



SVEUČILIŠTE U ZAGREBU



Fakultet
elektrotehnike i
računarstva

Diplomski studij

Razvoj komunikacijske programske podrške

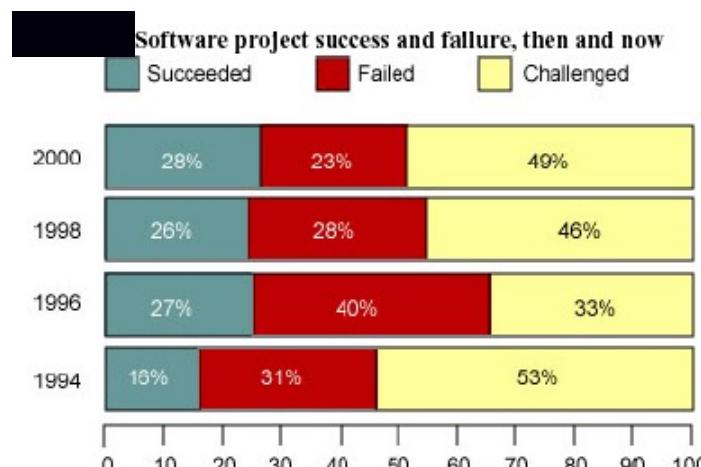
Ak. God. 2021/22.



Agilne metodologije

Rizik – osnovni problemi razvoja softvera

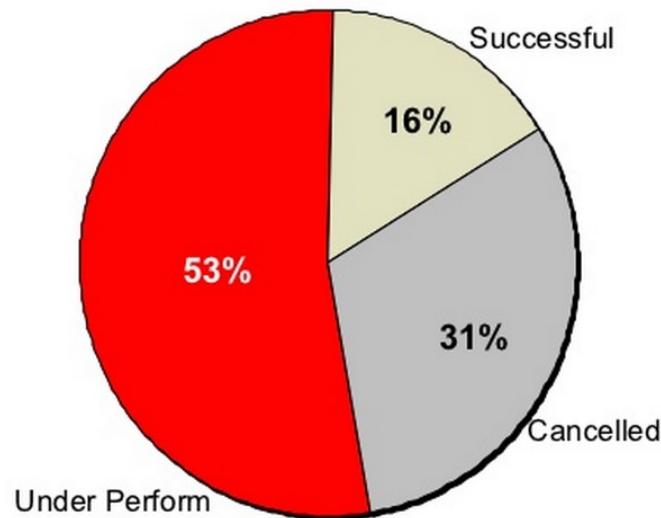
- Probijanje rokova
- Odustajanje od projekta
 - nakon probijanja rokova
- Nabacana arhitektura sustava
- Puno grešaka
 - nekorisnost sustava
- Sustav ne rješava probleme poslovne logike
- Promjene poslovne logike
- Kriva svojstva programa
 - zabavno, ali nekorisno
- Smjena programera



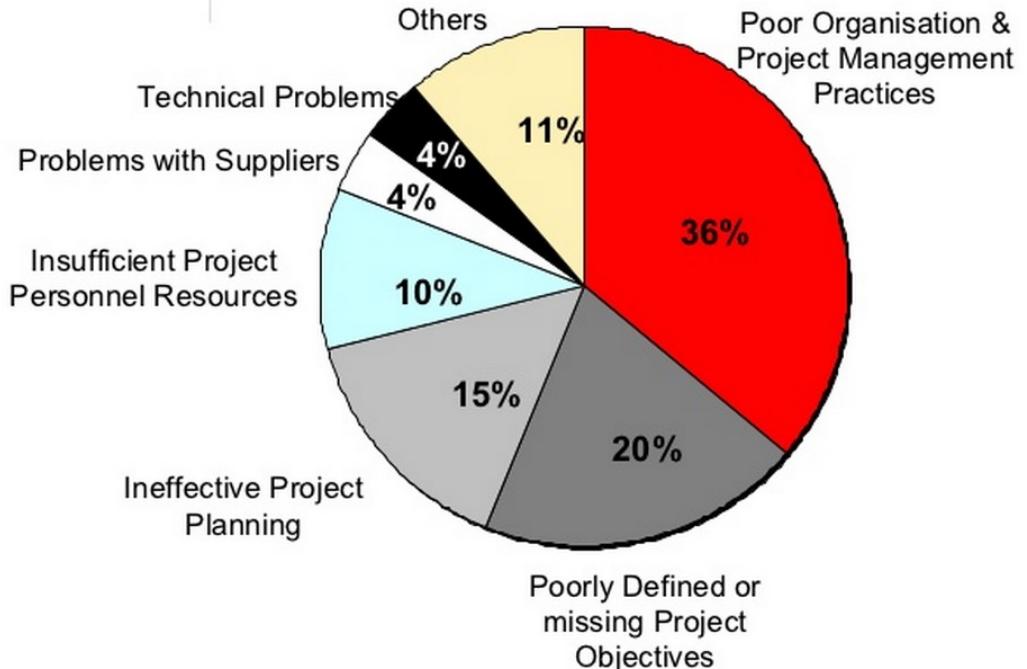
Izvor: *Standish Group*
članak "Keeping CHAOS quiet"

Uzroci...

