4. UML Dijagram aktivnosti

Dijagrami aktivnosti: Razine apstrakcije

1. Konceptualna
2. Logicka (specifikacijska)
3. Fizicka (implementacijska)

Dijagram aktivnosti se koristi za:

- modeliranje poslovnih procesa
- modeliranje poslovne logike unutar jednog slucaja ili scenarija korištenja (poslovno pravilo)

Dijagram aktivnosti se ne koristi za:

- opis unutarnje logike složene operacije (postoje bolji nacini)
- drugaciji prikaz dijagrama stanja

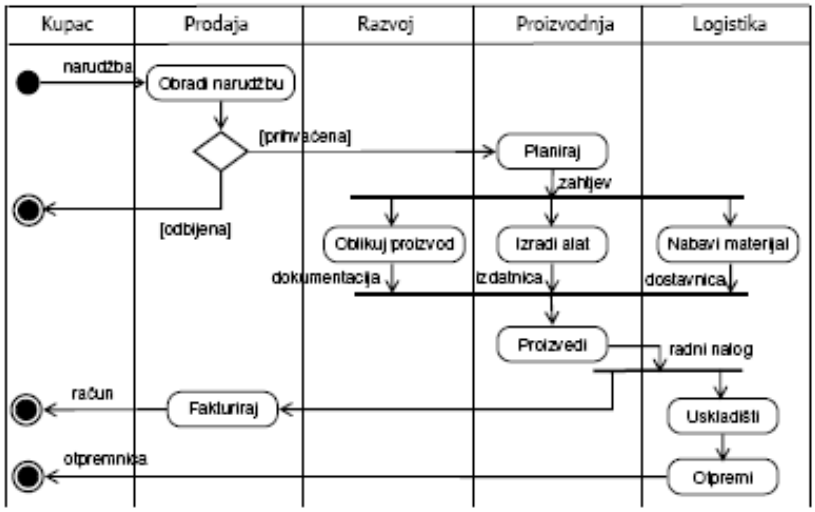
Koristan za analiziranje slucajeva korištenja opisujuci njegovu unutarnju logiku:

1. Pocetak slucaja korištenja
2. Opis što slucaj korištenja radi: slijed aktivnosti, odnosno tijek posla (workflow) od pocetne do završne tocke, paralelno odvijanje aktivnosti, tocke u kojima se donose odluke i u kojima se tijek posla grana
3. Završetak slucaja korištenja

Prikazuje mehanizme za odredivanje redoslijedakontrolnih i objektnih tokova medu akcijama:

- Objektni tokovi za odredivanje redoslijeda kojim cvorovi produciraju podatke za druge cvorove
- Kontrolni tokovi kojima se odreduje redoslijed izvršavanja cvorova
- Kontrolni cvorovi u kojima se ovisno o situaciji odlucuje o objektnim i kontrolnim tokovima
- Generalizacija kojom se jednom aktivnošcu zamjenjuju cvorovi i bridovi koje ona sadrži

- Objektni cvorovi koji reprezentiraju objekte i podatke što teku u i iz pozvanih ponašanja, ili reprezentiraju skupove tokova što cekaju da se prenesu dalje



Aktivnost i akcija

- Aktivnost prikazuje ponašanje
- Aktivnosti se mogu rašclanjivati i prikazivati dijagramima nižih razina.
- Model aktivnosti prikazuje akcije, njihov redoslijed i prijenos kontrole medu akcijama

Aktivnost je specifikacija parametriziranog ponašanja, u obliku koordiniranog slijeda podredenih jedinica ponašanja.

Elementarne jedinice ponašanja su akcije.

- Akcije se ne rašclanjuju na dijagramima aktivnosti, iako njihovo izvršavanje može zahtijevati pozivanje i izvršavanje drugih akcija ili aktivnosti.

UML Superspecifikacija

Aktivnosti mogu sadržavati akcije razlicitih vrsta:

primitivne funkcije, npr. aritmeticke funkcije

poziv nekog ponašanja, npr. aktivnosti

akcije komuniciranja, npr. slanje signala

rukovanje s objektima, npr. citanje ili promjena vrijednosti nekog atributa

Brid aktivnosti može biti:

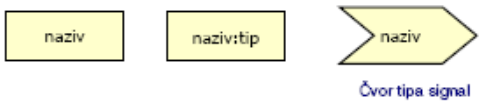
Objektni tok (Object flow) – sadrži podatke

Upravljacki tok (Control flow) – upravlja izvršenjem aktivnosti

Notacija: vektor, može imati i naziv, uz strelicu

Spojište (Connector) ima iskljucivo notacijski karakter i služi za nastavljanje toka u na drugom mjestu. Spojište mora imati jedinstvenu oznaku i samo jedan ulazni ili izlazni tok.

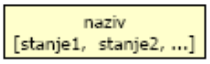
- *Objektni čvor (Object Node)* je čvor što prikazuje instancu nekog klasifikatora, često u određenom stanju, koja je raspoloživa na određenom mjestu u aktivnosti.
- Naziv ukazuje na tip objektnog čvora.



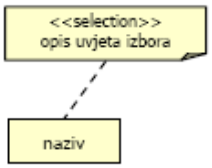
Čvor tipa signal

- Objektni čvor je dio definicije objektnog toka!

