling out perspektive na konkretnom primeru.

▼ Poglavlje 6

Pokazni primeri

AGILAN RAZVOJ

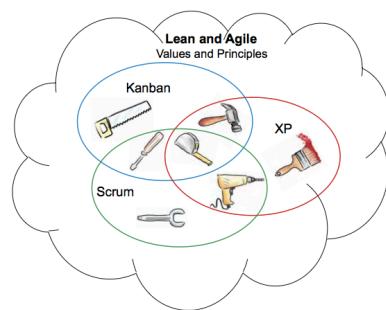
Definicija i nastanak agilnog razvoja softvera

Agilni razvoj softvera nastao je kao pojam 2001 godine i predstavlja skup okvira i metoda razvoja softvera na načelima iterativnog i inkrementalnog razvoja, gde zahteve korisnika i rešenja implementiraju samoorganizovani, multifunkcionalni razvojni timovi. Agilni razvoj promoviše adaptivno planiranje, razvoj u vremenski ograničenim iteracijama, koji stvara brz i fleksibilan odgovor na promene i konstantni feedback svih učesnika u razvoju kroz učestale inspect-and-adapt aktivnosti. Softver se isporučuje sukcesivno na način da se prvo razvijaju funkcionalnosti sa najvećim poslovnim prioritetom za korisnika. Agilni razvoj podrazumjeva organizovanu kolaboraciju svih učesnika u razvoju sistema i podržava brzo i efikasno donošenje odluka. **Agile manifesto** (slika 1) je uveo pojam agilnosti u razvoj softvera 2001 godine.

Sa postankom agile pokreta, nastale su brojne agilne metode i radni okviri (framework) kao što su Scrum, Lean, Kanban, Extreme Programming (XP), Discipline Agile Development (DAD), Dynamic systems development method (DSDM), Scaled Agile Framework (SAFe), i drugi. Iako je svaka od agilnih metoda i okvira jedinstvena u svom specifičnom pristupu, sve one dele zajedničku viziju i temeljne vrednosti koji potiču iz Agile Manifesta. Najpopularniji agilni okvir današnjice je Scrum i u nastavku ovih vežbi biće objašnjena upotreba SCRUM razvoja korišćenjem Agilefant softverskog alata.



Slika 6.1 Agile manifesto



Slika 6.2 Prikaz Agilnih metoda

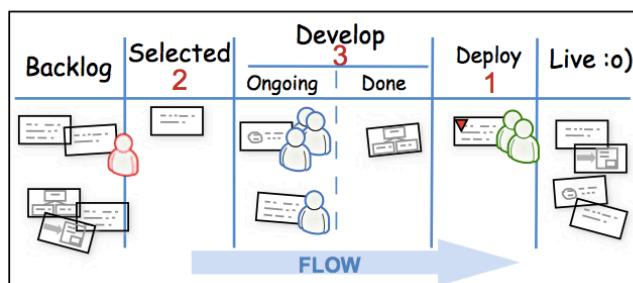
KANBAN

Kanban može biti kartica, vizuelni displej tj. bilo šta što daje signal da se počne sa proizvodnjom određenog dela proizvoda.

