

Agilni razvoj softvera

Programsko inženjerstvo

Ciljevi poglavlja

- Upoznati se s agilnim metodama.
- Ukazati na razlike između agilnih i planiranih metoda razvoja softvera.
- Ekstremno programiranje, Scrum

Agilne metode

- Nezadovoljstvo s viškom posla (dokumentacije) uključenim u razvojne metode softvera je u periodu od 1980 – 1990 dovelo do stvaranja agilnih metoda.
- Osnovne karakteristike:
 - Fokus je na kodiranju, a ne dizajnu i pisanju dokumentacije;
 - Baziraju se na iterativnom pristupu razvoja softvera;
 - Namijenjene su brzoj isporuci funkcionalnog softvera, te brzim izmjenama postojećeg kako bi se prilagodio zahtjevima tržišta.
 - U velikoj mjeri se oslanjaju na razne alate (npr. autom. test.)
- Cilj agilnih metoda je minimizirati potreban posao (npr. limitiranje dokumentacije) i što brže odgovoriti na promjene zahtjeva bez prevelikih izmjena u postojećem sustavu.



Agile manifesto

- We are uncovering better ways of developing software by doing it and helping others to do it. Through this work, we have come to value:
 - individuals and interactions over processes and tools;
 - working software over comprehensive documentation;
 - customer collaboration over contract negotiation;
 - responding to change over following a plan.
- While there is value on the items on the right, we value the items on the left more.



Principi agilnih metoda

Princip	Opis
Uključenost korisnika	Korisnik bi trebao biti uključen u cijeli proces razvoja softvera. Njegova je uloga zadavanje novih zahtjeva, postavljanje njihovog prioriteta i procjene gotovih iteracija sustava.
Inkrementne isporuke	Softver se isporučuje u inkrementima, pri čemu korisnik specificira koji će zahtjevi biti uključeni u pojedini inkrement.
Ljudi umjesto procesa	Koriste se vještine ljudi u razvojnog timu, procesi nisu unaprijed zadani nego razvojni tim sam definira svoj način rada.
Prihvaćanje promjena	Očekuje se da će se zahtjevi sustava promijeniti i on se dizajnira tako da im se može prilagoditi.
Jednostavnost	Važno je održati softver koji se razvija što jednostavnijim, kao i razvojni proces koji se koristi. Kada je god moguće potrebno je uklanjati složene dijelove iz sustava.



Primjena agilnih metoda

- Razvoj malih i srednje velikih softverskih rješenja.
- Razvoj sustava po narudžbi kada postoji jasna obaveza da predstavnik kupca bude uključen u razvojni proces i ne postoji puno vanjskih pravila i regulacija koji utječu na softver.
- Pošto su agilne metode fokusirane na malim, čvrsto povezanim timovima postoji problem primjene agilnih metoda na velike sustave.



Problemi s agilnim metodama

- Može biti teško zadržati interes korisnika koji uključen u proces.
- Osobine članova tima ne moraju biti prikladne za grupni rad koji podrazumijevaju agilne metode.
- Ukoliko ima više zainteresiranih strana može biti teško odrediti prioritet pojedinih zahtjeva.
- Održavanje jednostavnosti može zahtijevati dodatni rad, što nekada nije jednostavno zbog rokova.
- Problem je sastavljanje ugovora kao kod drugih inkrementnih metoda.

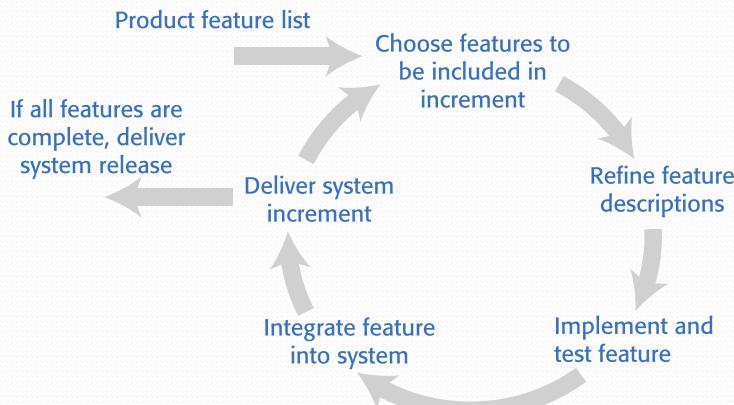


Inkrementalni razvoj

- Sve agilne metode su zasnovane na inkrementalnom razvoju i isporuci
- Razvoj se fokusira na softverskim značajkama (engl. *feature*), a značajka obavlja nekakav posao za korisnika softvera.
- Na početku inkrementalnog razvoja se postavlja prioritet za sve značajke, tako da se one najvažnije prve naprave.
 - Detalji se samo opisuju za one značajke koji će se implementirati u inkrementu.
 - Odabrane značajke se implementiraju i isporučuju korisniku.
- Korisnici ili zamjenski korisnici isprobavaju implementirani sustav i pružaju povratnu informaciju timu. Nakon toga se definira i isporučuje sljedeći inkrement.



Inkrementalni razvoj



Inkrementalni razvoj

- **Odabir značajki za inkrement**
Iz liste značajki planiranog produkta odabiru se one značajke koje će biti implementirane u sljedećem inkrementu.
- **Dodavanje detalja opisu značajki**
Dodaju se detalji opisu odabranih značajki kako bi timu svaka značajka bila jasna i opis treba sadžavati dovoljno detalja kako bi se moglo krenuti s implementacijom.
- **Implementacija i testiranje**
Implementacija značajke i razvoj automatskih testova koji provjeravaju radi li implementirani sustav u skladu s opisom.
- **Integracija značajke i testiranje**
Razvijena značajka se integrira u postojeći sustav, testira se ponašanje sustava u cjelini.
- **Isporuka inkrementa sustava**
Isporučuje se inkrement sustava korisniku/produkt menadžeru kako bi provjerili i komentirali postojeći sustav. Ukoliko je implementirano dovoljno značajki, isporučuje se verzija sustava korisniku.

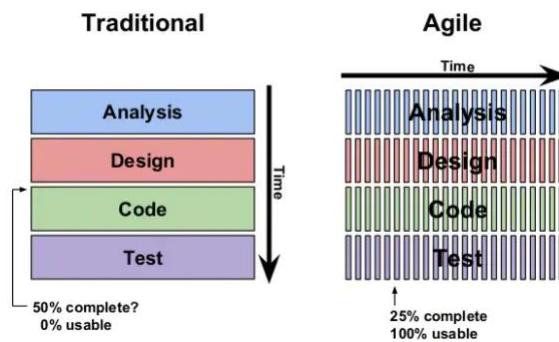
Planirani i agilni razvoj

- Planirani razvoj (engl. plan-driven)
- Planirani razvoj se bazira na strogo razdvojenim razvojnim fazama i svaka od tih faza ima strogo definirane izlaze.
 - Ne mora biti nužno vodopadni, planirani model može biti i inkrementalni.
 - Iteracije se odvijaju među aktivnostima.
- Agilni razvoj
 - Specifikacija, dizajn, implementacija i testiranje su međusobno povezani i izlazni produkti razvojnih procesa se određuju pregovorima u tijeku razvoja.