Large Programs Success Rates

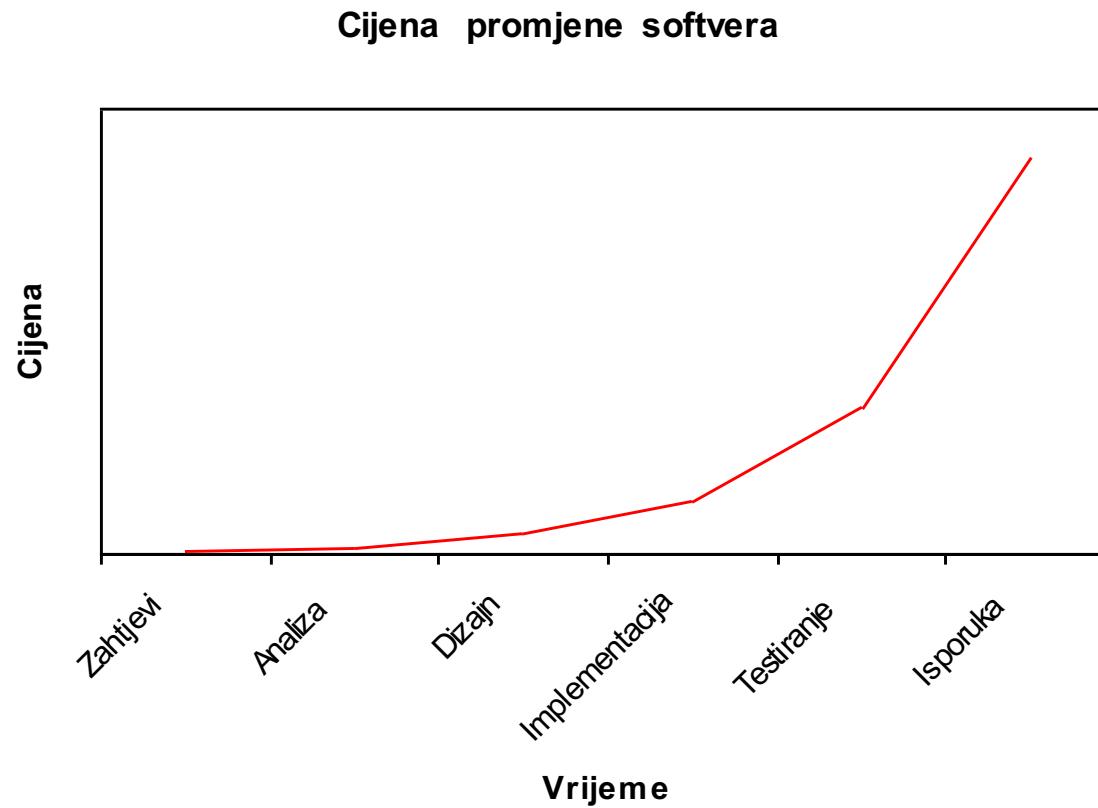


Why Projects / Programs Fail



Source: Standish Group International, Survey from 2500 personnel attending project management training

Cijena promjene softvera



Agilne metodologije – vrijednosti

- Novi sustav vrijednosti:
 - Osobe i interakcije umjesto procesa i alata
 - Softver koji radi umjesto opširne dokumentacije
 - Zajednički rad s kupcem umjesto pregovaranja preko ugovora
 - Reagiranje na promjene umjesto striktnog pridržavanja plana
- *Manifesto for Agile Software Development*
- *Agile Alliance*
 - Neprofitna organizacija koja traži i promovira znanje i rasprave o svim agilnim metodama
 - <http://agilealliance.org>

Agilne metodologije

- Ekstremno programiranje – **XP (Extreme Programming)**
 - <http://www.xprogramming.com>
 - <http://www.extremeprogramming.org>
- **Open Source**
 - The Cathedral and the Bazar – <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar>
- **Scrum**
 - <https://www.scrum.org/>
- **Feature Driven Development**
 - <http://www.featuredrivendevolution.com>
- **LEAN**
 - <https://kanbanize.com/lean-management/what-is-lean-management>
- **Kanban**
 - <https://kanbanize.com/kanban-resources/getting-started/what-is-kanban>



SVEUČILIŠTE U ZAGREBU



Fakultet
elektrotehnike i
računarstva

Diplomski studij

Ak. God. 2021/22.



Razvoj komunikacijske programske podrške

Agilne metodologije – XP (Extreme
Programming)

Što je XP?

- XP je metodologija za razvoj softvera gdje su:
 - timovi male ili srednje veličine
 - zadaci nejasni ili se često mijenjaju.
- Ekstremnost:
 - razumni principi i tehnike se dovode do ekstremne razine
- Novosti:
 - Korištenje razumnih principa i tehnika u jednoj metodologiji
 - Osiguravanje da se svi principi i tehnike podržavaju u najvećem mogućem obliku
 - Detaljno provođenje istih

Što nije XP?

- XP nije gotovi recept za izradu softvera
- Sve se mijenja – zahtjevi, dizajn, poslovna logika, tehnologija, ljudi u timu, ...
- Problem koji se rješava je nesposobnost prilagođavanja promjeni
- “VOŽNJA AUTOMOBILA”
- Kako se prilagoditi promjeni?

