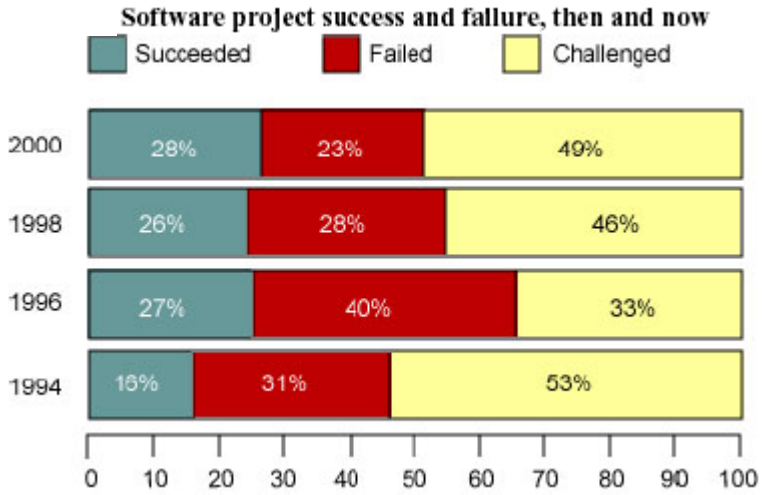
Zavod za telekomunikacije

Agilne metode razvoja programa

doc. dr. sc. Mario Kušek

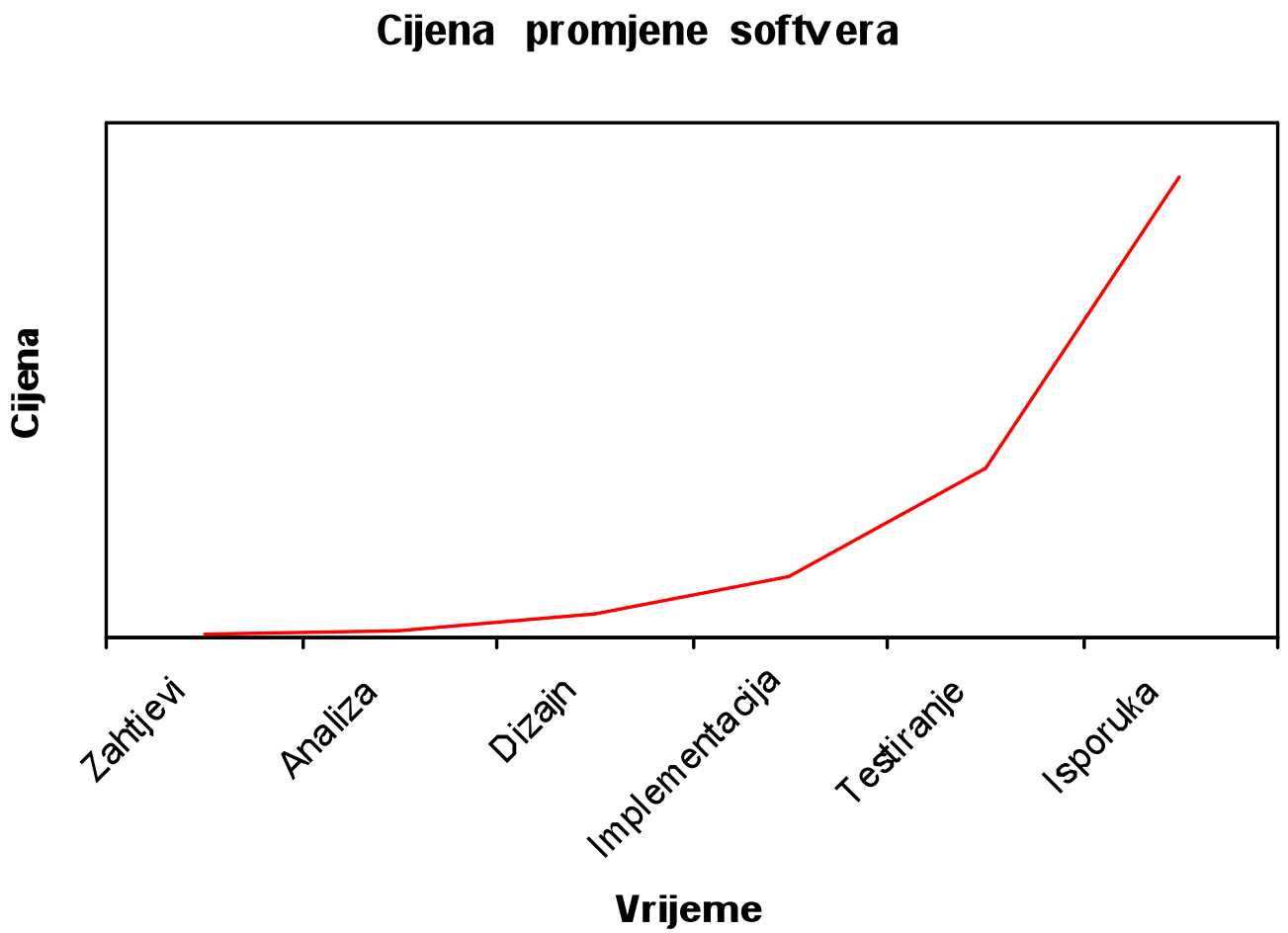
Rizik – osnovni problemi razvoja softvera

- Probijanje rokova
- Odustajanje od projekta
  - nakon probijanja rokova
- Sustav postaje “gorak”
  - nabacana arhitektura
- 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”
http://www.standishgroup.com/chaos_chronicles/introduction.pdf

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* - http://agilemanifesto.org
- *Agile Alliance*
  - Neprofitna organizacija koja traži i promovira znanje i rasprave o svim agilnim metodama
  - http://agilealliance.org

Principi (1)

- najviši prioritet je zadovoljiti kupca kroz rane i kontinuirane isporuke programskog proizvoda koji ima vrijednost kupcu
- promjena zahtjeva je uobičajena pa i u kasnim fazama projekta. Agilni procesi podržavaju promjene tako da kupac to može iskoristiti kao prednost nad konkurentima
- programski proizvod koji radi isporučuje se svakih nekoliko tjedana ili mjeseci (preporuča se kraći period)
- naručitelji koji će koristiti proizvod i razvijatelji moraju svakodnevno zajedno raditi kroz čitav projekt
- projekti se grade oko motiviranih osoba kojima se osigurava okolina, podržava ih se i vjeruje da će napraviti posao kako treba