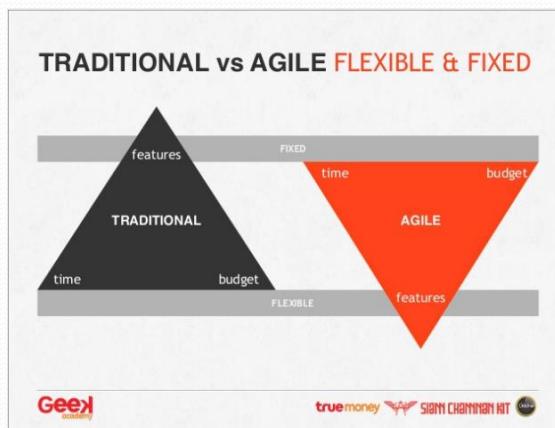


Planirani i agilni razvoj

Software development process



Planirani i agilni razvoj



<http://www.slideshare.net/Zyrauze/gkacdm-introtoagile06222013>



Kada koristiti koji pristup?

- Većina projekata uključuje i planirane i agilne procese, ali njihov omjer je različit i može ovisiti o:
 - Koliko je važno imati detaljnu specifikaciju i dizajn prije nego se kreće na implementaciju? Ako da, treba koristiti planirani pristup.
 - Koliko je realna strategija u kojoj se softver isporučuje inkrementalno? Ako da, koristiti agilni pristup.
 - Koliko je velik sustav koji se razvija?
 - Agilne su metode najefektnije kada sustav mogu razvijati mali samostalni timovi.
 - To možda neće biti moguće kada postoji više razvojnih timova na različitim lokacijama.



Kada koristiti koji pristup?

- Kakav sustav se razvija?
 - Planirani razvoj može biti potreban za sustave koji zahtijevaju opsežnu analizu prije implementacije (npr. sustavi za rad u realnom vremenu s kompleksnim vremenskim ovisnostima).
- Kakav je očekivani životni vijek sustava?
 - Sustavi s dugim životnim vijekom trebaju imati bolje dokumentiran dizajn kako bi se prenijele osnovne ideje timu za održavanje.
- Koje tehnologije su dostupne kao podrška razvoju?
 - Agilne metode se uvelike oslanjaju na alate koji vode računa o tijeku dizajna.
- Kako je organiziran razvojni tim?
 - Ako je razvojni tim fizički distribuiran ili dio razvoja obavljaju drugi timovi, tada najvjerojatnije treba dokumentacija dizajna za komunikaciju među timovima.

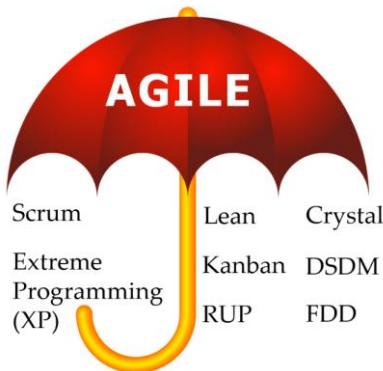


Kada koristiti koji pristup?

- Postoje li kulturološki ili organizacijski faktori koji mogu utjecati na sustav?
 - Tradicionalne organizacije imaju kulturu planiranog razvoja.
- Koliko su dobri dizajneri i programeri u razvojnom timu?
 - Postoji teza da agilne metode zahtijevaju višu razinu programerskog znanja nego planirani razvoj, u kojem se jednostavno detaljan dizajn pretvara u kod.
- Podliježe li sustav vanjskoj kontroli?
 - Ukoliko sustav prije korištenja treba odobriti vanjsko tijelo (npr. softver važan za upravljanje zrakoplova treba odobriti FAA), tada najvjerojatnije treba napraviti detaljnu dokumentaciju.



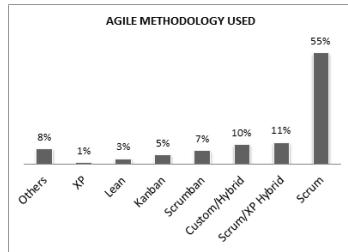
Agilne metode



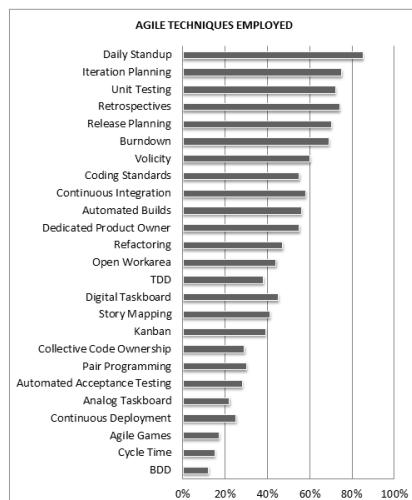
<http://masterofproject.com/blog/130302/agile-frameworks-methodologies>



Popularnost Agilnih metoda

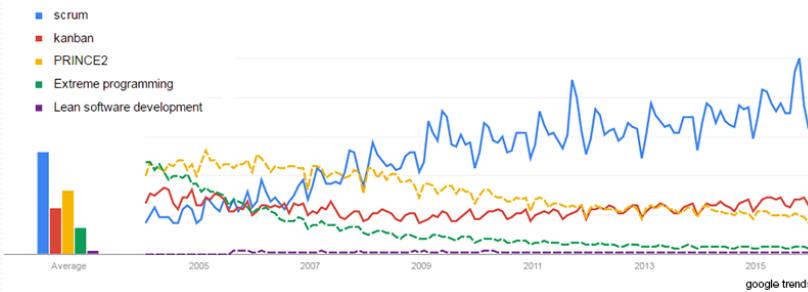


<http://blog.myscrumhalf.com/2015/02/o-estado-do-desenvolvimento-agil-2/?lang=en>



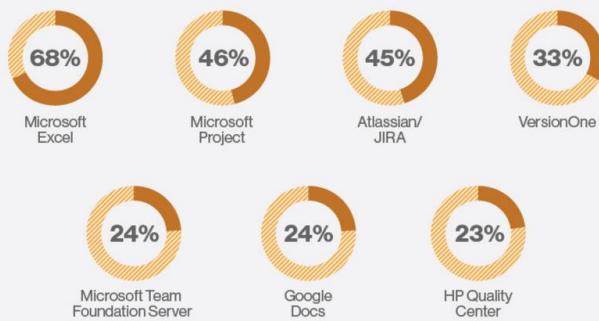
Popularnost Agilnih metoda

Trend of Agile Development Methodologies



Agilne metode u praksi

Top Agile project management tools



Source - VersionOne

<http://searchsoftwarequality.techtarget.com/photostory/4500251251/VersionOne-surveys-Agile-development-for-2015/7/Why-choose-Agile-software-development-tools>