Praktične tehnike (1)

- Metafora – zajednička terminologija
- Planiranje verzija
- Mala i česta izdanja
- Jednostavan dizajn
- Testiranje (*test first development*)
- Refaktoriranje
- Programiranje u paru
- Zajedničko vlasništvo nad kodom
- Stalna integracija
- 40-satni radni tjedan
- Naručitelj u timu
- Standardi pisanja koda

Praktične tehnike (2) – metafora

- metafora predstavlja pojednostavljenu sliku sustava
- jednostavan opis sustava koji je svima jasan
- ideja je da programeri preko metafore dobiju sliku gdje se njihov rad uklapa u sustav
- definira i zajednički jezik (termini)

Praktične tehnike (3) – planiranje verzija

- obavlja se jednom za svaku verziju
- postupak kojim se naručiteljeve priče pretvaraju u plan izvođenja istih
- stvara se grubi plan izdanja verzije koji se kasnije dorađuje (za vrijeme implementacije) – važnije stvari se rade prve
- ključno je da se naručitelj i programer pridržavaju svojih obaveza:
 - Programer:
 - Procjena vremena za pojedinu priču
 - Tehnološki trošak za korištenje pojedine tehnologije (učenje, ...)
 - Tehnološki rizik svake priče (nešto treba biti napravljeno prije,...)
 - Određuje raspored implementacije pojedine priče unutar jedne iteracije
 - Naručitelj:
 - Određivanje datuma izdanja verzije
 - Važnost pojedine priče (za naručitelja)
 - Određivanje prioriteta (rasporeda) implementacije pojedine priče unutar izdanja verzije

Praktične tehnike (4) – mala i česta izdanja

- česta izdanja omogućuju da korisnici softvera mogu što prije vidjeti kako softver radi
- korisnici mogu dati svoje komentare i želje koje se koriste za planiranje sljedećih verzija
- vrijednost softvera se daje naručitelju što je prije moguće
- naručitelj može prekinuti razvoj softvera
 - kada je zadovoljan s funkcionalnošću koja mu je dana
 - kada nije zadovoljan napretkom
- u slučaju prekida suradnje naručitelj ima najvažnije funkcije ugrađene

Praktične tehnike (5) – jednostavan dizajn

- dva osnovna principa:
 - nemojte raditi ono što nije zadatak
(YANGI – *You aren't gonna need it*)
 - napravite najjednostavniju stvar koja će raditi
- ne preporuča se prvo sve dizajnirati pa onda implementirati
- nikada se ne može znati što će još trebati ugraditi u sustav
- najjednostavnija stvar koja će raditi ponekad može biti i jako komplikirana
- alati i UML se koriste samo kada pomažu da se nešto napravi – ne trošiti vrijeme na uređivanje dijagrama!

Praktične tehnike (6) –

- svaki put kada programer promijeni kod potrebno je znati je li se nešto pokvarilo
- pišući prvo testove, a onda kod ima za posljedicu:
 - najcjelovitiji skup testova koji je moguć
 - povećavaju sigurnost programera u kod
 - najjednostavniji kod koji će raditi
 - omogućava jednostavnije refaktoriranje
 - vidljiva vizija namjere kod da
 - jednostavnije razumijevanje i refaktoriranje
- testovi su podijeljeni na dva dijela:
 - programerske (*unit*)
 - provjeravaju ispravnost rada sustava
 - uvijek moraju raditi
 - daju sigurnost programerima
 - objašnjavaju kako se kod koristi
 - moraju biti automatizirani i brzo se izvoditi
 - naručiteljske (*acceptance*)
 - provjeravaju da li sustav radi ono što je naručitelj htio

Praktične tehnike (7) – refaktoriranje

- tehnika poboljšavanja koda bez promjene funkcionalnosti
- nikako se ne smije istovremeno refaktorirati i pisati kod za novu funkciju
- refaktoriranje se izvodi
 - prije implementiranja neke funkcije
 - promijeniti kod tako da implementacija nove funkcije bude jednostavnija
 - nakon implementiranje neke funkcije
 - pokušava se pojednostavnići kod koji je napisan
- refaktoriranje je vrlo teško izvesti ako nema testova koji provjeravaju ispravnost koda
- postoje besplatni alati koji olakšavaju refaktoriranje

Praktične tehnike (8) – programiranje u paru

- dva programera istovremeno koriste jedno računalo
- prije pisanja k da raspravljaju kako nešto napraviti
- dvije uloge koje se često mijenjanju:
 - “driver” – piše po tipkovnici
 - navigator – provjerava napisano i razmišlja o mogućim problemima
- produktivnost para je veća nego produktivnost svakog pojedinačno jer programiranje nije samo tipkanje k da
- prednosti:
 - sve dizajnerske odluke uključuju dva programera
 - barem dva programera su upoznata sa svakim dijelom sustava
 - manja je vjerojatnost da oba programera izostave pisanje testova
 - zamjena parova širi znanje u timu
 - k d je uvijek pregledan od strane barem jednog programera – povećava kvalitetu

Praktične tehnike (9) – zajedničko vlasništvo

- svaki programer ima pravo promijeniti bilo koji dio koda
- zajedničko vlasništvo nad kodom znači da su svi odgovorni za taj kód
- “*You break it, you fix it.*”
- zahtjeva ekstremnu disciplinu – pogotovo kada nemate vremena
- testovi pomažu programeru da promjeni kod koji mu nije poznat, a da je siguran da nešto nije pokvario
- testovi pokazuju kako se taj dio koda koristi
- izbjegava se čekanje na nekoga da promjeni dio koda za koji je zadužen
- poboljšava se kvaliteta koda
 - zna se da će kad tad netko pogledati taj kod vidjeti kako je napisan
- potrebno je definirati formatiranje koda – pomažu alati

Praktične tehnike (10) – stalna integracija

- stalna integracija ne znači da se treba integrirati svaku sekundu već da se često integrira
- integracija promjena se treba vršiti nekoliko puta dnevno
- automatizirano mora biti kreiranje svakog dijela sustava
- integrirati se ne smije dok svi programerski testovi ne prorade
- Integrirani kod se stavlja na poslužitelj koji služi kontroli verzija (*CVS, SourceControl, ClearCase*)