Principi (2)

- najučinkovitiji način razmjene informacija u razvojnom timu je komunikacija licem u lice
- programski proizvod koji radi je osnovna mjera praćenja napredovanja
- agilni procesi promoviraju održivi razvoj (kao maraton)
- stalna pažnja usmjerena na tehničku izvrsnost i dobar dizajn povećavaju agilnost
- jednostavnost – "umjetnost" povećanja posla koji ne treba napraviti je ključno (tehnički gledano)
- najbolje arhitekture, zahtjevi i dizajn pojave se iz samoorganizirajućih timova
- tim periodički ispituje dobre i loše postupke te ih prokušava popraviti u sljedećem periodu

Agilne metodologije

- Ekstremno programiranje – XP (*Extreme Programming*)
  - http://www.xprogramming.com
  - http://www.extremeprogramming.org
- Crystal family of methodologies
  - http://members.aol.com/acockburn
- Open Source
  - The Cathedral and the Bazar – http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar
- Adaptive Software Development
  - http://www.adaptivesd.com
- Scrum
  - http://jeffsutherland.com/scrum/index.html
- Feature Driven Development
  - http://www.featuredrivendevelopment.com
- DSDM (Dynamic System Development Method)
  - http://www.dsdm.org

Zavod za telekomunikacije

Ekstremno programiranje (extreme programming – XP)

Što je XP?

- XP je metodologija za razvoj softvera gdje su timovi male ili srednje veličine, a 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* ili *test driven 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 se zajednički jezik (termini)

Praktične tehnike (3) – planiranje verzija

- obavlja se jednom za svaku verziju
- predstavlja postupak koji naručiteljeve priče pretvara 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:
    - Pisanje priče (zahtjev)
    - 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 (YAGNI – *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 vrlo kompliciranа
- alati i UML se koriste samo kada pomažu da se nešto napravi – NE TROŠITI VRIJEME NA UREĐIVANJE DIJAGRAMA

Praktične tehnike (6) – testiranje

- 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 koda
    - 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 radi li sustav 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 implementiranja neke funkcije
    - pokušava se pojednostavniti 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 koda 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 koda
- 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
  - kod 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 kod
- "*You break it, you fix it.*"
- zahtjeva ekstremnu disciplinu – pogotovo kada nemate vremena
- testovi pomažu programeru da promijeni 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 promijeni dio koda za koji je zadužen
- poboljšava se kvaliteta koda
  - zna se da će kad tad netko pogledati taj kod i 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, Subversion)
- automatsko dnevno/po potrebi prevođenje i povezivanje koda iz repozitorija te slanje/objavljivanje izvješća

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

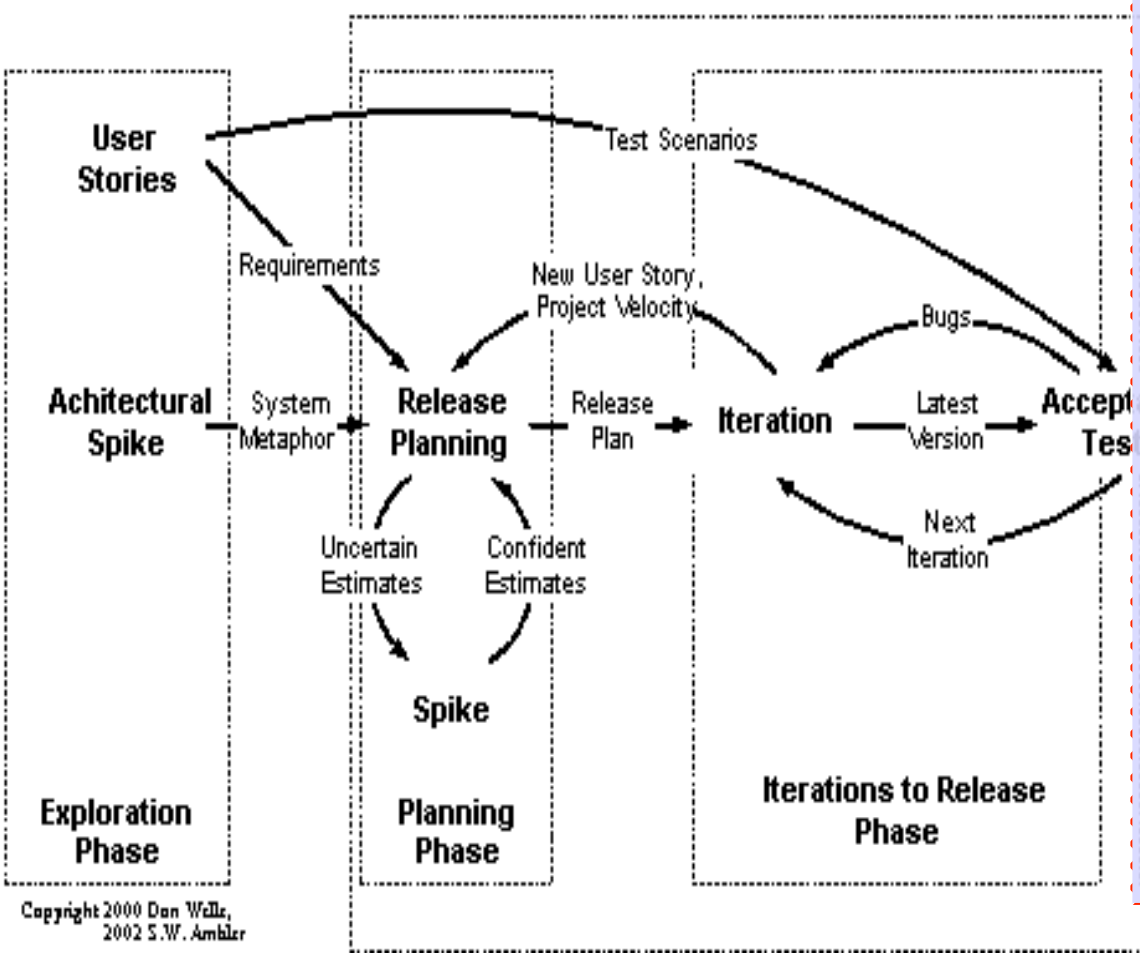
Praktične tehnike (12) – naručitelj u timu

- 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 (alati)
- 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