Agilne metode u praksi

The results of implementing Agile

Result	Got Better	No Change	Unknown	Got Worse
Ability to manage changing priorities	87%	2%	10%	<1%
Increased team productivity	84%	3%	12%	1%
Improved project visibility	82%	4%	13%	<1%
Increased team morale/motivation	79%	6%	12%	3%
Better delivery predictability	79%	6%	13%	2%

Legend: Got Better (dark brown), No Change (medium brown), Unknown (light brown), Got Worse (orange)

<http://searchsoftwarequality.techtarget.com/photostory/4500251251/VersionOne-surveys-Agile-development-for-2015/7/Why-choose-Agile-software-development-tools>

FESB Programsko inženjerstvo 27

Agilne metode u praksi

Why choose Agile software development tools?

Reason	Percentage
Accelerate product delivery	59%
Enhance ability to manage changing priorities	56%
Increase productivity	53%
Enhance software quality	46%
Enhance delivery predictability	44%

Source - VersionOne

<http://searchsoftwarequality.techtarget.com/photostory/4500251251/VersionOne-surveys-Agile-development-for-2015/7/Why-choose-Agile-software-development-tools>

FESB Programsko inženjerstvo 28

Tehnike agilnog razvoja



Ekstremno programiranje

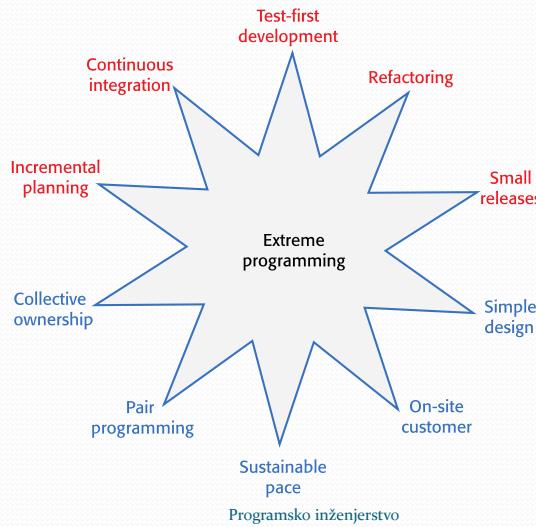
- Engl. *Extreme programming* – XP, pojavila se krajem 90-ih godina prošlog stoljeća i u najvećoj mjeri (od svih ostalih metoda) je utjecala na promjenu kulture razvoja softvera.
- Dobila je ime po tome jer je postoječe dobre pristupe razvoju softvera forsirala do krajnjih granica, tj. iterativan razvoj.
- Ekstremno programiranje je fokusirano na 12 tehnika razvoja koje su imale cilj ubrzati inkrementalni razvoj, promjene i isporuke.
 - Neke od tih tehnika se i danas koriste dok su ostale manje popularne.



Tehnike razvoja ekstremnog programiranja



31



Tehnike razvoja ekstremnog programiranja

Inkrementno planiranje / korisničke priče	Ne postoji 'veliki plan' za sustav, već se ono što se treba implementirati (zahtjevi) u svakom inkrementu dogovara s korisnikom. Zahtjevi se pišu u obliku korisničkih priča, a koje će biti uključene u inkrement se odlučuje prema dostupnom vremenu i njihovom prioritetu.
Male verzije za objavu	Minimalni korisni niz funkcionalnosti koji pruža nekakvu poslovnu vrijednost se prvo razvija. Često se izdaju nove verzije i inkrementalno se dodaju nove funkcionalnosti.
Jednostavan dizajn	Jednostavan dizajn koji je neophodan za postizanje trenutnih zahtjeva.
Razvoj baziran na testiranju	Koristi se razvojna okolina koja podržava jedinično testiranje i testovi se pišu prije implementacije određene funkcionalnosti.
Refaktoriranje	Refaktoriranje znači poboljšavanje strukture, čitljivosti, efikasnosti i sigurnosti programa. Razvojni tim bi trebao cijelo vrijeme procjenjivati kod i mijenjati ga kako bi ostao što jednostavniji i lakši za održavanje.



Programsko inženjerstvo

32

XP u praksi

Programiranje u paru	Članovi razvojnog tima rade u paru, provjeravaju jedan drugog i pružaju podršku.
Zajedničko vlasništvo	Članovi razvojnog tima rade na svim dijelovima sustava, svi poznaju cijeli kod – svi mogu raditi izmjene.
Kontinuirane integracije	Čim se završi rad na nekom zadatku, on se integrira u sustav. Nakon svake integracije pokreću se svi jedinični testovi na cijelom sustavu i svi trebaju biti uspješni da bi se prihvatile nova verzija sustava.
Održiv tempo	Velik broj prekovremenih sati nije prihvatljiv jer kao rezultat dolazi do smanjenja kvalitete rada i produktivnosti.
Prisutnost korisnika	Predstavnik krajnjeg korisnika bi trebao postati član razvojnog tima, a njegova je uloga opis zahtjeva za implementaciju.



Što se najčešće preuzima iz XP-a

- Ekstremno programiranje najčešće nije jednostavno integrirati u različite organizacije.
- Kao rezultat toga, iako se sama metoda kao takva i ne koristi previše, neke ideje i principe koje je uvela se koriste u agilnim metodama.
- Korišteni principi:
 - Korisničke priče
 - Refaktoriranje
 - *Test-first* razvoj
 - Programiranje u paru



Korisničke priče

- U XP-u korisnički zahtjevi se prikazuju u obliku korisničkih priča.
- Svaka korisnička priča je zapisana na kartici i razvojni tim je dijeli na pojedinačne zadatke za implementaciju.
 - Zadaci su osnova za planiranje radnih sati i rasporeda, te za naplatu.
- Korisnik, koji je dio razvojnog tima, bira koje će se korisničke priče uključiti u sljedeću verziju softvera, ovisno o njihovom prioritetu i procjeni rasporeda.