Praktične tehnike (11) – 40-satni radni tjedan

- član tima treba biti:
 - ujutro – svjež i oran za posao
 - navečer – umoran i zadovoljan
 - petkom nakon posla – zadovoljan tako da vikend provede ne razmišljajući o poslu
 - ponedjeljkom prije posla – oran i s puno ideja
- menadžment treba osigurati okolinu
 - ne zamarati tim s administracijom
 - osigurati okolinu bez stresa, pritiska i stalnog umora
- Kako?
 - poslati članove tima nakon 8 radnih sati kući
 - ne dozvoliti prekovremen rad više od jednog tjedna

Praktične tehnike (12) – naručitelj u timu

- težak organizacijski problem
 - treba procijeniti što je korisnije:
 - naručitelj (korisnik) radi svoj posao ili
 - naručitelj (korisnik) pomaže timu u izradi softvera
- odgovara na nejasnoće programerima u trenutku kada je nešto nejasno
- naručitelj bi trebao biti korisnik sustava koji se programira
- treba znati predstaviti potrebe korisnika sustava
- određuje prioritet pojedinih funkcija
- piše priče koje predstavljaju korištenje funkcija
- piše naručiteljske testove koji provjeravaju funkcioniranje sustava prema pričama
- u pisanju testova mu pomaže programer

Praktične tehnike (13) – standardi pisanja koda

- programeri stalno gledaju i proučavaju tuđi kod
- ako nema sporazuma o tome kako se kod piše
 - troši se vrijeme na razumijevanje koda
- standard ne smije trošiti puno programerskog vremena
- treba poboljšati komunikaciju
- Problem: svaki programer je naviknut na svoj stil pisanja koda
- standard mora biti dobrovoljno prihvaćen od svih članova tima
- ako se članovi tima ne mogu dogovoriti onda je najbolje da se preuzme standard koji je propisan od neke neutralne organizacije
- postoje alati koji pomažu formatiranju koda

Faze projekta

- Istraživanje – ideje, što, zašto, kako?
- Planiranje – izdavanja verzija (*release*)
- Iteracije
 - pisanje zadataka (priča)
 - procjena zadataka
 - sortiranje zadataka (prioritet, tehnički rizik)
 - planiranje iteracije (i verzija)
 - pisanje k da
- Izdavanje verzije (*release*)
 - performanse, opterećenje sustava
- Održavanje
 - dodavanje nove funkcionalnosti, ispravci grešaka
- Kraj – završetak projekta

Pisanje koda

- Postupak:
 1. Dizajniranje rješenja
 - u paru se raspravlja o mogućim rješenjima
 - moguće pisanje dijagrama (UML)
 2. Pisanje programerskog testa
 - prvo se napišu tipični testovi koji provjeravaju funkciju koda
 - zatim se pišu testovi za provjeru netipičnih slučajeva (rubni uvjeti, greške, ...)
 3. Pisanje koda
 - kod je gotov kada svi testovi za taj kod prođu
 4. Integracija
 - integracija s ostatkom koda (čitav softver)
 - gotovo je onda kada svi testovi prođu
- u postupku 2 se piše samo jedan test pa se napiše kod (postupak 3) pa se ponovno vraća na postupak 2, ...
- kada je funkcija kompletno implementirana onda se prelazi na 4

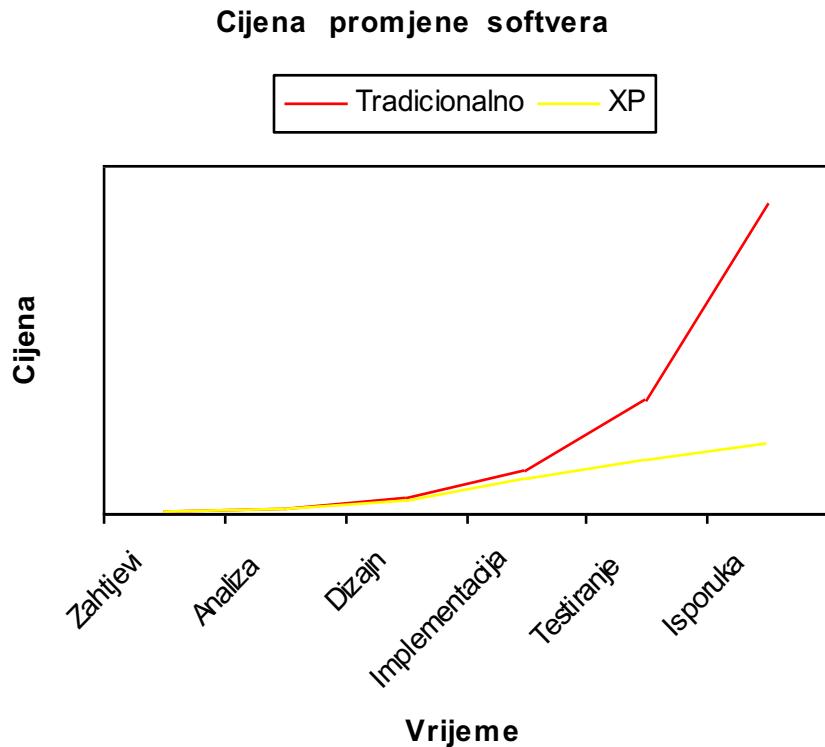
Kako XP rješava rizik? (1)

- **Probijanje rokova**
 - kratke iteracije
- **Odustajanje od projekta**
 - određivanje najmanje verzije koja ima smisla za posao
- **Sustav postaje “gorak”**
 - konstantno testiranje
- **Puno grešaka**
 - testovi – programerski i korisnički
- **Sustav ne rješava probleme poslovne logike**
 - naručitelj dio tima

Kako XP rješava rizik? (2)