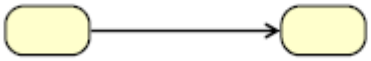
Dopuštena stanja se navode u uglatoj zagradi



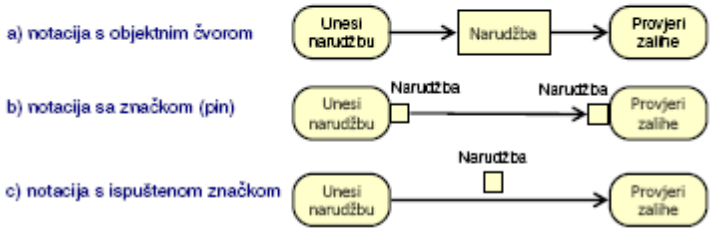
Ako stanje nije zadano, može biti bilo koje dopušteno



- *Kontrolni tok (Control Flow)*



- *Objektni tok (Object Flow)* prikazuje podatke ili objekte koji teku između dva čvora



Token (žeton)

Token je sadržaj tokova medu cvorovima.

Sadržaj tokena je objekt, datum, kontrola i sl.

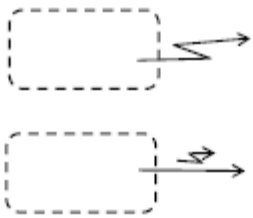
Svaki se token razlikuje od bilo kojeg drugog, cak i u slucaju da mu je sadržaj isti

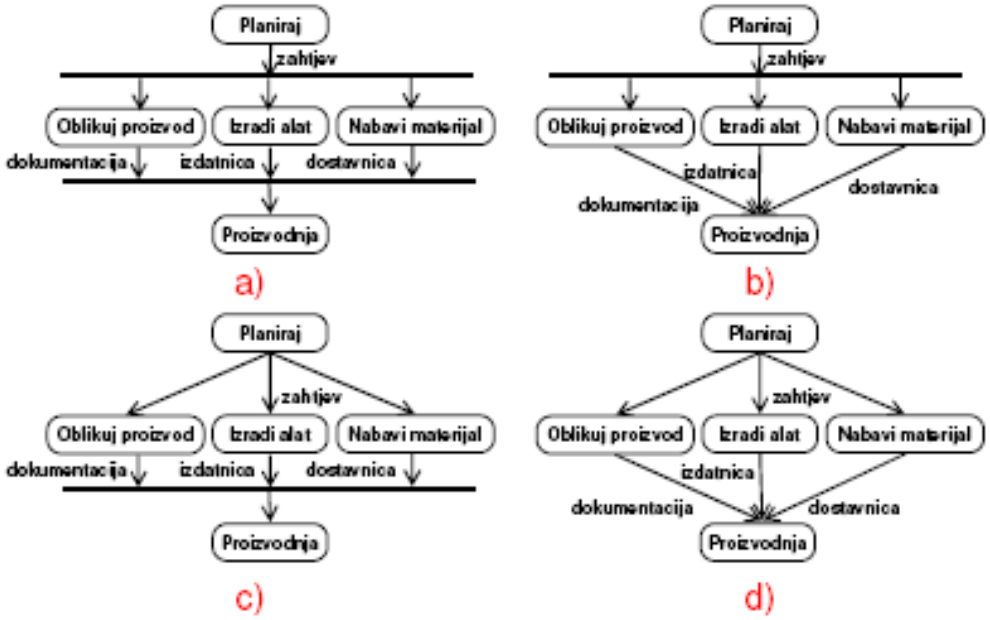
1. Cvor pocinje s izvršenjem kad su zadovoljeni uvjeti ulaznih tokena
2. Kad pocne izvršenje cvora, tokeni se prihvacaju s jednog ili više ulaznih bridova
3. Tokeni ne mogu zastati na kontrolnim cvorovima
4. Po okoncanju izvršenja, token se stavlja na raspolaganje sljedecim cvorovima.

Ponavljajuci dogadaj - Ova akcija prihvaca vremenski okidac ili se ponavlja tijekom vremena, u skladu sa zadanim pravilom

Prekidljivo područje aktivnosti (Interruptable Activity Region)

- Grupa akcija za koju vrijedi da je tijekom njihovog izvođenja moguća terminacija tokena





5. UML Dijagram strojeva stanja (Statemachine Diagram)

Dinamika sustava - Dinamika sustava ocituje se kroz promjene stanja objekata, koje nastaju kao posljedica uzajamnog djelovanja objekata.

1. Kao rezultat aktivnosti jednog objekta (pokretanje njegovih metoda posluživanja), može nastati poruka drugom objektu.
2. Za objekt koji prima poruku, ona je dogadaj koji može izazvati njegove aktivnosti i promjenu stanja.
3. Preduvjet za to je da objekt koji prima poruku ima odgovarajuci mehanizam, metodu posluživanja.

Stroj stanja (Statemachine) - Stroj stanja je model dinamike (ponašanja) jednog objekta sustava

Stroj stanja prikazuje:

- diskretna stanja koja bi objekt mogao imati tijekom svog životnog ciklusa
- pocetnu i završnu tocku u prikazanom nizu promjena stanja
- moguce prijelaze iz stanja u stanje
- uvjete i efekte prijelaza