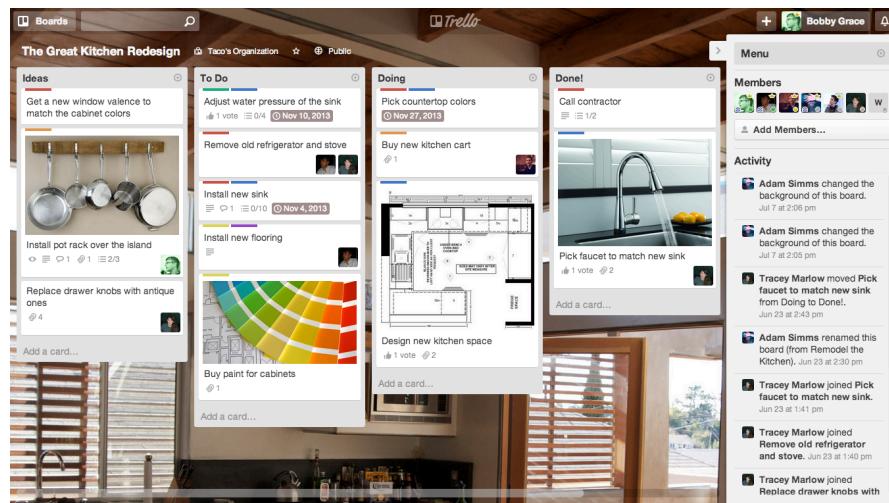
Kanban je japanska reč koja se prevodi kao kartica. Kartice su prikaćene na kontejnere koji sadrže određene delove potrebne proizvodnji. Tek kada delovi iz kontejnera budu počeli da se koriste u proizvodnom procesu, sa kontejnera se skida kartica i postavlja na tablu. Kada je kartica postavljena na signalizacionu tablu, to je signal da je nova količina delova potrebna proizvodnji. Sa ovim sistemom se izbegava fundamentalni gubitak - prekomerna proizvodnja.

Jedan od trenutno popularnih alata za Kanban je Trello (trello.com).

Trello može da se koristi kako za razmenu informacija unutar jednog sektora, tako i između sektora u okviru jedne ili više kompanija.