- **Promjena poslovne logike**
 - kratke iteracije nakon kojih se radi planiranje
- **Kriva svojstva programa**
 - prioriteti funkcionalnosti kod planiranja
- **Smjena programera**
 - programer bira zadatke,
 - određuje trajanje pojedinog posla,
 - novi članovi postepeno preuzimaju odgovornost i zadatke

Cijena promjene softvera



- Jednostavan dizajn bez dodatne fleksibilnosti
- Automatizirani testovi – pouzdanost da se ne pokvari neka bitna funkcionalnost sustava
- Uvježbanost u mijenjanju sustava – nema straha od promjene
- Posljedica je ublažavanje krivulje

Razlike između XP-a i ostalih metodologija

- Rane, konkretnе i stalne povratne informacije
 - kratke iteracije
- Inkrementalno planiranje
- Fleksibilnost rasporeda implementacije promijenjenih funkcionalnosti
 - posljedica promjena zahtjeva
- Oslonac na automatsko testiranje koje prati napredovanje
 - evolucija sustava
 - rano otkrivanje pogrešaka
- Oslonac na oralnu komunikaciju, testove i kod
 - jasnoća arhitekture
- Evolucijski dizajn kroz životni vijek sustava
- Prisna suradnja između programera

Kada XP nije dobar?

- kad postoji velik otpor XP načinu programiranja
- nefleksibilna okolina
- kad se treba stvarati velika količina dokumentacije
- u okolini gdje je prekovremen rad uobičajan
- kad ljudi u timu ne žele ili ne mogu komunicirati (uobraženost, fizička udaljenost, rad na daljinu)
- veliki timovi
- okolina u kojoj se povratne informacije dugo čekaju

Literatura

- Kent Beck: "*Extreme Programming Explained – Embrace Change*", 2000., Addison–Wesley
- David Astles, G. Miller, M. Novak: "*A Practical Guide to Extreme Programming*", 2002., Prentice Hall
- James Newkrik, R. C. Martin: "*Extreme Programming in Practice*", 2001., Addison–Wesley
- Eric M Bruke, B. M. Coyner: "*Java Extreme Programming Cookbook*", 2003. O'Reilly
- Kent Beck: "*Test-Driven Development by Example*", 2002., Addison–Wesley
- Johannes Link: "*Unit Testing in Java*", 2003., Morgan Kaufmann
- Roy W. Miller: "*Demystifying Extreme Programming: XP Distilled*", parts 1-3, <http://www.ibm.com/developerworks/java/library/j-xp/>
- Jean-Guy Schneider, Lorraine Johnston: "*eXtreme Programming at Universities: An Educational Perspective*", ICSE 2003, stranice: 594 – 599
- Laurie Williams, R. Upchurch: "*Extreme programming for software engineering education?*", Frontiers in Education Conference, 2001., T2D-12-17 vol.1
- <http://www.xprogramming.com>
- <http://www.extremeprogramming.org>



SVEUČILIŠTE U ZAGREBU



Fakultet
elektrotehnike i
računarstva

Diplomski studij

Razvoj komunikacijske programske podrške

Ak. God. 2021/22.