Primjer - MHC-PMS sustav

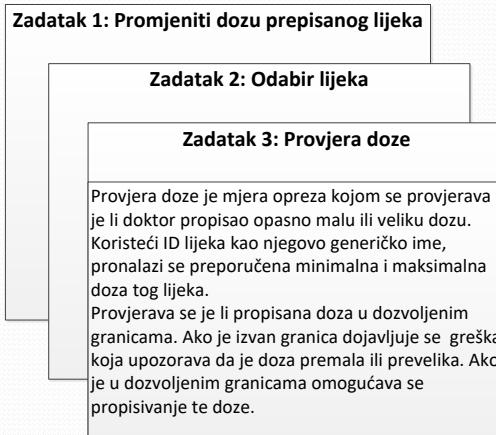
- Korisnička priča za prepisivanje lijeka

Prepisivanje lijeka
Zapis o pacijentu mora biti otvoren za unos. Klikne se na polje o lijekovima i odabere ili 'trenutni lijekovi', 'novi lijekovi' ili 'lijekovi za izradu'.
Ako se odaberu 'trenutni lijekovi' bit će potrebno provjeriti dozu; ako se želi promijeniti doza, unese se nova doza i potvrdi recept.
Ako se odabere 'novi lijek' sustav pretpostavlja da znate koji lijek propisujete. Upiše se prvi nekoliko slova imena lijeka. Tada će se prikazati lista mogućih lijekova koji počinju s tim nizom slova. Izabere se željeni lijek. Zatim je potrebno provjeriti jeste li odabrali željeni lijek. Odabere se doza i potvrdi recept.
Ako se odabere 'lijekovi za izradu', prikazat će se tražilica za željenim lijekom. Pronađe se željeni lijek te ga se odabere. Zatim je potrebno provjeriti jeste li odabrali željeni lijek. Odabere se doza i potvrdi recept.
<u>Za sve slučajeve sustav provjerava je li doza u dozvoljenim granicama i pita želimo li je promijeniti ako nije u preporučenim granicama.</u>
Nakon što se potvrди recept, bit će prikazan za provjeru. Potrebno je kliknuti ili 'OK' ili 'Promijeni'. Ako kliknete 'OK' recept će biti spremljen u bazu. Ako kliknete 'Promijeni' ponovno prolazite kroz proces propisivanja lijeka.



MHC-PMS sustav

- Kartice zadataka - prepisivanje lijeka



Refaktoriranje

- Dobra praksa u programskom inženjerstvu je dizajnirati sustav da omogući promjene.
 - Isplati se utrošiti vrijeme i trud kako bi se predviđele promjene jer to štedi vrijeme (novac) u kasnijim fazama životnog ciklusa softvera.
- Kod XP-a se smatra da se ne isplati trošiti vrijeme na predviđanje promjene jer se to i onako ne može dovoljno kvalitetno napraviti.
 - Radije, on predlaže konstantne izmjene koda kako bi promjene bile što jednostavnije kada ih bude trebalo implementirati.



Refaktoriranje

- Programeri cijelo vrijeme procjenjuju kod i nastoje ga poboljšati čak i kada nema neposredne potrebe za tim.
- Prednosti su:
 - Bolje razumijevanje softvera i manja potreba za dokumentacijom.
 - Promjene su lakše za napraviti jer je kod dobro strukturiran i jasan.
- Međutim u nekim slučajevima promjene mogu zahtijevati izmjene arhitekture što je znatno skuplje.



Testiranje u XP-u

- Testiranje je centralni dio XP-a pa se za testiranje koristi pristup u kojem se cijeli program testira svaki put kada se napravi izmjena.
- Osnovne karakteristike testiranja u XP-u su:
 - “Test-first” razvoj;
 - Inkrementni razvoj testova iz scenarija;
 - Sudjelovanje korisnika u razvoju i validaciji testova;
 - Korištenje automatiziranih okolina koje svaki put kada se pokrene testiranje provodi sve do tada napravljene testove.



“Test-first” razvoj

- Pisanje testova prije samog koda poboljšava razumijevanje zahtjeva koji će se implementirati.
- Testovi su napravljeni kao programi, pa se mogu izvršavati automatski.
- Kada se doda nova funkcionalnost, svi prethodni i novi testovi se automatski izvršavaju i na taj se način provjerava da dodavanje nove funkcionalnosti nije uzrokovalo grešku u prethodnim dijelovima koda.



Uključenost korisnika u testiranje

- Korisnik pomaže u stvaranju testa prihvata za zadatke koje će se implementirati u sljedećoj verziji sustava.
- Korisnik koji je dio tima piše nove testove kako razvoj napreduje. Tako da se sav novi kod provjerava kako bi se osiguralo da je to ono što korisnik traži.
- Međutim najčešće ljudi koji su prihvatali ulogu korisnika u razvojnog timu na raspolaganju imaju vrlo limitirano vrijeme i ne mogu raditi stalno s razvojnim timom. Najčešće smatraju da je dovoljan doprinos pisanje zahtjeva i ne žele sudjelovati u testiranju.



MHC-PMS sustav

- Opis test casea za provjeru doze

Test 4: Provjera doze
Uzorak:
1. Broj u mg koji predstavlja jednu dozu lijeka. 2. Broj koji predstavlja broj doza u danu.
Testovi:
1. Test za ulaze kada je pojedinačna doza dobra, ali se prečesto ponavlja. 2. Test za ulaze kada je pojedinačna doza previška ili preniska. 3. Test za ulaze kada je pojedinačna doza*učestalost uzimanja previška ili preniska. 4. Test za ulaze kada je pojedinačna doza*učestalost uzimanja u dozvoljenim granicama.
Izlaz:
OK ili obavijest o greški ako je doza izvan dozvoljenih granica.



Automatizacija testiranja

- Automatizacija testiranja znači da se testovi pišu u obliku izvršnih komponenti prije nego se određeni zadataka implementira.
 - Testne komponente moraju biti samostalne, moraju simulirati slanje ulaznih podataka i trebaju provjeriti odgovara li rezultat navedenoj specifikaciji.
 - Automatizirane testne okoline (npr. Junit) su radne okoline koje omogućavaju jednostavno pisanje testova i slanje niza testova na izvršavanje.
- Kako je testiranje automatizirano, izvršavanje testova je pojednostavljeni.
 - Kada se doda nova funkcionalnost u sustav, može se pokrenuti test i otkriti problemi koje je eventualno unijela u sustav.



Problemi kod “test-first” razvoja

- Programeri preferiraju pisanje koda, a ne testova pa se ponekad pišu nepotpuni testovi koji ne provjeravaju neke iznimne situacije.
- Ponekad je teško napraviti inkrementalne testove (npr. kod složenih korisničkih sučelja problem je napraviti jedinične testove za dio koda koji se odnosi na prikaz na ekranu, kao i za prebacivanje između različitih formi u aplikaciji).
- Problem je ocijeniti u kolikoj mjeri testovi pokrivaju aplikaciju.
- Korisnik najčešće ne sudjeluje cijelo vrijeme u radu s razvojnim timom.