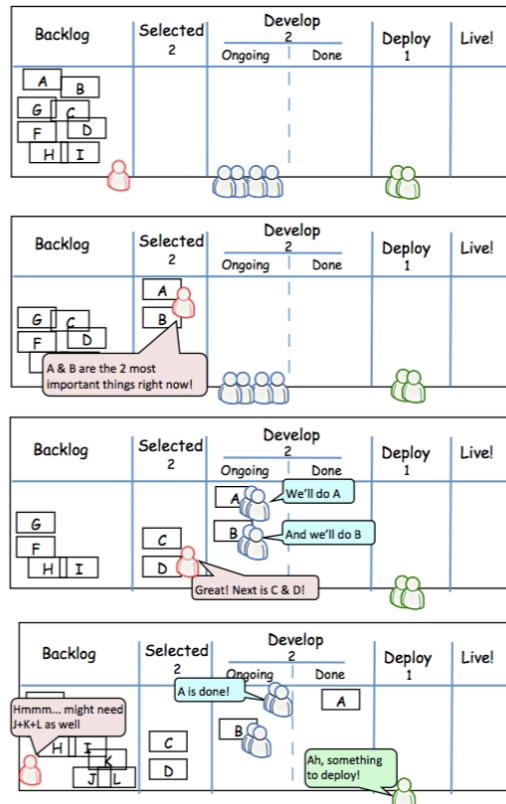
Slika 6.3 Kanban generalni prikaz



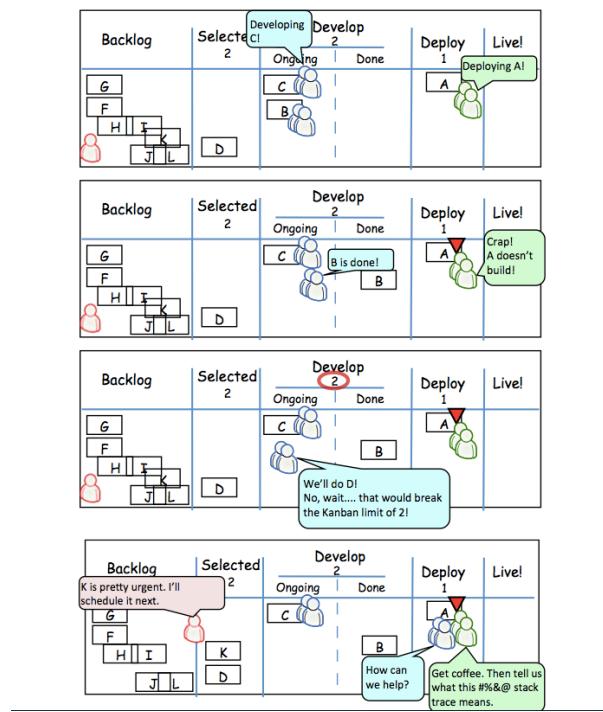
Slika 6.4 Prikaz Trello borda

PRIMENA KANBAN METODOLOGIJE

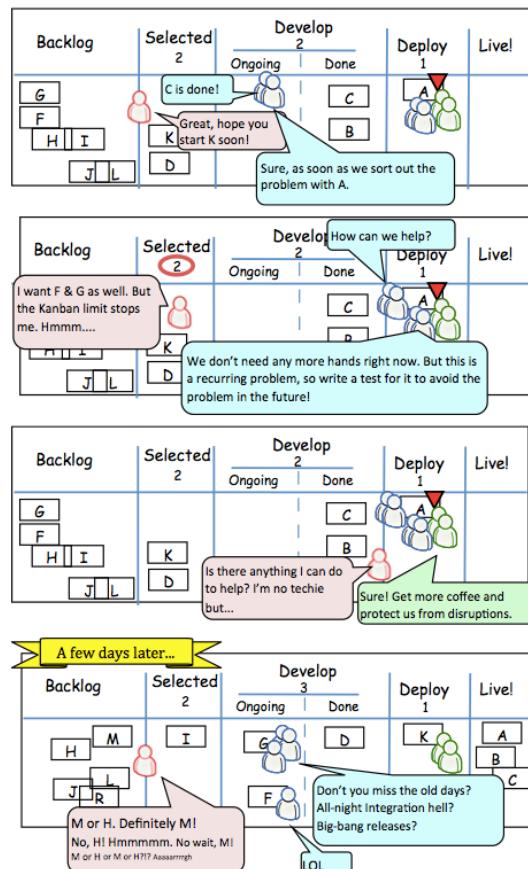
Slikovit prikaz razvoja softvera tokom dana korišćenjem Kanban metodologije



Slika 6.5 Prvi pregled Kanban primera



Slika 6.6 Drugi pregled Kanban primera



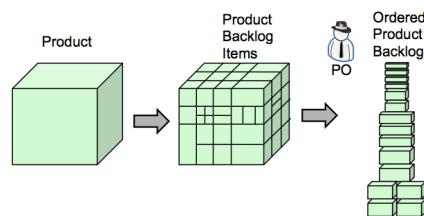
Slika 6.7 Treći pregleda Knban primera

SCRUM

SCRUM predstavlja iterativni i inkrementalni Agilni okvir (framework) za upravljanje softverskim projektima i proizvodima ili razvoju aplikacija

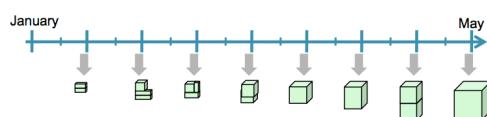
Scrum je postupak razvoja softvera nastao 1990 godine od strane Jeff Sutherland i Ken Schwaber.

Osnovni koncept **SCRUM** metodologije ogleda se u sledećih nekoliko koraka. Prva stvar koju treba uraditi jeste podela projekta u manje celine. Podelite Vaš posao u listu malih konkretnih proizvoda (**product backlog**). **Vlasnik proizvoda** definiše viziju proizvoda i manje proizvode (backlog) u zavisnosti od poslovnih vrednosti i drugih faktora kao što su rizici.



Slika 6.8 Podela projekta na manje celine

Druga bitna stvar jeste podeliti organizaciju na male, funkcionalne, samoorganizuće timove. Svaki tim treba da ima vlasnika proizvoda koji daje viziju i vodi računa o prioritetima i **SCRUM Master-a** koji treba da poboljša tim i pomogne u otklanjanju prepreka.