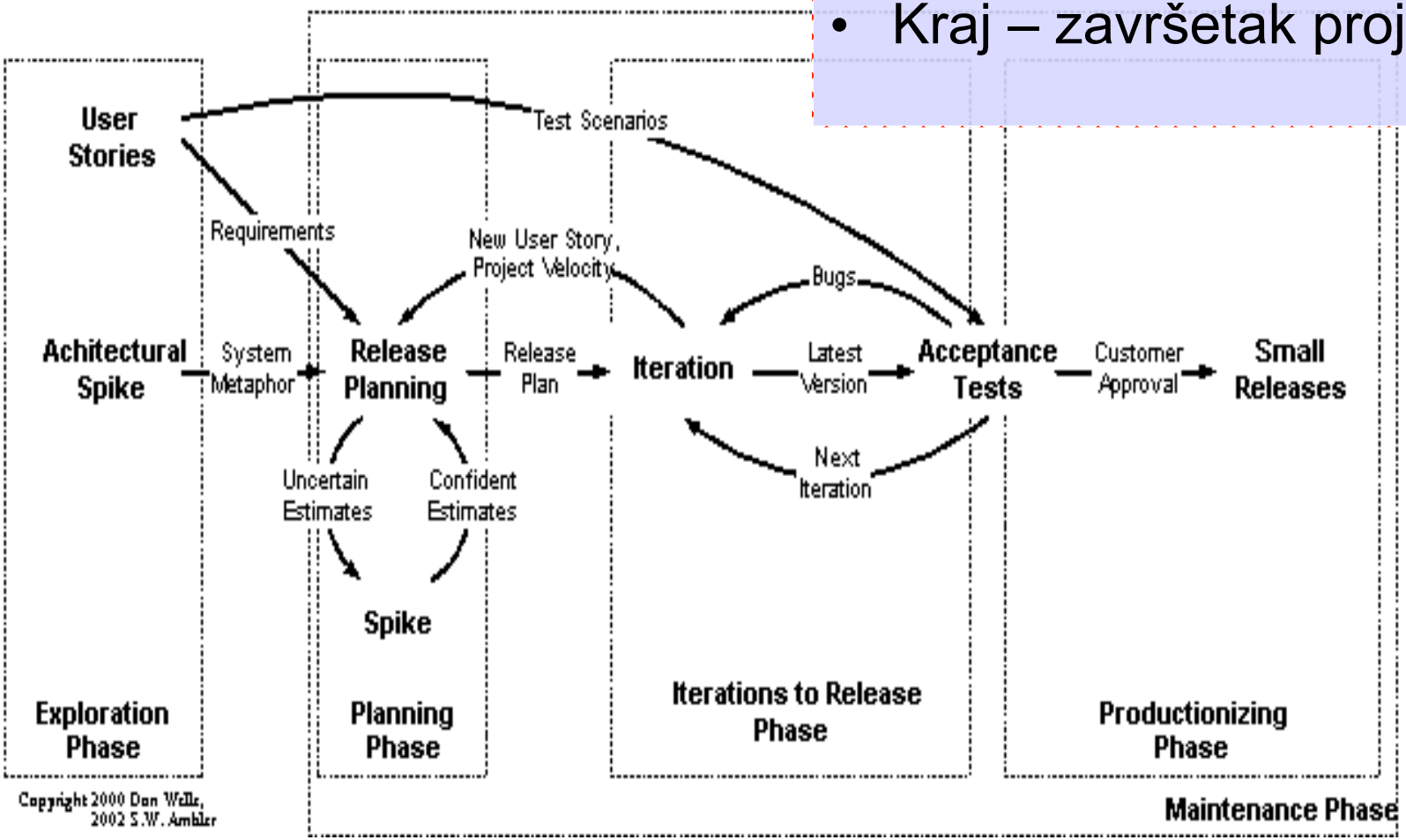
Faze projekta (1)



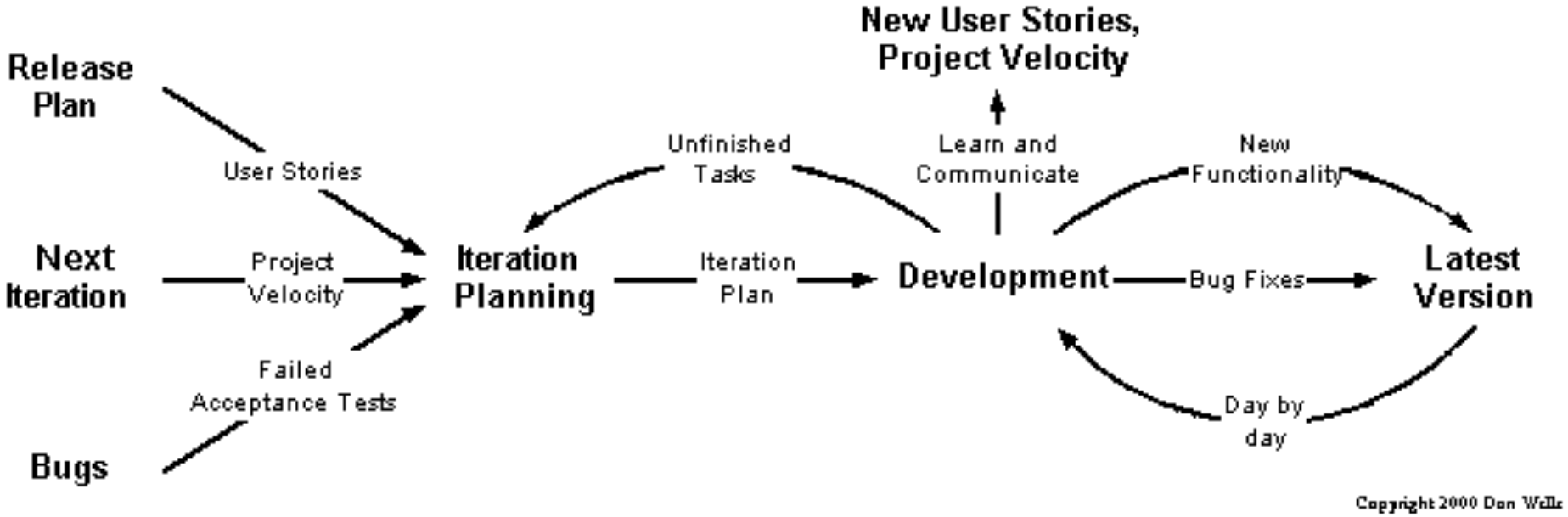
- Istraživanje – ideje, što, zašto, kako?
- Planiranje – izdavanja verzija (*release*)
- Iteracije
  - pisanje novih zadataka (priča)
  - procjena zadataka
  - sortiranje zadataka (prioritet, tehnički rizik)
  - planiranje iteracije (i verzija)
  - pisanje koda