Programiranje u paru

- Kod XP-a programeri sjede zajedno i rade na istom dijelu koda što pomaže razvoju zajedničkog vlasništva nad kodom i širi poznavanje koda u timu.
 - Radom u paru dolazi do rasподjele znanja među članovima tima što umanjuje vjerojatnost propasti projekta u slučaju odlaska ključnih ljudi.
- Parovi se formiraju dinamički, tako da svi članovi tima mogu raditi jedni s drugima u paru za vrijeme trajanja projekta.
- Ovaj način rada se koristi kao neformalni proces provjere, s obzirom da s obzirom da svaku liniju koda provjerava više od jedne osobe, pa se otkrije veći broj grešaka.
- Ohrabruje refaktoriranje, pa cijeli tim ima koristi od toga.
- Mjerenja su pokazala da je programiranje u paru približno jednako produktivno kao i da dvoje ljudi radi neovisno na istom problemu.



Scrum

- Scrum je agilna metoda koja osigurava radni okvir (engl. *framework*) za organizaciju i planiranje agilnih projekata.
 - Ne zahtjeva korištenje nikakvih određenih metoda, već je naglasak na upravljanje iterativnim razvojem.
- U Scrumu postoje tri faze:
 - *Inicijalna faza* je okvirna faza planiranja, gdje se identificiraju generalni ciljevi projekta i dizajnira arhitektura softvera.
 - Nakon toga slijedi *niz sprintova*, a svaki od njih razvija jedan inkrement sustava.
 - *Faza zatvaranja projekta*, objedinjuje projekt, producira/dovršava se potrebna korisnička dokumentacija (npr. upute, pomoć u programu) i radi se procjena što se naučilo u projektu.



Scrum terminologija

Scrum naziv	Definicija
Product	softverski produkt koji razvija Scrum tim
Product owner	Osoba ili mala grupa ljudi čiji je posao identificirati osnovne osobine ili zahtjeve projekta, postaviti prioritete i kontinuirano pratiti <i>backlog</i> kako bi se osiguralo da projekt prati poslovne potrebe. To može biti kupac, <i>product manager</i> u firmi koja razvija softver ili netko drugi. Prate napravljen posao i pomažu u testiranju.
Product Backlog	Popis stvari koje tek treba napraviti. To mogu biti korisnički zahtjevi, priče ili opisi dodatnih stvari koje treba napraviti (definiranje arhitekture, pisanje dokumentacije, ispravljanje grešaka, ...)
Razvojni tim	Samoorganizirajuća grupa od 5 do 8 ljudi. Oni su odgovorni za razvoj softvera i svih drugih neophodnih dokumenata.



Scrum terminologija

Scrum naziv	Definicija
Sprint	Period od 2-4 tjedna u kojem se tim izolira i radi na razvoju.
Scrums/ daily stand-ups	Dnevni sastanak u kojem svi članovi tima rade pregled napretka i postavljaju prioritet za stvari koje trebaju biti napravljene taj dan.
ScrumMaster	ScrumMaster je osoba koja osigurava se efikasno prate Scrum procesi. Zadužen je za komunikaciju s osobama van tima i treba osigurati neometan rad tima. On nije <i>Project manager</i> .
Potencijalno isporučiv inkrement produkta	Inkrement softvera koji se isporučuje nakon sprinta. Trebao bi biti „potencijalno isporučiv”, što znači da je dovršen i ne zahtjeva nikakav dodatan rad (npr. testiranje) da bi se uključio u izvršnu verziju. U praksi se to ne može uvijek postići.
Velocity	Procjena koliko tim može napraviti unutar jednog sprinta. Mjerenje i razumijevanje ove mjere daje uvid u poboljšanje rada tima.



Osnovne uloge u Scrumu

- **Product Owner** mora osigurati da se razvojni tim bude fokusiran na proizvod koji se radi umjesto da se bave tehnički interesantnijim ali manje važnim stvarima.
- **The ScrumMaster** je stručnjak čiji je posao osigurati da tim na efikasan način koristi Scrum metodu. On je više poput trenera nego *Project manager*.
 - Organizira dnevne sastanke, provjerava popis zadataka koje još treba napraviti, bilježi odluke, provjerava napredak u odnosu na popis i komunicira s korisnicima i menadžmentom.



Scrum i sprintovi

- U Scrumu softver se razvija u tzv. sprintovima, tj. razvijaju se odabrane značajke koje se na kraju sprinta isporučuju korisnicima.
- Sprintovi su fiksne dužine, 2-4 tjedna.
- Za vrijeme sprinta tim ima dnevne sastanke, kako bi pratili napredak i osvježili listu nedovršenih stvari.
- Sprintovi bi trebali rezultirati s „isporučivim inkrementom sustava”, što znači da razvijeni softver treba biti gotov i spreman za *deploy*.

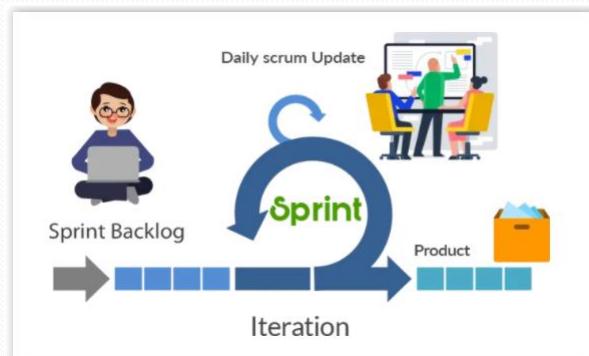


Scrum i sprintovi

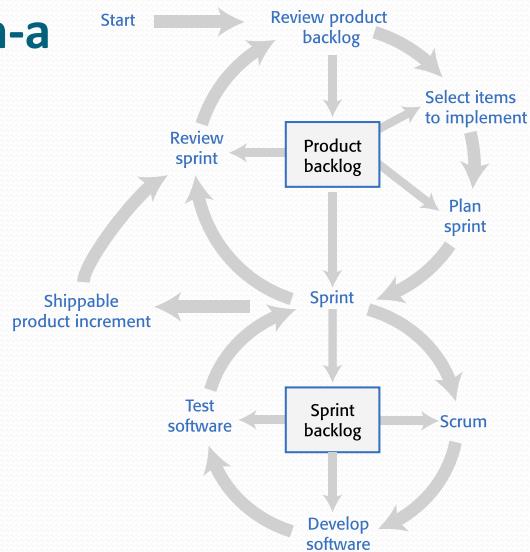
- Početna točka planiranja je popis zadataka koje još treba obaviti na projektu (engl. *backlog*).
- U fazi odabira članovi razvojnog tima u suradnji s korisnikom biraju funkcionalnosti koje će se razviti u sprintu.
- Jednom kada se dogovore funkcionalnosti, članovi tima sami raspodjele uloge u timu i kreće razvoj.
 - U tijeku ove faze tim se izolira od korisnika i organizacije, a sva se komunikacija prema vani ide preko *Scrum master-a*.
- *Scrum master* treba štititi razvojni tim od bilo kakvih vanjskih utjecaja i smetnji.
- Na kraju sprinta napravljeni posao se kontrolira i prezentira svim zainteresiranim stranama. Nakon toga kreće novi sprint.