Agilne metodologije - SCRUM

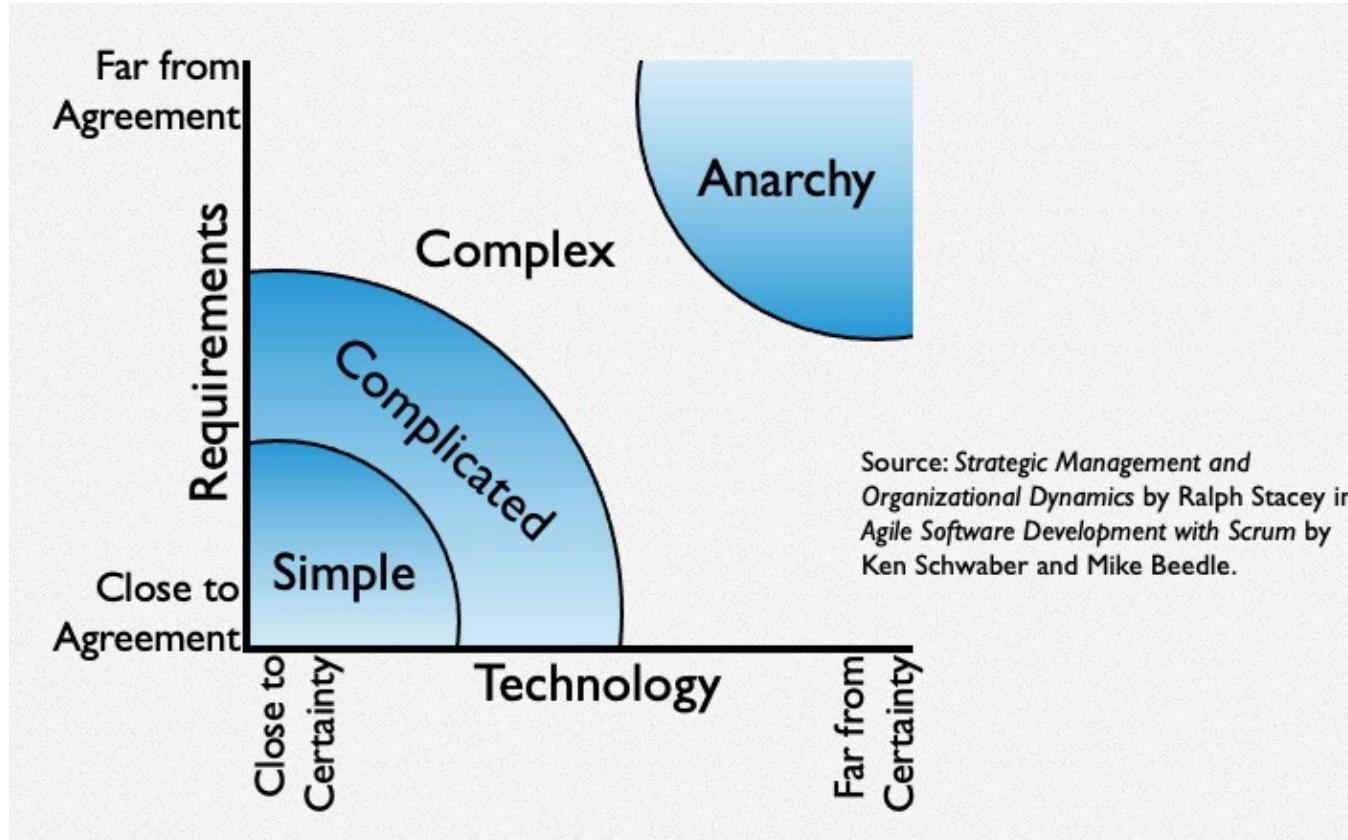
Scrum ukratko

- Scrum je agilna metodologija koja omogućuje da se usredotočimo na pružanje najveće poslovne vrijednosti u najkraćem vremenu
- Omogućuje nam brzi i česti pregled stvarnog softvera koji radi (svaka dva tjedna do mjesec dana).
- Posao definira prioritete. Timovi se samoorganiziraju kako bi odredili najbolji način za rješavanje zahtjeva najvišeg prioriteta.
- Svaka dva tjedna do mjesec dana svatko može vidjeti stvarni softver koji radi i odlučiti ga objaviti kakav jest ili ga nastaviti poboljšavati za još jedan sprint.

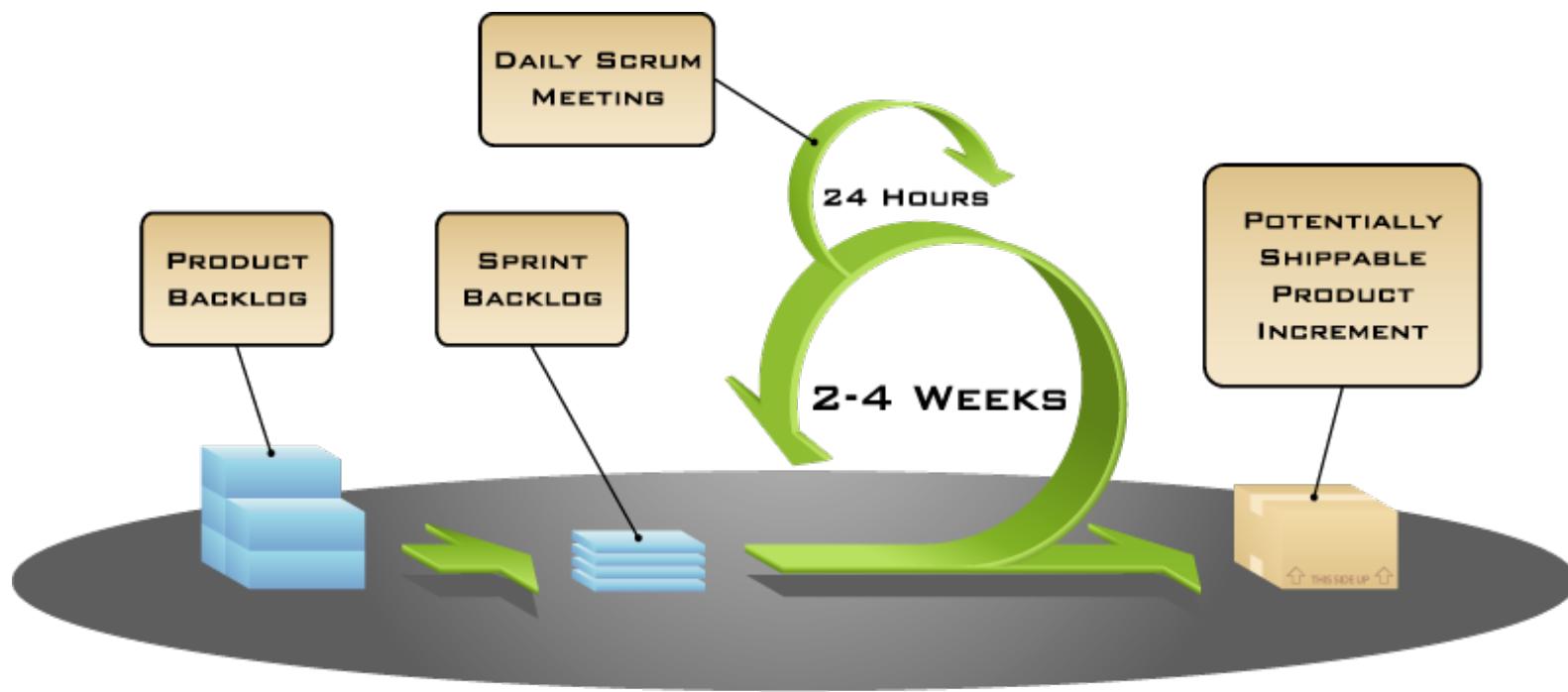
Glavna svojstva

- Agilna metodologija / proces
- Timovi se samoorganiziraju
- Proizvod napreduje u nizu mjesecnih "sprintova"
- Zahtjevi se bilježe kao stavke na popisu koje treba implementirati
- Nisu propisane posebne inženjerske prakse
- Koristi jednostavna pravila za stvaranje agilnog okruženja za isporuku projekata

Projekti i "šum"



Kako izgleda cijeli proces?