Faze projekta (2)

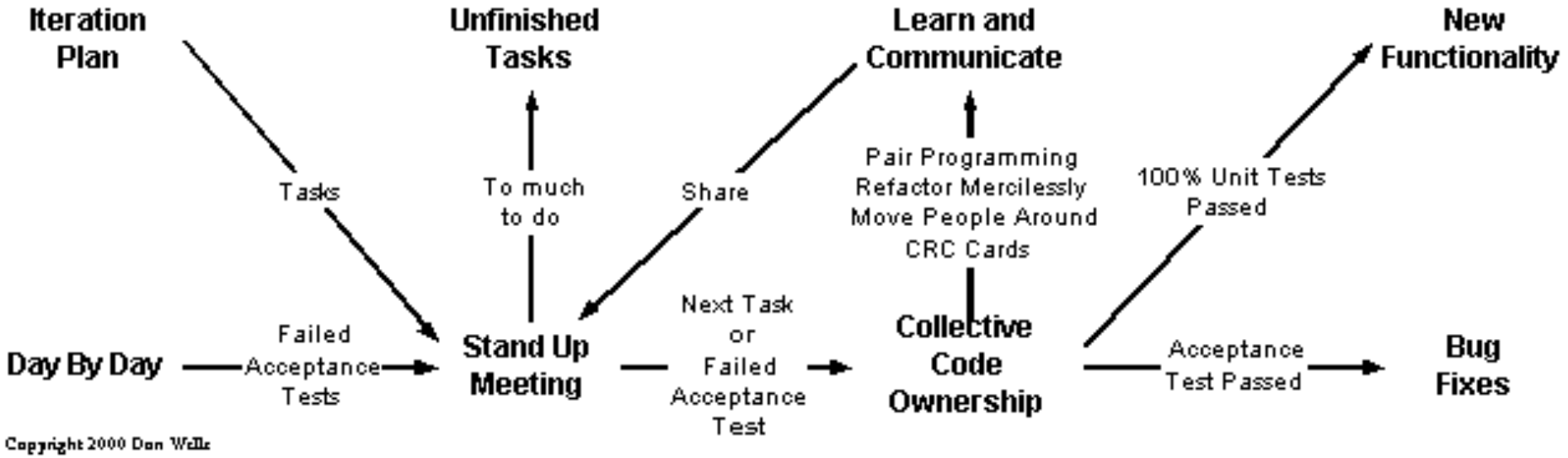
- Izdavanje verzije (*release*)
  - performanse, opterećenje sustava
- Održavanje
  - dodavanje nove funkcionalnosti, ispravci grešaka
- Kraj – završetak projekta



Životni ciklus iteracije

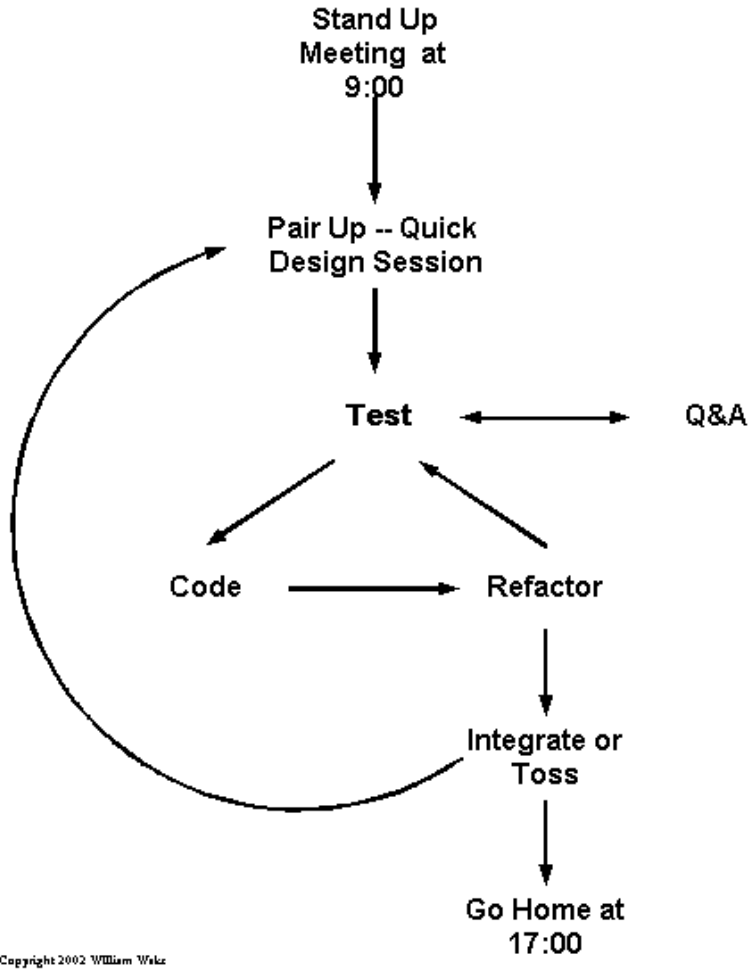


Životni ciklus razvoja u jednom danu



Jedan dan razvijatelja

- 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 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