Slika 6.9 Vremenska raspodela

Treća važna stvar čine **sprintovi** (Sprints). Potrebno je podeliti vreme u male iteracije fiksne dužine odnosno sprintove (standradno 2 ili 3 nedelje dužine). Tim bira koliko proizvoda će se naći u tim malim iteracijama. Svaka iteracija završava se demonstracijom i testiranjem potencijalnog izdanja (release)

Kreator proizvoda prilagođava i optimizuje plan daljih iteracija sa kupcem softvera nakon dostavljanja svake iteracije. Ovaj plan se dogovara sa klijentom jer je moguće da je došlo do promena prioriteta i drugo.

Tim optimizuje proces razvoja svake sledeće iteracije tako što analizira retrospektivno svaku iteraciju koja je završena.

Dakle sa SCRUM metodologijom, umesto kreiranja velike grupe koja provodi mnogo vremena za izradu velikog proizvoda, dobija se mala gurpa koja za kratko vreme pravi male proizvode.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci za samostalni rad koji obuhvataju agilni proces razvoja softvera.

1. Zadatak: Shodnu SCRUM tehnicu izvršiti planiranje postupka razvoja softvera. Opisati svaki korak za proizvoljnu aplikaciju i simulirati podelu organizacije koja se bavi razvojem softvera.

2. Zadatak: Prikazati primenu kanban metodologije na konkretnom primeru upotrebom Trello alata. Isplanirati proces razvoja jednog modula sistema za upravljanje insulin pumpom.

✓ Poglavlje 7

Agilefant softverski alat

AGILEFANT OSNOVNI KONCEPTI

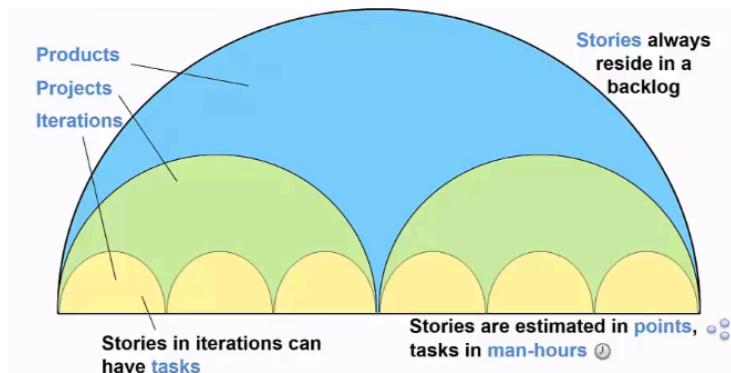
Osnovni koncepti Agilefant softverskog alata. Podela proizvoda na projekte i iteracije.

U okviru Agilefant SW postoji troslojna podela (backlog-s) i to su proizvod, projekat i iteracija. Proizvodi sadrže sve karakteristike, greške i ideje koje mogu biti implementirane. Unutar proizvoda mogu se kreirati projekti. Projekti sadrže priče koje je neophodno uraditi npr. pre puštanja komercijalne verzije.

Unutar proizvoda mogu se naći iteracije. Iteracije sadrže priče na kojim a se trenutno radi uključujući i zadatke koji su nephodni da bi se priča završila. Priče se uvek nalaze na jednom od tri nivoa (proizvod, projekat ili iteracija) i vidljive su kroz nivoe. Priče se predstavljaju/mere kao priča-poen (story points), dok se iteracije mere po čovek-sat (man-hours).

U okviru Agilefant softverskog alata moguće je pregledati i organizovati projekat na male celine po želji u zavisnosti od biznis zahteva, pravila i drugo.