Scrum procesi



Ciklusi scrum-a



Prednosti Scruma

- 1) Produkt se dijeli u niz manjih jasnih zadataka.
- 2) Nejasni zahtjevi ne zadržavaju napredak.
- 3) Cijeli tim ima uvid u projekt i kao posljedica toga komunikacija unutar tima je olakšana.
- 4) Korisnicima se na vrijeme isporučuju inkrementi, na osnovu kojih mogu dati komentare o radu softvera.
- 5) Uspostavlja se povjerenje između kupca i razvojnog tima i stvara se pozitivna okolina u kojoj svi očekuju uspjeh projekta.



Važne prakse Scruma

- ***Product backlog***
- ***Vremenski ograničeni sprintovi***
- ***Samoorganizirajući timovi***
 - Samoorganizirajući timovi samostalno donose odluke, rade na način da raspravljaju o idejama i dogovaraju se oko odluka.



Product Backlog

- Lista stavki koje treba napraviti kako bi se dovršio razvoj produkta.
- Stavke koje se nalaze na listi se nazivaju *product backlog items* (PBIs).
- Može sadržavati različite stavke kao što su značajke produkta koje treba implementirati, korisnički zahtjevi, osnovne razvojne aktivnosti, potencijalna poboljšanja, ...
- Redovito treba osvježavati prioritet stavki u *backlogu* kako bi se što prije implementiralo ono što ima najveći prioritet.



Primjer *backlog-a*

- 1. As a teacher, I want to be able to configure the group of tools that are available to individual classes. (feature)
- 2. As a parent, I want to be able to view my childrens' work and the assessments made by their teachers. (feature)
- 3. As a teacher of young children, I want a pictorial interface for children with limited reading ability. (user request)
- 4. Establish criteria for the assessment of open source software that might be used as a basis for parts of this system. (development activity)
- 5. Refactor user interface code to improve understandability and performance. (engineering improvement)
- 6. Implement encryption for all personal user data. (engineering improvement)

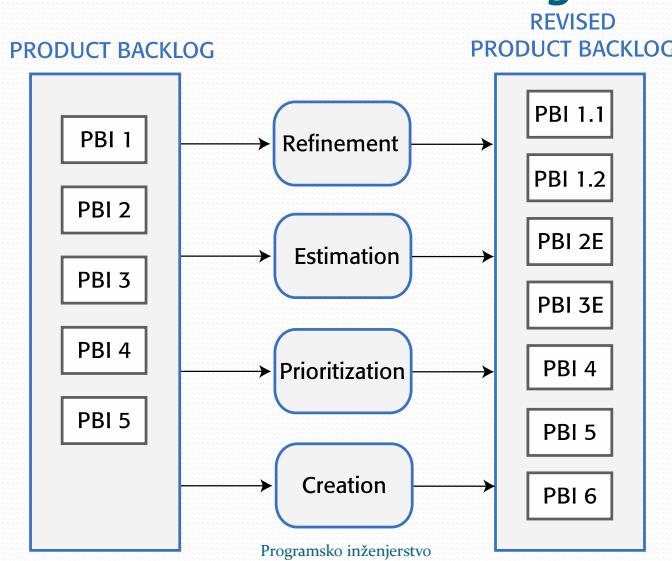


Stanja stavki u *backlog-u*

- **Spremno za razmatranje (engl. Ready for consideration)**
 - Ideje i opisi značajki na vrlo visokom nivou koje se razmatraju da se uključe u finalni produkt. To su nekakvi prijedlozi značajki, pa se mogu značajno promijeniti s vremenom ili ne moraju uopće biti uključene u konačni produkt.
- **Spremno za doradu (engl. Ready for refinement)**
 - Tim je odlučio da je neka značajka važna, te da će se implementirati. Uglavnom je jasno što se traži, međutim neke stvari treba malo doraditi.
- **Spremno za implementaciju (engl. Ready for implementation)**
 - Značajka je dovoljno detaljno opisana da tim može procijeniti potreban napor i implementirati je. Identificirane su ovisnosti o drugim značajkama.



Aktivnosti na *Product backlog-u*



Aktivnosti na *Product backlog-u*

- **Dorada**

Postojeće stavke se analiziraju i dorađuju kako bi se stvorile stavke s dovoljno detalja. Ovo može rezultirati kreiranjem novih stavki u *backlog-u*.

- **Procjena**

Tim procjenjuje količinu rada koji je potreban za implementaciju značajke i zbraja ukupno vrijeme potrebno za realizaciju značajki koje se planiraju uključiti u sprint.

- **Stvaranje**

Dodaju se nove stavke u *backlog*. To mogu biti nove značajke, promjene postojećih značajki, potencijalna poboljšanja, procesne aktivnosti poput procjene razvojnih alata koji će se koristiti, ...

- **Postavljanje prioriteta**

Mijenja se raspored stavki kako bi se nove informacije i promjene uzele u obzir.



Metrika procjena

- **Potreban napor (engl. effort)**

- Može biti izražen u čovjek-danima, čovjek-satima, a označava koliko dana (sati) treba osobi da implementira neku stavku. Ovo nije kalendarsko vrijeme, jer ukoliko više ljudi radi isto kalendarski gledano produkt će ranije biti gotov.

- **Bodovi priče (engl. Story points)**

- Bodovi priče su proizvoljna procjena napora uključenog u implementaciju stavke. Pri procjeni se u obzir uzima veličina zadatka, njegova složenost, potrebna tehnološka rješenja i eventualne „nepoznate“ karakteristike.
- Radi se o relativnoj mjeri, pri čemu se tim dogovori o bodovima priče za neku osnovnu stavku, a onda je ostale procjenjuju u odnosu na nju (veće/manje, više ili manje složene, ...).

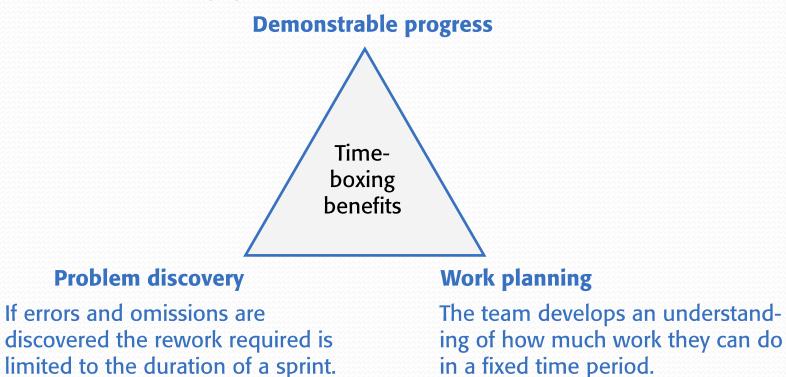


Vremenski ograničeni sprintovi

- Produkt se razvija kroz niz sprintova, od kojih svaki isporučuje inkrement sustava
- radi se o kratkotrajnim aktivnostima (2-4 tjedna) i odvijaju se između predefiniranog početnog i krajnjeg datuma. Vremenski su ograničeni što znači da sprint prestaje kada istekne vrijeme bez obzira je li posao obavljen do kraja.
- Za vrijeme sprinta, članovi tima rade na stavkama iz *backlog-a*.