Stroj stanja: Razine apstrakcije

Konceptualna

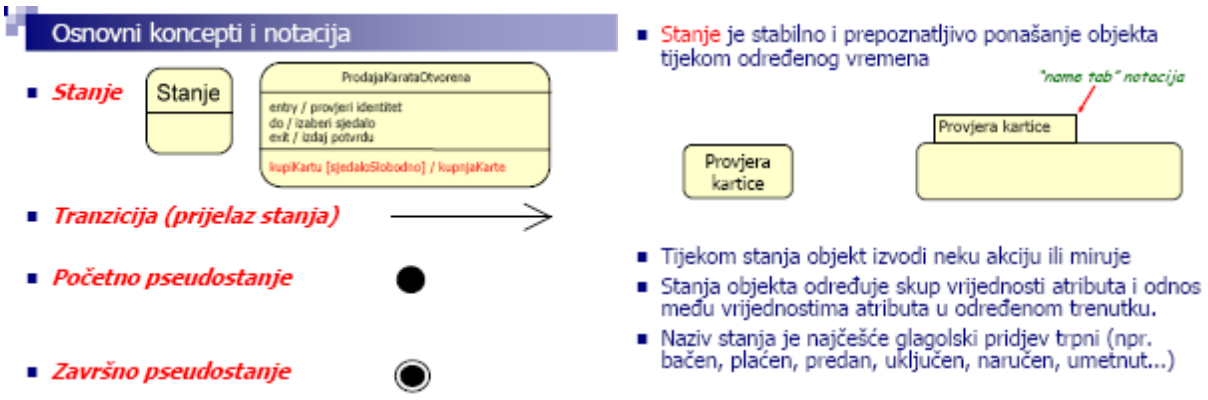
Apstraktna stanja (npr. "modem ukljucen", “provjeren identitet vozaca" i sl.)

Logicka (specifikacijska) - detaljna specifikacija stanja vezana uz vrijednosti atributa (npr. numericka vrijednost, iskaz kvalitete i sl.)

Fizicka (implementacijska) - realizacija stanja (npr. "signal 5 V na port bth124)

UML Dijagram strojeva stanja (Statemachine Diagram)

Pomaže analiticarima, dizajnerima i programerima da bi razumjeli ponašanje objekata u sustavu

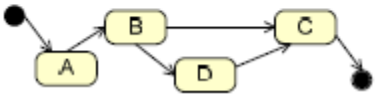


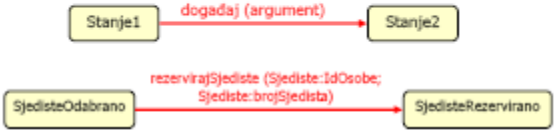
- Vektor predstavlja promjenu iz jednog stanja u drugo
- Predstavlja događaj i aktivnost koja se tijekom promjene stanja izvodi
- Događaj pokreće prijelaz stanja

događaj(argument)

npr.

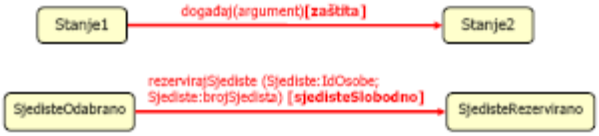
događaj(argument, npr. klasa:atribut)

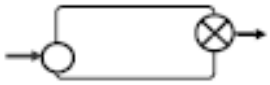




- *Zaštita (Guard) je preduvjet (precondition) koji mora biti ispunjen da bi se neki događaj desio, odnosno da bi došlo do prijelaza stanja.*

događaj(argument)[zaštita]



- UML 2.0 poznaje:
  - početno ● i završno pseudostanje ◉
  - ulazna i izlazna točka 
  - povijesna pseudostanja; bliža (H) i dalja (H*) povijest.
- Pseudostanja nisu prava stanja. Ona nemaju trajanje, u istom trenutku nastupaju i prestaju.
- Mora postojati barem jedan prijelaz stanja između pseudostanja i pravog stanja.

Pocetno i završno pseudostanje

Dijagrami zapocinje s pocetnim pseudostanjem objekta.

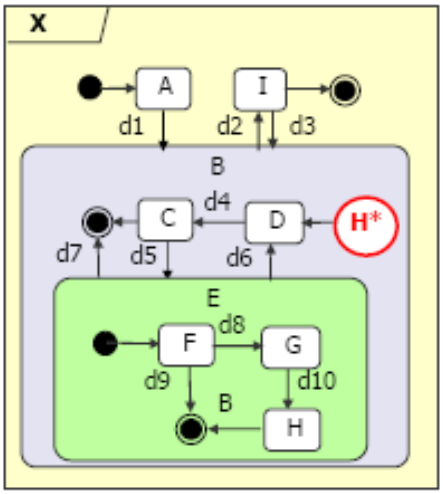
Da bi se smanjila kompleksnost, dijagrami strojeva stanja ustrojavaju se hijerarhijski

Pamcenja povijesti složenog stanja služi tome da se aktiviranjem složenog stanja zapravo aktivira ono njegovo podstanje koje je bilo aktivno kad je složeno stanje posljednji puta napušteno.

Kad povijest ne postoji, nego prvi puta pristupamo složenom stanju, aktivira se prvo podstanje nakon povijesnog pseudostanja.

H je plitko povijesno pseudostanje jer ne pamti zadnja podstanja unutar složenih podstanja.