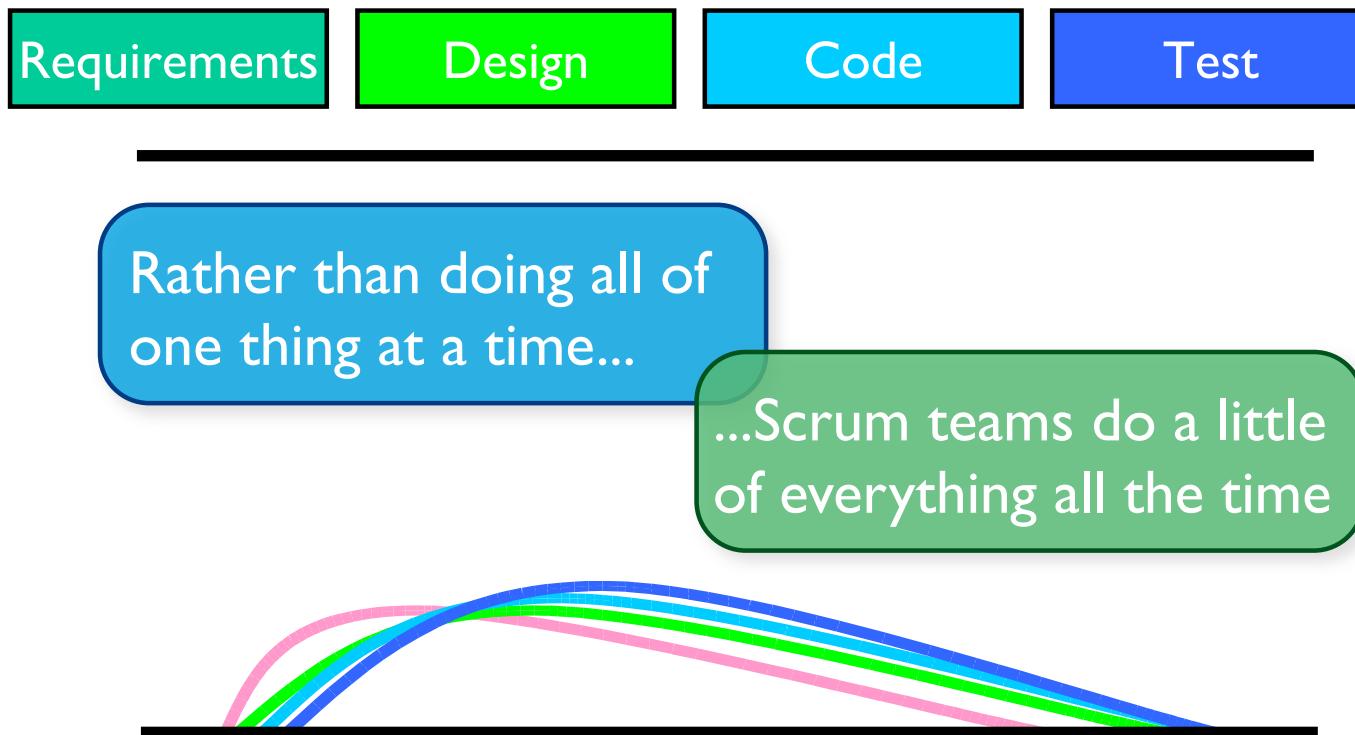
COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

www.mountaingoatsoftware.com/scrum

Sprintovi

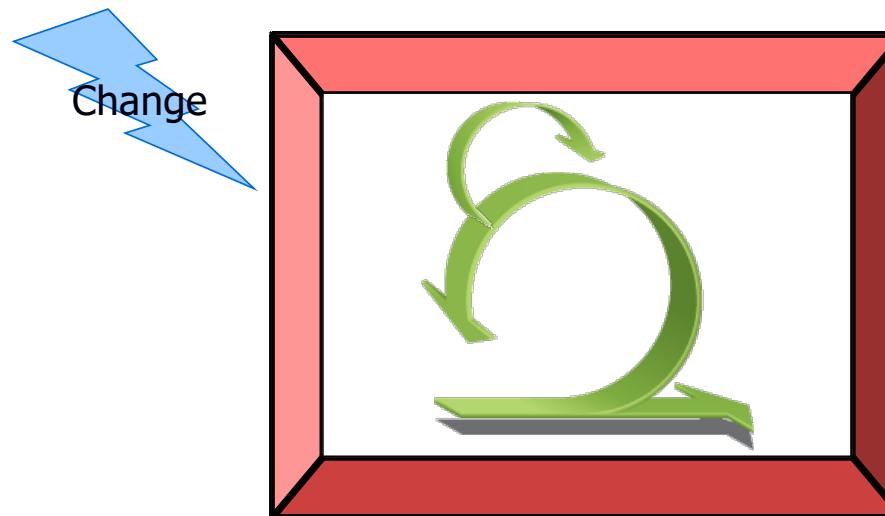
- Scrum projekti napreduju u nizu "sprintova"
 - Slično iteracijama kod XP-a
- Uobičajeno trajanje je 2–4 tjedna ili najviše mjesec dana
- Stalno „trajanje“ dovodi do boljeg ritma
- Proizvod se dizajnira, implementira i testira tijekom sprinta

Slijedni vs preklapajući razvoj



“The New New Product Development Game” by Takeuchi and Nonaka. *Harvard Business Review*, January 1986.

Nema promjena za vrijeme sprinta!