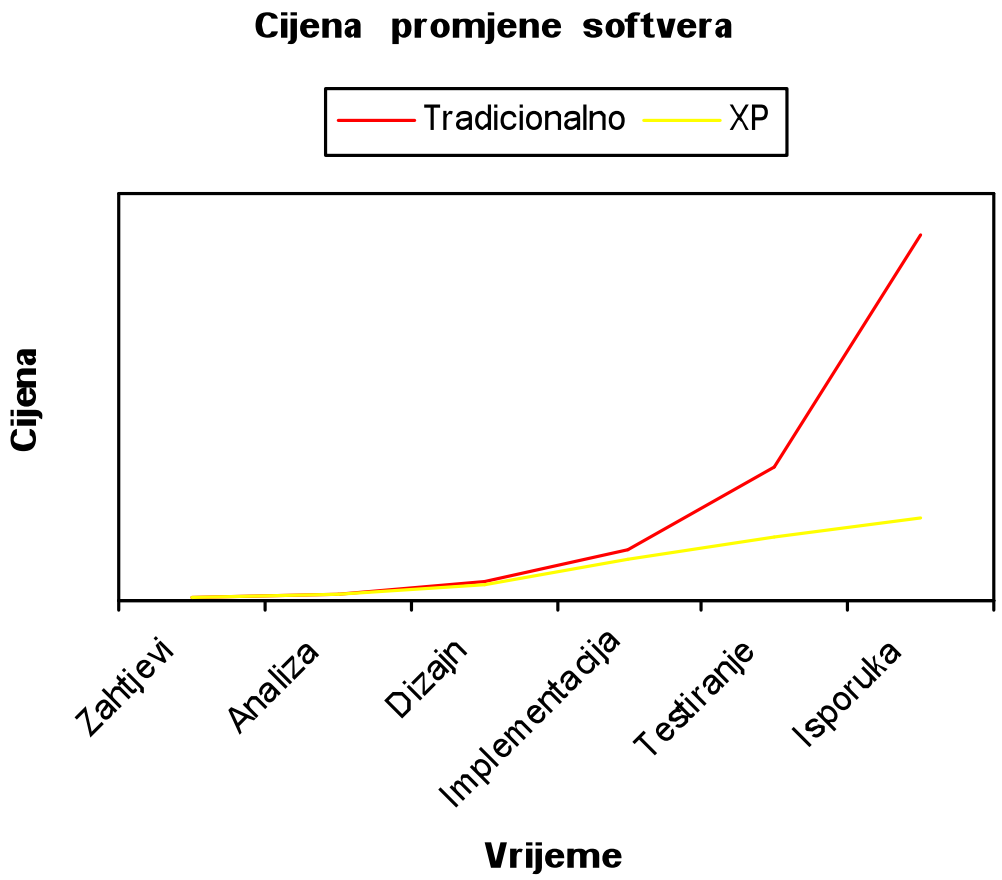
Kako XP riješava rizik? (1)

- **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 riješava 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, konkretne 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?

- kada postoji velik otpor XP načinu programiranja
- nefleksibilna okolina
- kada se treba stvarati velika količina dokumentacije
- u okolini gdje je prekovremeni rad uobičajen
- kada 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

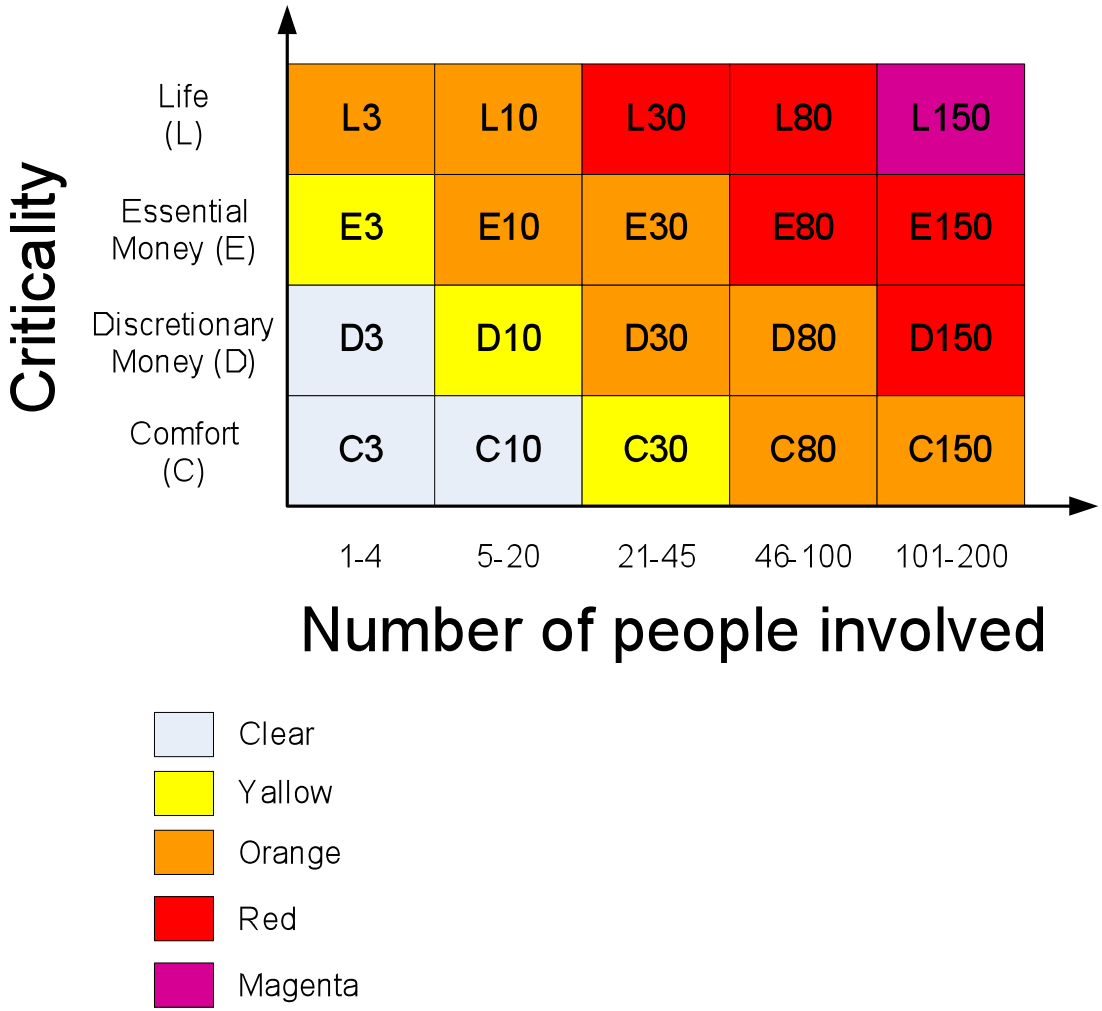
Kristalna obitelj metodologija (*Crystal Method*)

- imena prema geološkom kristalima: čisti (clear), žuti, narančasti, crveni, ...
- filozofija:
  - orijentirani ljudima (people-centric) i komunikaciji
  - ultralagana – smanjuje papirologiju, "*overhead*" i birokraciju
  - prilagođava se okolini, problemima, timu, ...
- svaka metodologija je za određenu svrhu
- najpoznatije: crystal clear, crystal orange, crystal orange web