Agilefant se može preuzeti ili koristit kao SaaS (Software as a service) na linku (agilefant.com). Nepohodno je kreirati nalog. Nakon toga korišćenje je dozvoljeno besplatno za 5 osoba u timu. Detaljno uputstvo za korišćenje može se naći na sledećem linku <https://www.agilefant.com/support/user-guide/>



Slika 7.1 Agilefant koncept

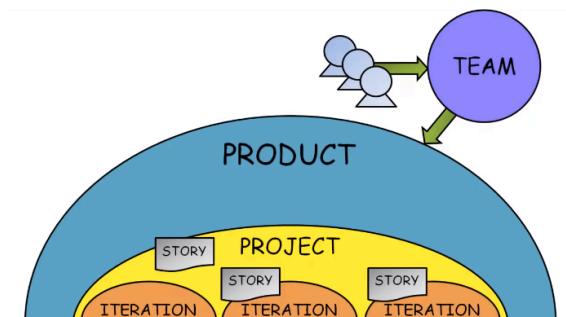


AGILEFANT KONCEPT UPOTREBE

Agilefant koncept upotrebe. Najbolja praksa upotrebe Agilefant softvera u različitim situacijama razvoja softvera.

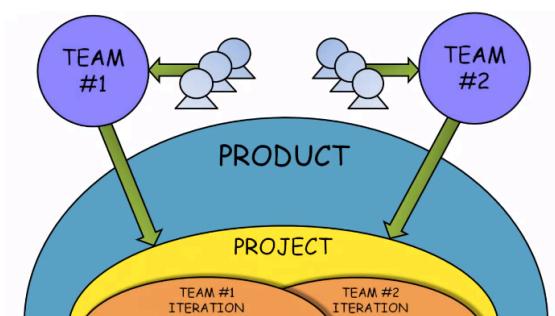
U okviru Agilefanta proizvodi su glavno mesto za skladištenje informacija. Kao što je već rečeno proizvodima se predstavljaju projekti na kojima se radi. Ukoliko je potrebno proizvodima se može nazvati i sam klijent u zavisnosti od potreba.

Kada je kreiranja prozivoda potrebno je kreirati tim koji će imati određen broj članova i omogućiti im pristup kriermanom proizvodu. Nakon dodavanja tima u projekat neophodno je kreirati "priče". Obzirom da se Agilefant zansiva i na projektima i iteracijam.



Slika 7.2 Agilefant generalni koncept

Nakon kreiranja projekata i iteracija neophodno je "prebaciti/dodeliti" priče određenim projektima i iteracijama. U slučaju da u okviru jednog projekta postoji više timova nepohodno je kreirati iteraciju za svaki tim posebno.

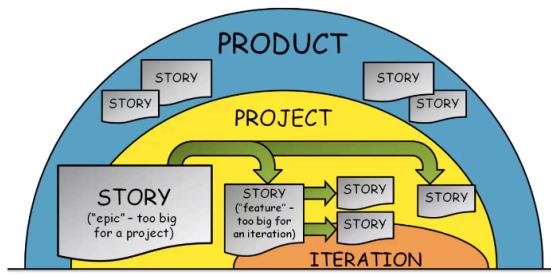


Slika 7.3 Kolaboracija više timova na jedno projektu

U slučaju da imamo samo jedan tim koji radi na više projekata neophodno je kreirati samostalnu iteraciju. Agilefant ima mogućnost kombinovanja ljudi između različitih timova ili pojedinačan rad na više projekata u zavisnosti od potreba.

Agilefant takođe omogućava hijerarhiju priča, odnosno ukoliko je priča velika da stane u jedan proizvod može se podeliti na više pod-priča nazvanih **features**. U slučaju da je i pod-priča dosta velika i ona se može razbiti na manje priče.