Za razliku od njega H* pamti koje podstanje unutar složenog podstanja je bilo zadnje aktivno i aktivira ga kod sljedeceg pristupa



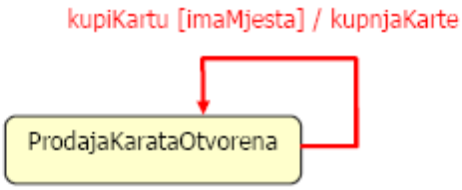
- Stroj stanja ima tri stanja: A, B i I. Stanje B je složeno i sadrži stanja C, D i E. Stanje E je složeno i sadrži stanja F, G i H.
- B sadrži povijesno pseudostanje.
- Kada se stanje B inicira prvi puta, stanje B nema povijesti. Stoga se inicira stanje povezano s povijesnim pseudostanjem, a to je D (default stanje povijesnog pseudostanja).
- .Kod napuštanja stanja B, H pamti zadnje stanje koje je bilo aktivno. U primjeru na slici, to je ili C ili E.
- Plitko povijesno pseudostanje ne pamti zadnja aktivna stanja u podstanjima. Kod ponovnog aktiviranja stanja B, aktivirat će se C ili E, ovisno od toga koje je bilo aktivno kod zadnjeg napuštanja stanja B.
- Duboko povijesno pseudostanje pamti i zadnja aktivna stanja u podstanjima (E), a ako je ono složeno i njegovo zadnje aktivno podstanje (F ili H). Kod ponovnog aktiviranja stanja B, aktivirat će se C ili E, ovisno od toga koje je bilo aktivno kod zadnjeg napuštanja stanja B. Ako se aktivira E, aktivirat će se ili F ili H, ovisno od toga koje je bilo aktivno prije zadnjeg napuštanja E, odnosno B.

U slucaju da unutar složenog stanja ima više podrucja govori se o ortogonalnoj podjeli i konkurentnim podrucjima.Ortogonalna podjela oznacava da je složeno stanje podijeljeno na više podrucja podstanja. Svako konkurentno podrucje sadrži jedan podmodel stroja stanja. Svi se ovi podmodeli izvršavaju medusobno konkurentno.

Racvanje sabiranje prijelaza stanja

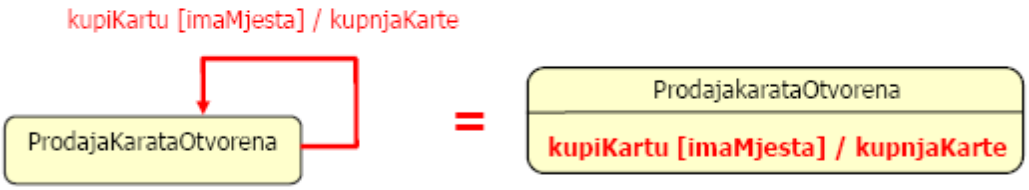
Racva prijelaza stanja (Forked transition) omogucava aktiviranje konkurentnih podrucja unutar složenog.

- *Samoprijelaz (Self transition)* je prijelaz stanja kojem je izvorište i odredište isto stanje.
- Samoprijelaz ne mijenja stanje, a prikazuje da je nastao neki događaj koji je zbog određenih razloga važan, ali ne mijenja stanje objekta.



- *Problem:* Objekt mora znati što će s primljenom porukom učiniti. Poruka može aktivirati neku metodu, a u svakom ju slučaju objekt mora terminirati kako u sustavu ne bi došlo do greške. Zato je kod ponašajnih strojeva stanja uveden *unutarnji prijelaz stanja,* istovjetan samoprijelazu.

- *Unutarnji prijelaz stanja* je događaj koji se desi u određenom stanju objekta i ne mijenja stanje.
- Naziv ovog događaja upisuje se u posebnom odjeljku simbola stanja.
- Unutarnji prijelaz ne izaziva grešku, niti promjenu stanja, može imati *zaštitu (guard)*, a prikazuje reakciju objekta na događaj koji može nastati, a da ne mijenja stanje.



- *Problem:* Važno je pravilo da isti događaj u nekom stanju ne može biti povezan i s vanjskim i unutarnjim prijelazom stanja!!

U okviru UML-a, na razini superspecifikacije, razlikuju se **dvije vrste strojeva stanja:**

1. Ponašajni stroj stanja - opisuje promjene stanja tijekom životnog ciklusa nekog objekta. Prikazuje i aktivnosti koje objekt izvodi u odredenom stanju ili tijekom promjene stanja.

Upravo aktivnosti predstavljaju ponašanje objekta.

2. Protokolni stroj stanja - specificira protokol medudjelovanja objekata, odnosno poruke na koje objekt može reagirati i nacin na koji to cini, s obzirom na stanje u kojem se nalazi u trenutku primanja poruke. Protokolni smjer stanja namijenjen je prikazivanju medudjelovanja objekata.

Postoje cetiri vrste aktivnosti:

1. Prijelazna aktivnost (transition activity)
2. Ulazna aktivnost (entry activity)
3. Izlazna aktivnost (exit activity)
4. Do aktivnost (do activity).

6. UML Dijagram slijeda (Sequence Diagram)

Dijagram strojeva stanja	Dijagram slijeda
Specificira ponašanja za sve moguće scenarije	*Ilustrira end-to-end ponašanje* za jedan scenarij
Prikazuje stanja i promjene stanja koje su posljedica događaja	Prikazuje izvorišta i odredišta ulaza i izlaza sustava
Orijentiran projektantu, korisniku može pomoći odrediti šifarnike	Orijentiran korisniku, projektantu može pomoći u provjeri strojeva stanja

Dijagrami slijeda (Sequence diagrams) prikazuju slijed poruka koje razmjenjuju objekti, koristeci životne crte

Model dinamike:

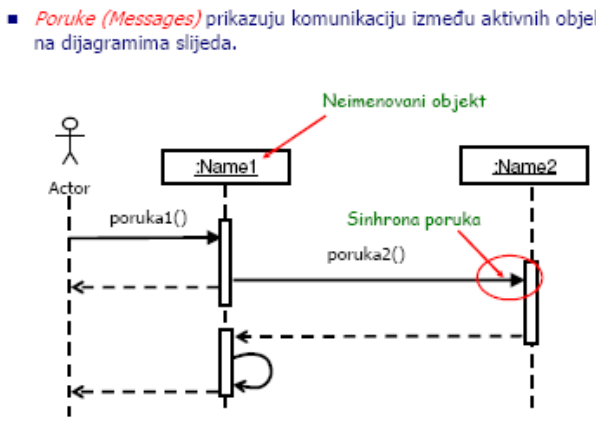
- pokazuje što ce se u sustavu dogoditi kad se desi neki dogadaj, odnosno objekt primi neku poruku
- ako nije naglašena vremenska komponenta, nema ga smisla crtati
- Detaljniji je od dijagrama aktivnosti

Razine apstrakcije i upotreba

Konceptualna - Dijagram suradnje se odnosi na jedan slucaj korištenja, jedan njegov dio ili više

Logicka (specifikacijska) - Razrada kompleksnih operacija, procedura i funkcija

Fizicka (implementacijska) - Razrada metoda i njihove unutarnje logike u programskom jeziku (npr. web servisi)



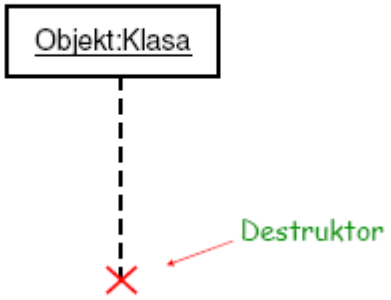
Poruke i operacije

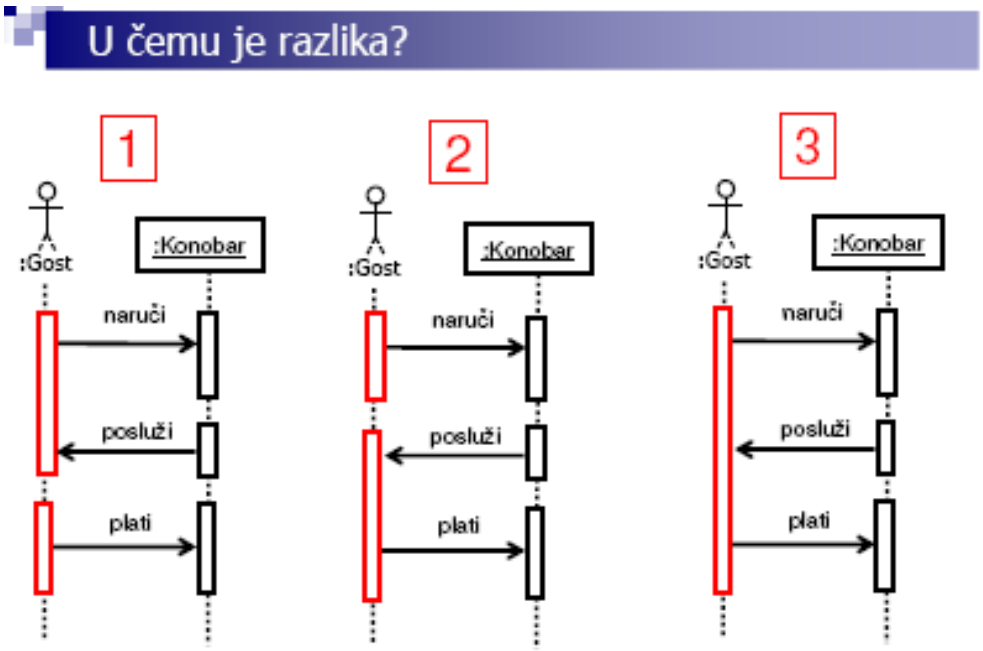
Operacije reprezentiraju ponašanje klase, a mogu se odrediti istražujuci razlicite scenarije ponašanja objekata, prikazane dijagramima slijeda

Dijagrami slijeda pomažu odredivanje operacija kod klasa i metoda kod objekata

Obratno, **operacije i metode se koriste u scenarijima dijagrama slijeda**.

Destruktor na dnu životne crte prikazuje da se objekt uklanja (iz memorije) po napuštanju životne crte





7. RUP - Rational Unified Process

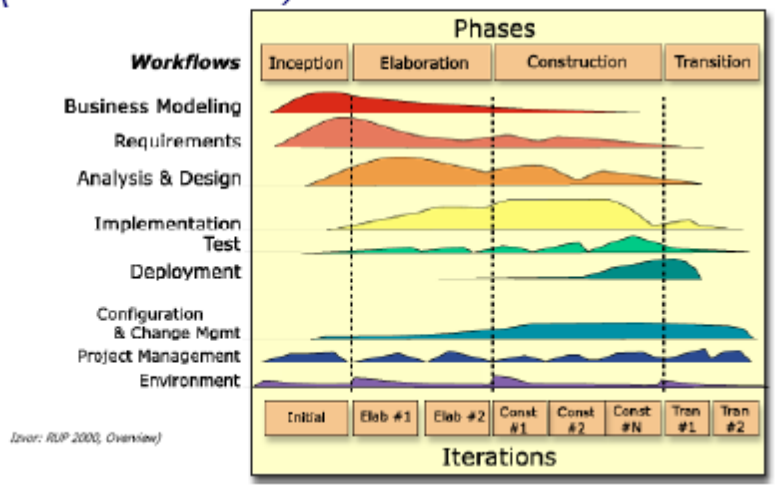
Rational Unified Process (RUP) je objektno orijentirana metodika projektiranja informacijskih sustava

Pristup

1. Iterativni razvoj
2. Upravljanje zahtjevima
3. Arhitektura zasnovana na komponentama
4. Vizualno modeliranje (UML)
5. Provjera kvalitete
6. Kontrola promjena

Faze razvoja – vremenska dimenzija projekta

- Inicijacija *(Inception Phase)*,
- Elaboracija *(Elaboration Phase)*,
- Konstrukcija *(Construction Phase)*,
- Tranzicija *(Transition Phase)*.