- Važno je planirati duljinu sprinta tako da budete sigurni da neće biti zahtjeva za promjenama tijekom izvođenja sprinta!

Scrum

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

Scrum

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

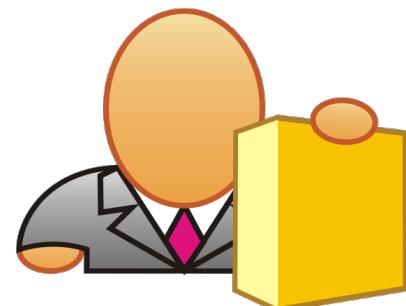
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

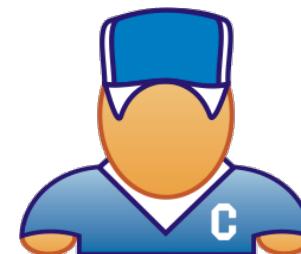
Product owner – vlasnik proizvoda

- Definira funkcionalnosti proizvoda
- Odlučuje o datumu i funkcionalnostima verzija
- Odgovoran za povrat investicije
- Daje prioritete zahtjevima prema tržištu
- Mijenja prioritete svake iteracije po potrebi
- Prihvata ili odbija rezultate (sprinta)



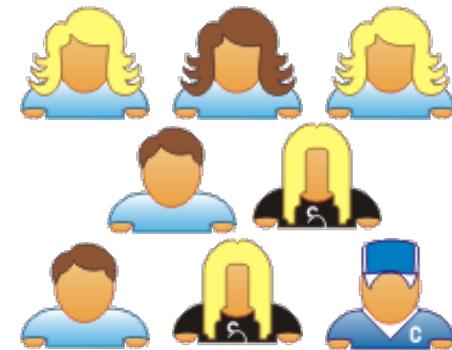
Scrum Master

- Predstavlja upravljanje projektom
- Odgovoran za provedbu metode Scrum
- Uklanja prepreke
- Osigurava produktivost i funkcionalnost tima
- Omogućuje suradnju svih u timu
- Štiti tim od vanjskih utjecaja!



Tim

- 5-9 osoba
- Uloge:
 - programeri, testeri, dizajneri...
- Full-timeri (uz rijetke iznimke)
- Timovi se samoorganiziraju
 - Nema “šefova”, barem ne formalnih
- Članovi tima se ne mijenjaju tijekom sprinta!



Scrum

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts



Sprint planning – planiranje sprinta

- Tim odabire koje “zaostale” zahtjeve može odraditi
- Definira se popis zahtjeva sprinta
 - Definiraju se zadaci (trajanja 1-16 sati)
 - Cijeli tim sudjeluje

As a vacation planner, I want to see photos of the hotels.

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)

The daily scrum – dnevni scrum

- Karakteristike:
 - Svaki dan
 - 15 minuta
- Ne služi za rješavanje problema
 - Svi mogu prisustvovati
 - Govore samo scrum master, članovi tima i product owner
- Važno: eliminira potrebu za nepotrebnim (dužim) sastancima

Svi odgovaraju na 3 pitanja:

1

Što si radio/la jučer?

2

Što ćeš raditi danas?

3

Ima li kakvih problema?

- Odgovori su dogovori s ostalim članovima tima
 - ne prema SCRUM masteru ili product owneru

The sprint review – pregled (završenog) sprinta

- Tim prezentira napravljeno
- Demonstracija novih funkcionalnosti
- Neformalno
 - Pravilo: priprema do 2 sata
 - Nema prezentacije
- Sudjeluje čitav tim

Sprint retrospective – retrospektiva sprinta

- Periodička provjera što radi a što ne
- 15-30 minuta
- Nakon svakog sprinta
- Sudjeluje čitav tim
 - SCRUM master
 - Product owner
 - Tim
 - Ostali – npr. naručitelj

Start / Stop / Continue

- Tim dogovara što bi željeli:

Započeti raditi

Prestati raditi

Nastaviti raditi

Scrum

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

Product backlog - zaostali zahtjevi



- Zahtjevi
- Popis svog posla na projektu
- Predstavljeno tako da svaka stavka ima vrijednost za nekoga
- Prioritete definira vlasnik proizvoda
- Ponovna dodjela prioriteta na početku svakog sprinta

Primjer zaostalih zahtjeva

Backlog item	Estimate
Allow a guest to make a reservation	3
As a guest, I want to cancel a reservation.	5
As a guest, I want to change the dates of a reservation.	3
As a hotel employee, I can run RevPAR reports (revenue-per-available-room)	8
Improve exception handling	8
...	30
...	50

Cilj sprinta

- Na što ćemo se usmjeriti tijekom sprinta?

Database Application

Make the application run on SQL Server in addition to Oracle.

Life Sciences

Support features necessary for population genetics studies.

Financial services

Support more technical indicators than company ABC with real-time, streaming data.

Upravljanje zahtjevima sprinta

- Članovi tima se sami javljaju za posao / rješavanje nekog zahtjeva
 - Posao se nikada ne delegira
- Preostali posao se ažurira na dnevnoj bazi
- Svaki član tima može mijenjati popis zaostalih zahtjeva
- Ako je nejasno što treba napraviti, definiraj nadzahtjev s više potrebnog vremena i rasporedi ga kasnije
 - Ažuriraj potrebno vrijeme za rješavanje kada zadatak postane jasniji

Primjer zahtjeva sprinta

Tasks	Mon	Tues	Wed	Thur	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

Literatura

- www.mountaingoatsoftware.com/scrum
- www.scrumalliance.org
- www.controlchaos.com
- scrumdevelopment@yahoogroups.com