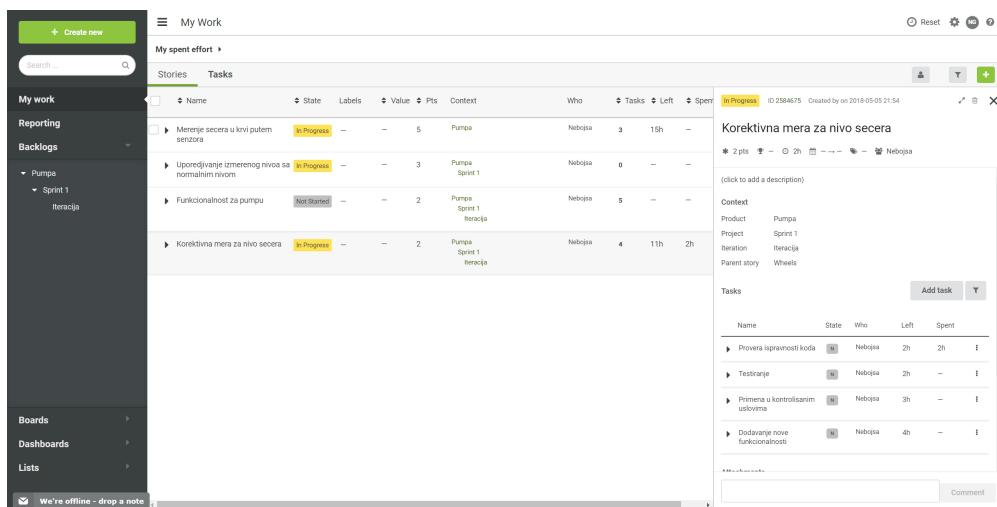
Pored toga Agilefant ima mogućnost podele priča na više delova. Ukoliko je priča prevelika ono što nije završeno može se prebaciti u sledeću iteraciju.



Slika 7.4 Podela priča

PRIMER KORIŠĆENJA AGILEFANT SOFTVERA U AGILNOM PROCESU RAZVOJA

U primeru koji je dat predstavljena je upotreba Agilefant softvera za agilni razvoj na sistemu za upravljanje insulin pumpom.



The screenshot shows the Agilefant software interface. On the left, a sidebar includes 'Create new', 'Search...', 'My work', 'Reporting', 'Backlogs', 'Pumpe', 'Sprint 1', and 'Iteracija'. The main area is titled 'My spent effort' and contains two tabs: 'Stories' and 'Tasks'. The 'Stories' tab displays a list of items:

Name	State	Labels	Value	Pts	Context	Who	Tasks	Left	Spent
Merenje sečera u krvi putem senzora	In Progress		-	5	Pumpa	Nebosja	3	15h	-
Upoređivanje izmerenog nivoa sa normalnim nivom	In Progress		-	3	Pumpa Sprint 1	Nebosja	0	-	-
Funkcionalnost za pumpu	Not Started		-	2	Pumpa Sprint 1 iteracija	Nebosja	5	-	-
Korektivna mera za nivo sečera	In Progress		-	2	Pumpa Sprint 1 iteracija	Nebosja	4	11h	2h

The right side of the screen shows a detailed view of the 'Korektivna mera za nivo sečera' story, including its context and tasks:

Korektivna mera za nivo sečera
 * 2 pts 2h Nebosja
 (click to add a description)

Context
 Product: Pumpa
 Project: Sprint 1
 Iteration: Iteracija
 Parent story: Wheels

Tasks

Name	State	Who	Left	Spent
Provjeri ispravnost koda	In Progress	Nebosja	2h	2h
Testiranje	In Progress	Nebosja	2h	-
Primena u kontrolisanim uslovima	In Progress	Nebosja	3h	-
Dodavanje nove funkcionalnosti	In Progress	Nebosja	4h	-

Slika 7.5 Korišćenje Agilefant softvera za razvijanje softvera pumpe za doziranje insulina