Izvor: RUP 2000, Overview)

Faza inicijacije – Dokumenti i modeli

1. Rjecnik pojmova (Glossary),
2. Zahtjevi narucitelja projekta (Stakeholder Requests),
3. Model slucajeva korištenja poslovnog sustava (Business Use-Case Model),
4. Objektni model poslovnog sustava (Business Object Model),
5. Dodatne specifikacije poslovnog sustava (Supplementary Business Specification),
6. Poslovna pravila (Business Rules),
7. Vizija (Vision),
8. Model slucaja korištenja (Use-Case Model),
9. Dodatne specifikacije (Supplementary Specification),
10. Dijagram isporuke fizickog sustava (Deployment Diagram),
11. Lista rizika (Risk List),
12. Plan razvoja softvera (Software Development Plan),
13. Poslovni pogled na projekt (Business Case).

možda bude pitanje nabrojite 8 ili svih 13 dokumenata i modela LOL

Rjecnik pojmova

Pomocni dokument

Na jednom mjestu okuplja sve kljucne pojmove koji se koriste u ostatku projektne dokumentacije

Jedinstvenost upotrebe pojmova olakšava komunikaciju i razmjenu ideja

Razina dovršenosti: U dokument se dodaju pojmovi onog trenutka kad se uoce

Zahtjevi narucitelja projekta

Prikazuje sve zahtjeve narucitelja projekta koji su trenutno na snazi

Ažuriranje promjena zahtjeva tijekom rada na projektu.

Promjena se može referencirati ili opisati

Model slucajeva korištenja poslovnog sustava

Prikazuje pogled na poslovni sustav iz vana, s gledišta korisnika

Opisuje uloge u poslovanju i nacin na koji uloge koriste poslovni sustav

Olakšava razumijevanje poslovnog sustava i zadataka koje želimo riješiti informacijskim sustavom

Objektni model poslovnog sustava

Prikazuje unutarnji pogled na poslovni sustav, odnosno slucajeve korištenja iznutra

Opisuju uloge u poslovnom sustavu (radnici poslovnogsustava), entitete koji se koriste u sustavu, te nacin na koji se sve to povezuje s akterima pri realizaciji slucajeva korištenja poslovnog sustava.

Trebamo ici onoliko duboko i široko pri izradi modela koliko nam je potrebno poznavati iznutra poslovni sustav da bi razumjeli problem koji rješavamo te došli do ideje kako ga riješiti upotrebom informacijskog sustava.

Poslovna pravila

Opisuje poslovna pravila koja se koriste u poslovnom sustavu, a utjecu na buduci informacijski sustav.

Za fazu inicijacije nije bitna potpunost skupa poslovnih pravila, ali ih je dobro zabilježiti da se ne zaborave

Vizija

Dio je odgovora na pitanje: “Gdje su granice željenog sustava?"

Odreduje osnovne smjernice razvoja buduceg sustava

Predstavlja globalni pogled na sve najbitnije dijelove sustava s naglaskom na korisnika i problem koji se rješava ovim informacijskim sustavom

Bitno je što prije izraditi dio u kojem se opisuje pozicioniranje sustava. Ostatak dokumenta je zbirni pregled ostalih dokumenata, koji se ažurira tijekom projekta

Model slucaja korištenja

Prikazuje vanjski pogled na sustav

Opisuju se akteri i nacin na koji oni koriste informacijski sustav. Potrebno je uociti sve aktere i nacine korištenja, te ih kratko opisati. Malo detaljnije se razraduju kljucni slucajevi korištenja, te po potrebi i akteri. Razradu treba zaustaviti kad smo sposobni uociti sve kriticne rizike, te imamo dovoljno informacija da možemo izraditi grubu sliku arhitekture sustava.

Dodatne specifikacije

Odreduje nefunkcionalne zahtjeve prema buducem sustavu.

Nefunkcionalni zahtjevi nisu vezani uz odredeni slucaj nacin korištenja, nego se odnose na ponašanje, izvore rizika, mogucnost modifikacije i sl.

Dijagram isporuke fizickog sustava

Odgovara na pitanje "Što je kostur moguce arhitekture sustava?".

Prikazuje grubu sliku osnovne arhitekture sustava.

Dijagram se detaljizira do razine koja je dovoljna za odlucivanje o tome da li je moguce izgraditi fizicki sustav koji zadovoljava postavljene zahtjeve i ogranicenja.

Lista rizika

Odgovara na pitanje "Koji su kriticni rizici, te na koji nacin treba savladati ili zaobici one rizike koji mogu uzrokovati ne izgradnju sustava?".

Dokument na jednom mjestu objedinjuje rizike koji prijete sustavu, te pored opisa i mogucih posljedica prikazuje i nacin na koji se mislimo "boriti" s svakim pojedinim rizikom.