Prednosti vremenskog ograničenja sprinta

There is a tangible output (usually a software demonstrator) that can be delivered at the end of every sprint.



Aktivnosti sprinta

- **Planiranje sprinta**

Odabiru se stavke koje će se napraviti u sprintu, po potrebi dorađuju i stvara se sprint *backlog*. Ovo ne bi trebalo trajati više od jednog dana, na početku sprinta.

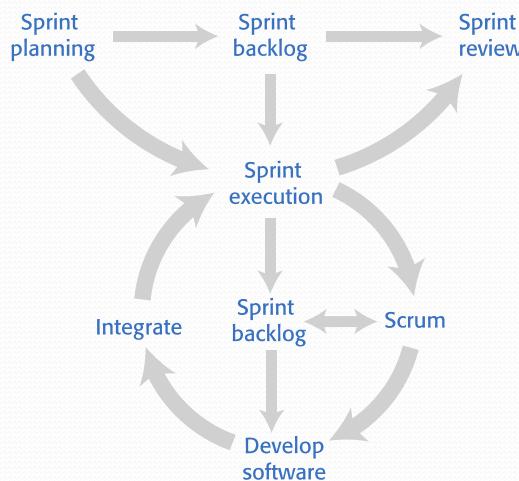
- **Odrađivanje sprinta**

Tim radi na implementaciji stavki iz *backlog-a* sprinta. Eventualno nedovršene stavke se vraćaju u *product backlog* na kraju sprinta.

- **Recenzija sprinta (engl. Sprint reviewing)**

Obavljeni posao u sprintu tim analizira zajedno s vanjskim sudionicicima. Tim komentira što je bilo dobro, što je bilo loše za vrijeme sprinta i na koji način se može poboljšati rad tima.

Aktivnosti sprinta



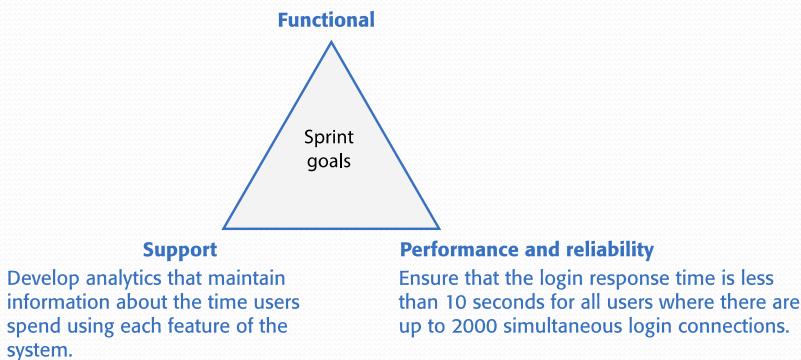
Planiranje sprinta

- Određuju se ciljevi sprinta koji se mogu odnositi na funkcionalnosti softvera, podršku, performanse, pouzdanost, ...
- Odabire se lista značajki iz product backloga koje će se implementirati.
- Kreira se sprint *backlog*
 - Detaljnija verzija *product backlog-a* koja opisuje što će se napraviti za vrijeme sprinta.



Ciljevi sprinta

Implement user roles so that a user can select their role when they login to the system



Scrum

- Kratki, dnevni sastanak koji se održava na početku dana.
- Za vrijeme scrum-a svi članovi tima opisuju svoj napredak od prethodnog sastanka, probleme koji su se pojavili i planove za taj dan. Kao rezultat toga svaki član tima zna što se događa i ako se pojave problemi mogu organizirati posao kako bi ih što prije riješili.
- Sastanci moraju biti kratki i fokusirani, pa se često organiziraju kao „stojeći” sastanci bez stolica u sobi.
- Za vrijeme scrum-a analizira se *sprint backlog*, uklanjaju se napravljene stavke, po potrebi dodaju nove i tim odlučuje tko će na čemu raditi taj dan.



Agilne aktivnosti

- Scrum ne zahtjeva korištenje agilnih aktivnosti, međutim praksa je pokazala da je dobro koristiti:
 - **Automatizaciju testova**
Koliko god je moguće testiranje produkta treba biti automatizirano i dok se razvija produkt razvija se i niz automatskih testova koji se mogu pokrenuti bilo kada.
 - **Kontinuirane integracije**
Svaki put kada neko izmjeni postojeću komponentu ili se napravi nova, te komponente se odmah moraju integrirati s ostalim komponentama kako bi se stvorila nova verzija sustava. Sustava se tada testira i provjeravaju nekakve neželjene interakcije među komponentama.



Checklist dovršenosti koda

- **Recenziran**
Kod koji se predaje treba pregledati neki drugi član tima koji provjerava da kod odgovara dogovorenim standarde kodiranja, razumljiv je, komentari su prikladni, te je refaktoriran ako je to bilo potrebno.
- **Jedinično testiran (engl. Unit tested)**
All unit tests have been run automatically and all tests have executed successfully.
- **Integriran**
Kod je uspješno integriran s postojećim i nisu se pojavile nikakve integracijske greške.
- **Testirana integrirana verzija**
Svi automatski testovi su pokrenuti na integriranoj verziji i uspješno su izvršeni.
- **Prihvaćen**
Test prihvata je izvršen i *product owner* ili razvojni tim su potvrdili da su stavke iz *product backlog-a* dovršene.

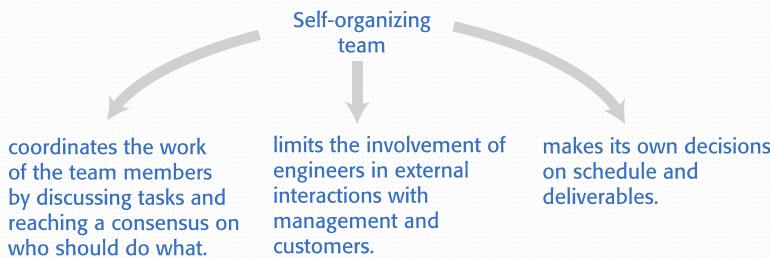


Recenzija sprinta

- Na kraju svakog sprinta održuje se recenzija sprinta (*engl. Sprint review*) na kojoj su prisutni svi članovi tima,
- Cilj recenzije je:
 - Odrediti je li sprint postigao svoje ciljeve,
 - Izdvojiti eventualne nove probleme i stavke koji su se pojavili za vrijeme sprinta.
 - Potaknuti tim da analizira svoj rad i predlože na koji način ga mogu poboljšati.
- *Product owner* jedini može odrediti zapravo je li dosegnut cilj sprinta i treba potvrditi za svaku stavku je li gotova.
- Recenzija sprinta treba uključivati i recenziju procesa, u kojoj tim komentira na vlastiti način rada i kako se Scrum koristi.
 - Cilj je pronaći načine kako poboljšati rad tima i produktivnost.