Kristalna obitelj metodologija (2)

- osnovni principi:
  - inkrementalni razvoj
  - pogled unatrag i učenje iz grešaka
  - tolerancija
  - svaki projekt ima svoja pravila

- razlika u odnosu na XP:
  - manje discipline
  - više tolerancije
  - jednostavnije



Otvoreni kod (*Open Source*)

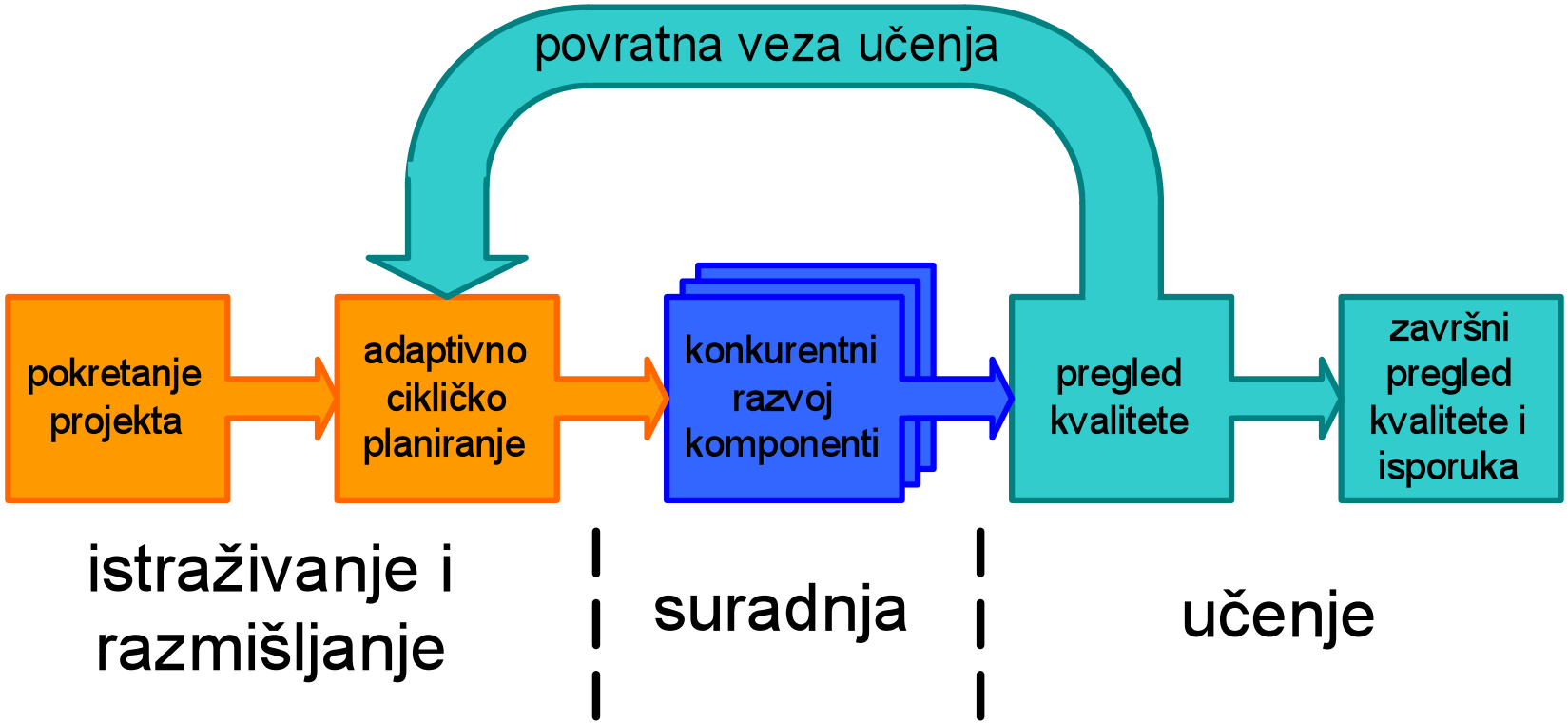
- Velik broj razvijatelja, testera, pisača dokumentacije i korisnika
- Svatko može vidjeti kako što radi
- Svaki angažirani sudionik želi to raditi
- Zaslužni dobiju veća prava (npr. izmjena koda ili web stranice)
- Velika fluktuacija ljudi
- Raspodijeljeni ljudi
- Samostalni rad
- Glavna komunikacija su *mailing* liste gdje odgovori vrlo brzo stižu
- Strogo definirana pravila kodiranja
- Svaki projekt je drugačije organiziran

Adaptive Software Development (1)

- istraživanje i razmišljanje
  - zamjenjuje planiranje
  - idejni plan
- suradnja
  - ujedinjuje upravljanje i izvođenje projekta
  - spontana, adaptivna, živa
  - balans između mrtvila i anarhije
- učenje
  - ispitivanje pretpostavki
  - analiza procesa, proizvoda, zahtjeva

Adaptive Software Development (2)