Potrebno je uociti sve kriticne rizike i one malo manje kriticne, te ih detaljno opisati i izraditi plan rješavanja problema. Ostale rizike koje uocimo možemo samo nabrojati, pošto nam u ovoj fazi nisu još bitni.

Plan razvoja softvera

Odgovara na pitanje "Kakav je okvirni plan izrade itroškova projekta?".

U ovoj fazi potrebno je staviti naglasak na grubom planiranju vremenskog redoslijeda, te planiranju troškova. S obzirom na uocene rizike i obim projekta ponekad je moguce zahtijevati izradu i drugih vrsta planova.

Poslovni pogled na projekt

Odgovara na pitanje. "Da li predloženi sustav stvarno rješava problem korisnika te koje su sve dobiti od sustava?".

Poslovni pogled na troškove (štete) i koristi (dobitke) od informacijskog sustava.

Nužno je kvantificiranje iskazanih troškova i koristi, a po mogucnosti i opis nacin njihovog utvrdivanja

Dokument treba biti u potpunosti dovršen na kraju faze inicijacije.

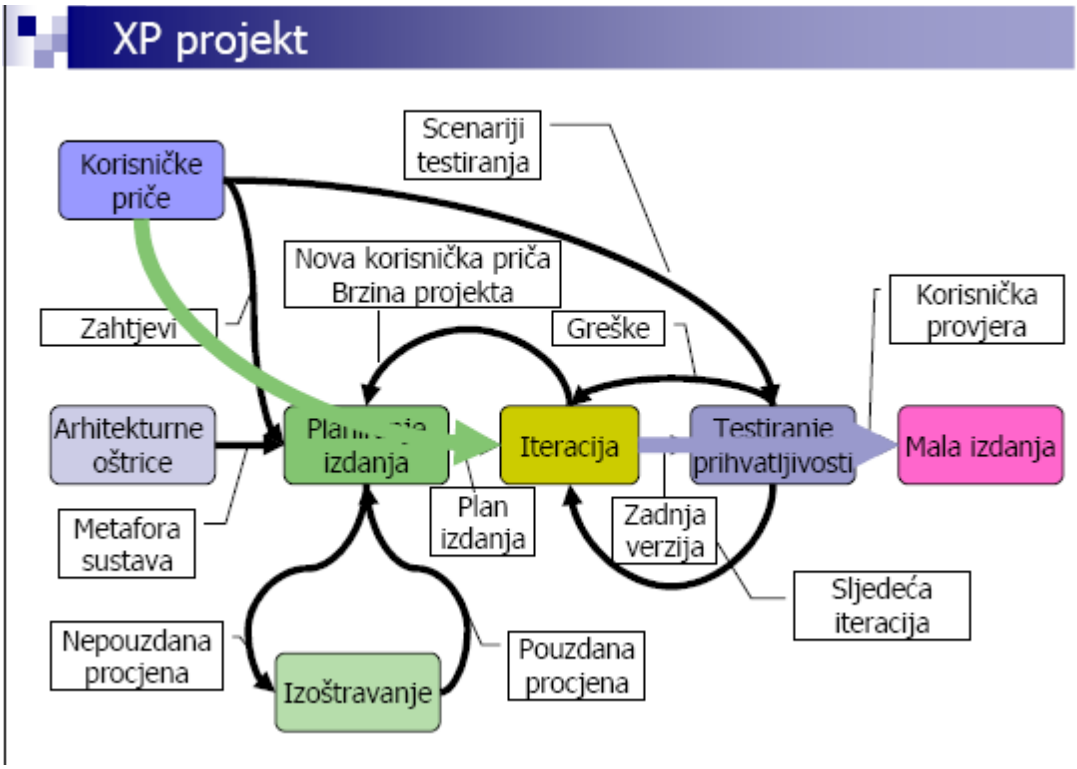
Podjela uloga

1. Analiticar
2. Developer
3. Ispitivac
4. Manager
5. Proizvodnja i podrška

8. Ekstremno programiranje XP – Extreme Programming

Ekstremno programiranje je promišljen i discipliniran metodološki pristup razvoju programa nastao 1996. godine

Skracenica XP nema nikakve veze sa Windowsima XP LOL



Ekstremno programiranje je:

- Slobodan, ali i disciplinirani metodološki pristup razvoju programa
- Proces eksperimentiranja s praksom i poboljšanja rezultata programiranja
- Istovremeno preuzimanje individualne odgovornosti i razvijanje vrijednosti tima

Ekstremno programiranje nije:

- Standard, metodika, obrazac procesa
- Jezik, metoda, tehnologija

Podrucje primjene

Rizicni projekti u kojima korisnicki zahtjevi nisu stabilni

“Lagani” pristup – što manje administriranja i dokumentacije

Timski razvoj u malim timovima, uz veliku participaciju korisnika

Opcenito, projekti pogodni za evolutivni i prototipni razvoj:

- tipicno desetak iteracija (npr 10-30 covjek*tjedan)
- uspješno primijenjen i na velikim projektima (10 covjek*godina)

Stil

Projekt je podijeljen na jedinice koda – cjelina koja se može testirati

Naglasak na ljudskom aspektu (sastanci s nogu, programiranje u paru, poštivanje rokova, medusobna odgovornost)

XP tim radi u jednoj sobi, racunala su centralno smještena

Klijent je neprestano prisutan

Nitko ne smije raditi prekovremeno uzastopno dva tjedna

Nema specijalizacije clanova tima, svi rade na specifikacijama, dizajnu, kodiranju i testiranju