Na slici 6. se može videti način na koji se koristi Agilefant sa ciljem praćenja i kontrole razvoja softvera pumpe za insulin. U okviru My Work kartice se mogu naći svi storiji (mali delovi

softvera) koje je potrebno razviti, a u okviru njih se mogu pronaći i taskovi odnosno zadaci koji se moraju ispuniti kako bi se uspešno napravio deo softvera. Takođe se može videti i utrošeno vreme na nekom od zadataka, ali i predviđeno vreme za izvršenje zadatka koje obično programer govori prilikom preuzimanja zadatka. Pošto se svaka agilna metoda razvoja softvera sastoji iz velikog broj iteracija, pomoću programa Agilefant se može pratiti i taj segment razvoja softvera. Na konkretnom primeru se vidi deo programa Korektivna mera za nivo šećera, koja se odnosi na pumpu i u slučaju povišenog šećera potrebno je da ona reaguje. Mogu se videti i četiri zadatka koja su vezana za ovaj stori a to su: Dodavanje nove funkcionalnosti, Primena u kontrolisanim uslovima, Testiranje i Provera ispravnosti samog koda. Takođe može se videti i očekivano vreme za koje bi zadatak trebao da se ispunji kao i već potrošeno vreme. Sa Agilefant-om se može pratiti i statistika utrošenih sati na samom projektu.

ZADACI ZA SAMOSTALAN RAD

Upotreba Agilefant softverskog alata na konkretnom primeru

1. U okviru Agilefant softverskog alat potrebno je napraviti proizvod (Projekat SE201) i u okviru njega treba napraviti nekoliko proizvoda. Potrebno je kreirati sve priče koje se odnose na razvoj porjektnog zadatka, koje treba dodeliti određenim proizvodima. Napraviti nekoliko iteracija koje će definisati razvoj projektnog zadatka. Rasporediti priče po iteracijama i za svaku priču definisati taskove koji treba da se urade, i procenjeno vreme izrade.
2. Potrebno je kreirati web prodavnici automobilske opreme. Razraditi projekat po sopstvenom izboru. Krierati plan razvoja korišćenjem Agilefant softverskog alata. Služiti se Agilnim tehnikama i SCRUM metodologijom.
3. Preuzeti OpenSource verziju Agilefant alata i pokrenuti ga na lokalu (<https://www.agilefant.com/open-source/>).

Napomena: Prva dva zadatka raditi na Cloud-u koji nudi Agilefant.

✓ Poglavlje 8

Zaključak

ZAKLJUČAK

1. Agilni metodi su metodi inkrementalnog razvoja usmerene ka brzom razvoju, čestim publikovanja novih verzija softvera, smanjivanjem troškova administriranja procesa, i usmerena ka dobijanju koda visokog kvaliteta. Oni uključuju korisnika direktno u proces.
2. Odluka da li upotrebiti agilni ili planom vođen pristup razvoja zavisi od
 - a) tipa softvera koji se razvija,
 - b) sposobnosti razvojnog tima, i
 - c) kulture organizacije koja razvija sistem.
3. Ekstremno programiranje je dobro poznat agilni metod koji integriše niz dobrih programerskih praksi, kao što je česta proizvodnja novih verzija softvera, stalno poboljšanje softvera, učešće korisnika u razvojnom timu.
4. Jaka strana ekstremnog programiranja je razvoj automatskih testova pre kreiranje koda. Svi testovi se moraju uspešno izvršiti kada se inkrement integriše u sistem.
5. Scrum metod je agilni metod koji obezbeđuje okvir za upravljanje projektom. U njegovom središtu je skup sprintova, koji su fiksirani vremenski intervali u okviru koji se razvija inkrement sistema. Planiranje se vrši određivanjem prioriteta poslova na spisku zahtevanih poslova (engl. backlog) i i izborom zadatka sa najvećim prioritetom.
6. Primena agilnih metoda kod velikih sistema je teško. Veliki sistemi zahtevaju prethodno projektovanje i dokumentovanje sistema. Stalna integracija je praktično nemoguća kada u projektu učestvuјe nekoliko razvojnih timova.