Faze životnog ciklusa ASD-a



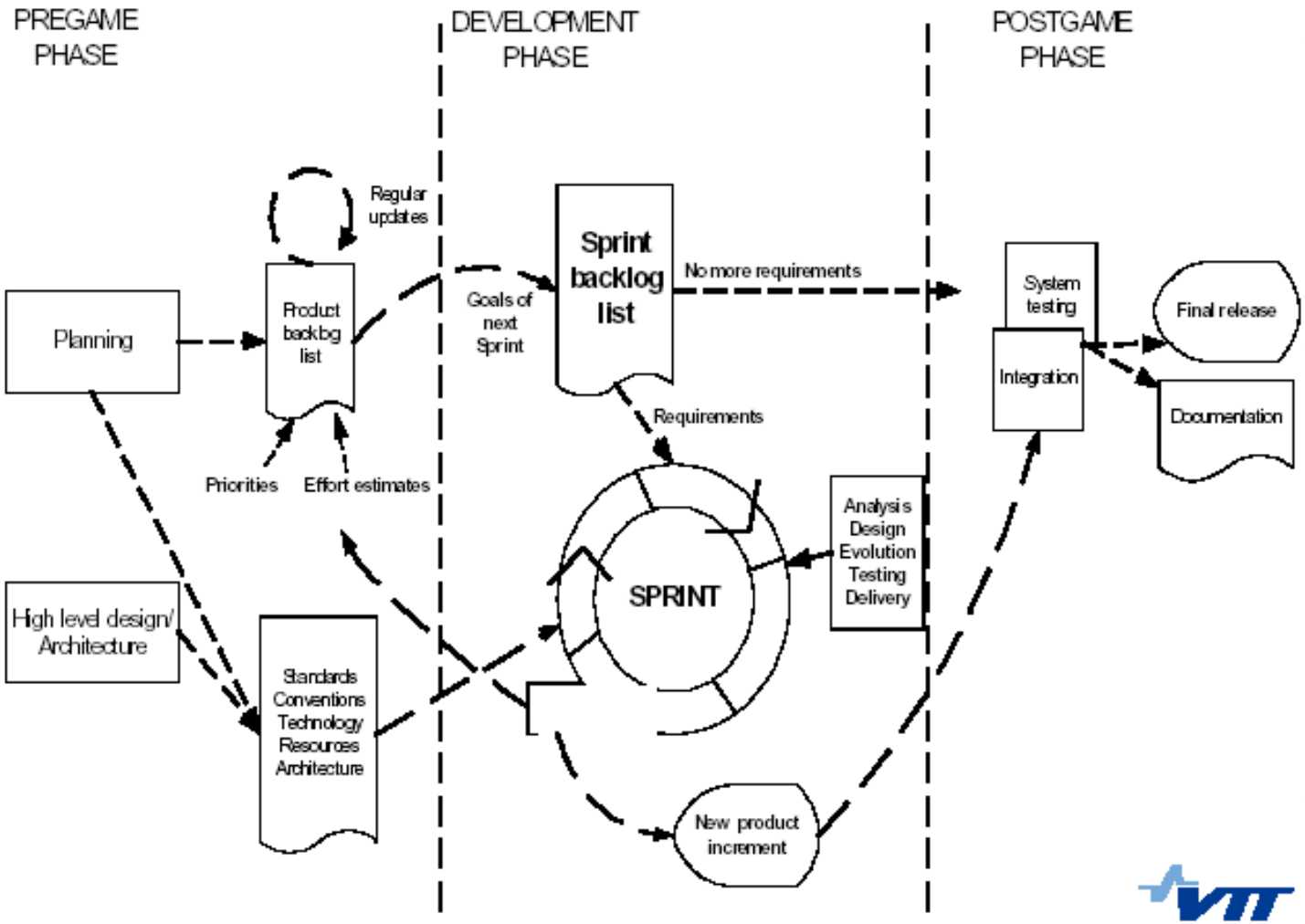
Adaptive Software Development (3)

- svojstva:
  - orijentiran zadatku
    - svaka iteracija mora biti u skladu sa zadatkom
  - razvoj komponenti
    - u svakom ciklusu se razvije barem jedna komponenta koja radi
  - iterativno
  - vremenski ograničeno (*time-boxed*)
    - svaka iteracija ima ograničeno vrijeme
  - tolerantan promjenama
    - promjene su normalne i njima se treba prilagođavati
  - vođen rizikom
    - rizičniji dijelovi se rade na početku projekta

Scrum (1)

- Termin iz ragbija
- Nakon strategijskog plana igra se jedan sprint koji igrače približi cilju
- Sprint traje 4 tjedna (iteracija)
- Jedna igra (projekt) traje 3 do 8 sprintova
- Mali timovi
- Prilagođavanje naručitelju
- Česta izdanja
- Podjela na timove i zadatke
- Proizvod može biti gotov u svakom trenutku

Scrum (2)

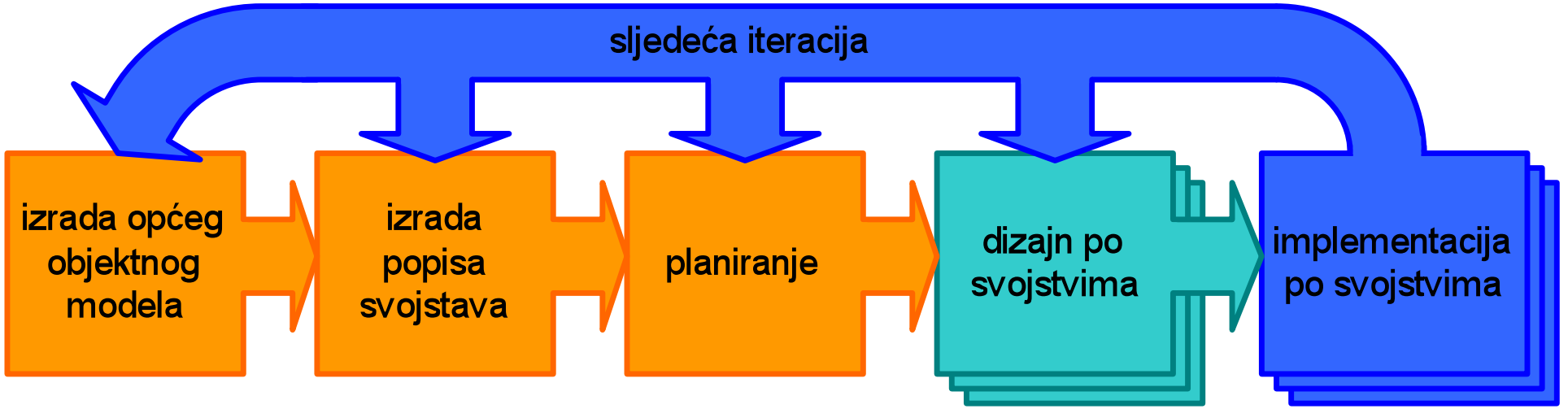


- "Pregame"
  - Planiranje
  - Arhitek.
  - Dizajn općeg modela
- Faza izrade
  - Sprint (30 dana)
    - Izrada (analiza, dizajn, kodiranje)
    - Preklapanje
    - Pregledavanje
    - Prilagodba
- "Postgame"
  - Zatvaranje proj.