Samoorganizirajući timovi



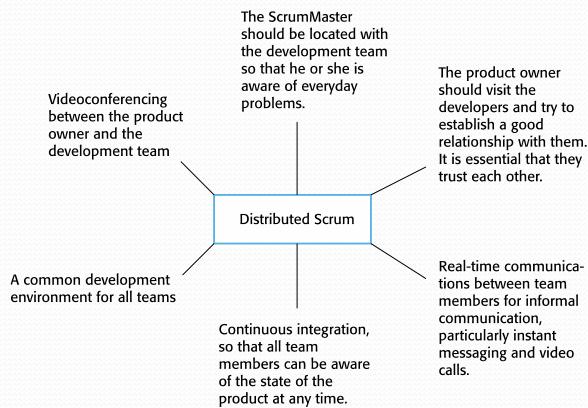
Scrum timovi

- Idealna veličina Scrum tima je od 5 do 8 ljudi.
 - Članovi tima trebaju obavljati različite zadatke, što zahtjeva ljude različitih znanja i sposobnosti (umrežavanje, dizajn baze podataka, dizajn sučelja, ...)
 - Često su razine znanja članova tima različite.
 - Ti od 5 -8 ljudi je dovoljno velik da se pokrije različita područja, a dovoljno malen da članovi mogu jednostavno komunicirati i dogovarati prioritete tima.
- Prednost takvog tima da je dovoljno kohezivan i lako se prilagođava promjenama.
 - Tim, a ne pojedinci preuzimaju odgovornost za posao, pa je lakše nositi se s odlaskom/dolaskom članova tima.
 - Dobra komunikacija znači da članovi tima neizostavno nauče iz područja drugih članova tima.

Koordinacija tima

- Pretpostavka je da su članovi tima smješteni na istoj lokaciji, rade u istoj sobi tako da mogu neformalno komunicirati.
 - Dnevni sastanci znače da svi članovi tima znaju što je napravljeno i na čemu rade ostali.
- Međutim dnevni sastanci kao koordinacijski mehanizam je baziran na dvije pretpostavke koje ne moraju biti uvijek zadovoljene:
 - Scrum sastanak podrazumijeva da će tim činit zaposlenici s punim radnim vremenom i da svi članovi tima dijele isti radni prostor. U stvarnosti to ne mora biti istina (rad s dijelom radnog vremena, rad s neke druge lokacije)
 - Scrum sastanak podrazumijeva da svi članovi tima mogu prisustvovati dnevnom sastanku, međutim netko možda ima fleksibilno radno vrijeme ili radi na više projekata istovremeno.

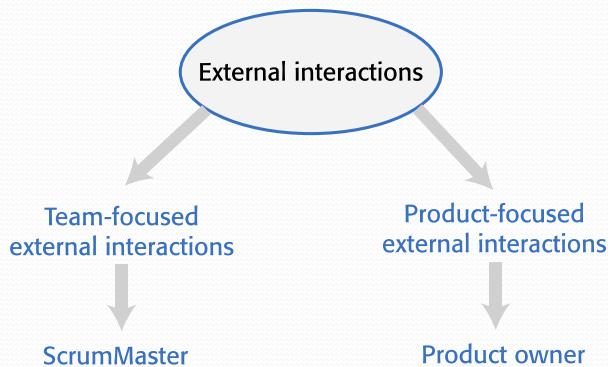
Distribuirani Scrum



Vanske interakcije

- Pod vanjskim iteracijama se podrazumijevaju interakcije koje članovi tima imaju s ljudima van tima.
- U Scrum-u je osnovna ideja da se članovi tima fokusiraju na razvoj i samo *ScrumMaster* i *Product Owner* su uključeni u vanske interakcije.
- Cilj ovoga je da tim može slobodno raditi na razvoju bez vanjskog utjecaja i ometanja.

Upravljanje vanjskim interakcijama

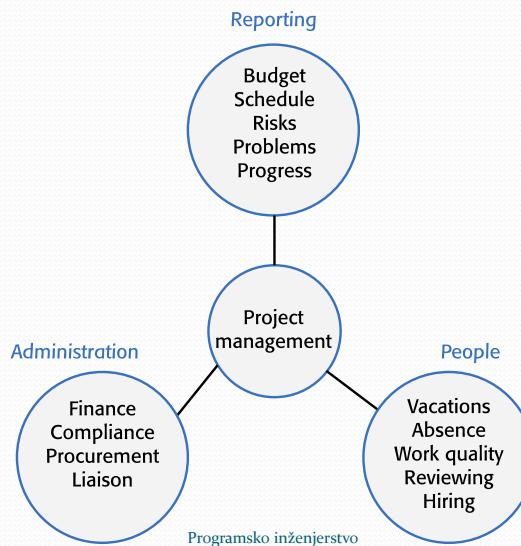


Upravljanje projektom

- U većini kompanija koje se bave razvojem softvera potrebno je da razvojni timovi podnose izvještaj o napretku.
- Samoorganizirajući timovi se dogovaraju tko u timu će obavljati tu ulogu.
 - To je stalna uloga i nije dobro da se članovi tima izmjenjuju kako bi se zadržao kontinuitet u komunikaciji.
- Ljudi koji su osmislili Scrum metodu nisu predviđjeli da ScrumMaster ima odgovornosti i *project manager-a*.
 - U velikom broju slučajeva ScrumMaster ipak preuzima i tu ulogu.
 - Oni znaju što se događa na projektu i na najboljoj su poziciji da pruže sve potrebne informacije, prate plan rada i napredak.



Odgovornosti *project manager-a*



Više timski Scrum

- Replikacija uloga
 - Svaki tim ima *Product Owner-a* za dio koji treba napraviti i *ScrumMaster-a*.
- *Product architects*
 - Svaki tim bira svog *Product architect-a* i svi oni zajedno rade na arhitekturi čitavog sustava.
- Dogovor o datumima isporuke
 - Svi timovi zajedno dogovaraju datum isporuke tako da se može isporučiti i pokazati što više komponenti sustava.
- Scrum Scrumova
 - Postoji i dnevni Scrum svih Scrumova na kojem predstavnik svakog tima predstavlja napredak i što se planira napraviti.



Agilni razvoj softvera

Kraj