Feature Driven Development (1)

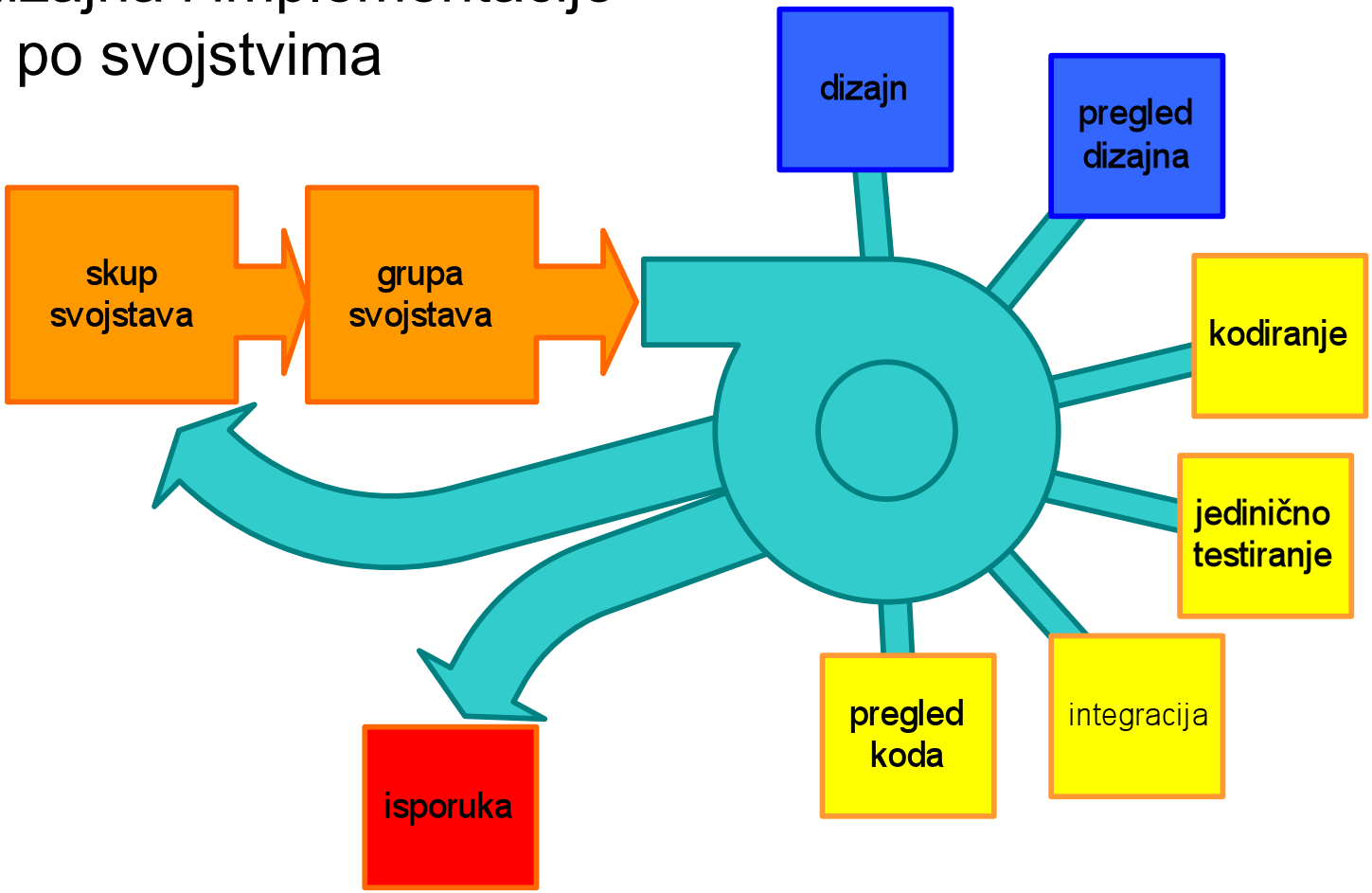
- Dizajn s UML dijagramima
- Dekompozicija svojstava
- Fokus na dizajn i implementaciju
- Ne pokriva sve aspekte procesa izrade programskog proizvoda
- Nije za male timove (<10 ljudi)
- Dobar za velik broj razvijatelja
- Kratke iteracije od 2 dana do 2 tjedna

Feature Driven Development (2)



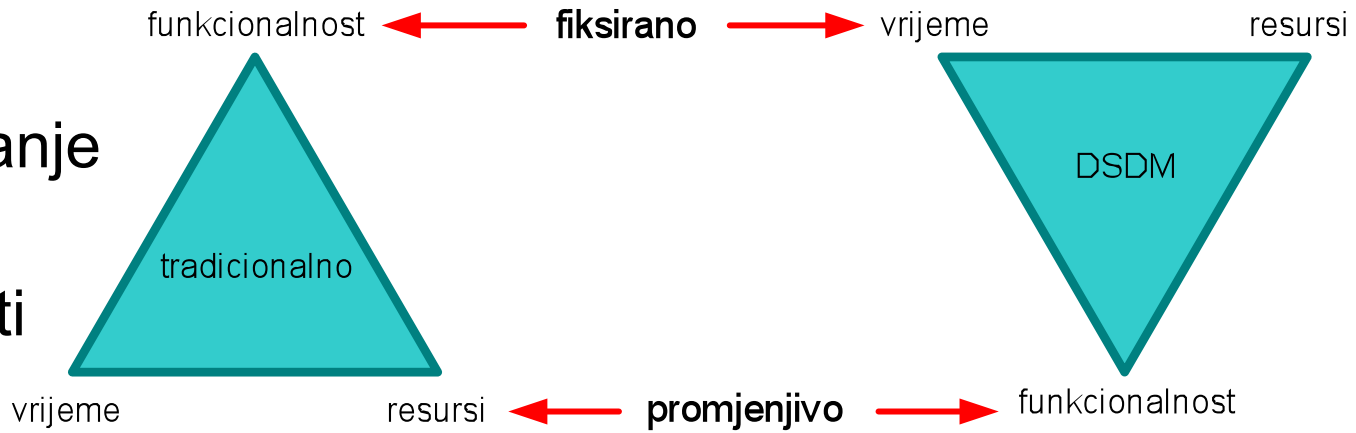
Feature Driven Development (3)

Proces dizajna i implementacije po svojstvima



DSDM (Dynamic System Development Method)

- Principi:
  - Aktivno sudjelovanje korisnika
  - Tim mora donositi odluke



  - Česta isporuka proizvoda
  - Tim mora isporučiti proizvod koji služi korisniku
  - Iterativni i inkrementalni razvoj
  - Sve promjene za vrijeme razvoja se mogu poništiti
  - Osnovni zahtjevi su definirani na početku
  - Testiranje i integracija se provodi tijekom čitavog projekta
  - Suradnja i kooperacija između svih sudionika je ključna

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