Ne postoji izrazita faza dizajna. **Model evoluira i modificira se iterativno.**

"Programiranje u paru"

- Svaka jedinica koda je rezultat rada dva programera koji rade na jednom racunalu
- Vozac-navigator – vozac piše kod, a navigator predlaže alternative, usmjerava, "buši" rješenja
- Povecanje kvalitete uz isti utrošak vremena (kodiranje traje duže, ali otklanjanje grešaka je krace)
- Neiskusni clan uci uz iskusnog
- Sva znanja su podijeljena, što smanjuje rizik
- Sporije kodiranje

"Zajednicko vlasništvo nad kodom"

Svaki clan tima ima pristup svakom dijelu koda sustava

Inicijalni razvoj jedinice koda povjeren je odredenom paru

Svatko smije vidjeti, koristiti i promijeniti svaki dio koda, uz uvjet "ako pokvariš, dužan si popraviti"

Za:

- Povecanje kvalitete zbog provjera na ravnopravnoj razini (peer review)
- Razumijevanje dijelova sustava izvan najužeg podrucja interesa
- Uvid u tuda iskustva i praksu – organizacijsko znanje, uceca organizacija

Protiv:

- Efekt “hidre”
- Problem skrivene semantike
- Nužno je disciplinirano upravljanje konfiguracijom
- Gubi se pojam osobne zasluge

Ucestalo integriranje

Parovi unutar tima inkrementalno razvijaju dijelove sustava, vodeci brigu o cjelini

Promijenjena jedinica koda (nova verzija) koja je prošla jedinicno testiranje zamjenjuje staru u repozitoriju

Integriranje koda u repozitorij na dnevnoj bazi ili cešce, ali nakon jedinicnog testiranja

Za:

- Greška ili drugi problem se lakše izoliraju na promijenjenu jedinicu koda
- Izvor greške ili problema kompatibilnosti uocava se vec kod integracijskog testiranja
- Izbjegnuta je duga i zahtjevna faza integracije nakon razvoja svih jedinica koda

Protiv:

- Teško postizanje potrebne discipline
- Utrošak vremena na integriranje

Naglašeno testiranje

prije izrade jedinice koda piše se (automatizirani) jedinicni test, minimalna jedinica koda je komad programa koji zadovoljava jedinicni test

Sav kod mora biti podvrgnut jedinicnom testiranju

Svako kasnije izdanje mora zadovoljiti jedinicno testiranje

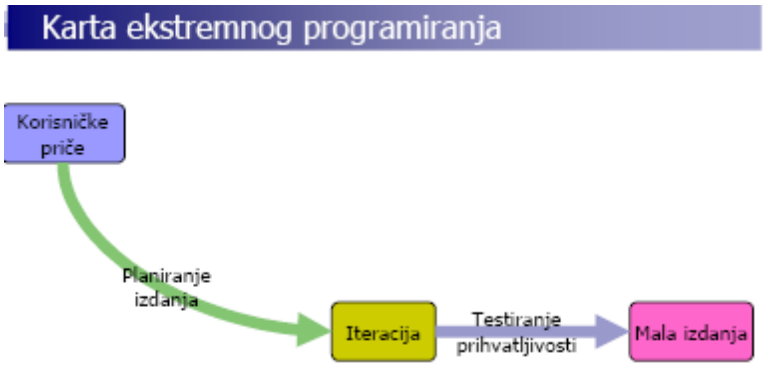
Za svaku naknadno otkrivenu grešku treba razviti test koji doticnu grešku otrkriva

Za:

- Programeri su usmjereni na kvalitetu, zadatak je zadovoljavanje testa

Protiv:

- Degeneriranje pristupa kvaliteti



Iteracija

Jedinica koda mora imati veliku unutarnju povezanost (koheziju) - manja kompleksnost znaci lakše testiranje, smanjivanje mogucnosti greške i vecu ponovnu iskoristivost

Problem integracije – postizanje optimuma unutarnje i vanjske povezanosti

Svaka iteracija sadrži dizajn i kodiranje!

Korisnicke price

Koriste se umjesto opširnih specifikacija zahtjeva ili "tehnickih" mini-specifikacija

Pišu ih korisnici i sadrže njihove zahtjeve – nekoliko recenica teksta bez tehnickih detalja, tehnicke terminologije ili detalja izvedbe (npr. sucelja, structure baze ili algoritama)

Planiranje

Planiranje izdanja na temelju korisnickih prica – plan izdanja odreduje koje ce se korisnicke price (slucajevi upotrebe) realizirati u pojedinoj iteraciji. **Plan se donosi konsenzusom narucitelja, korisnika i izvoditelja.**

Arhitekturna oštrica je jednostavni prikaz ili model koji služi za istraživanje potencijalnih problema i rješenja. To može biti i primjer tudeg rješenja.

Podjela razvojnog ciklusa na iteracije, ciji su izlazni rezultati mala izdanja

Svaka iteracija zapocinje planiranjem iteracije

Svaki dan zapocinje sastankom s nogu – tim se sastaje svakodnevno, kratki operativni sastanci, clanovi stoje u krugu – Just-in-time planiranje

Mjere za opstanak projekta su cirkuliranje ljudi i popravljanje XP-a.

Brzina projekta je odredena korisnickim pricama (slucajevima upotrebe) koje su završene u pojedinoj iteraciji. **Mjerenje brzine projekta je kontrolni mehanizam.**

Pravila dizajna

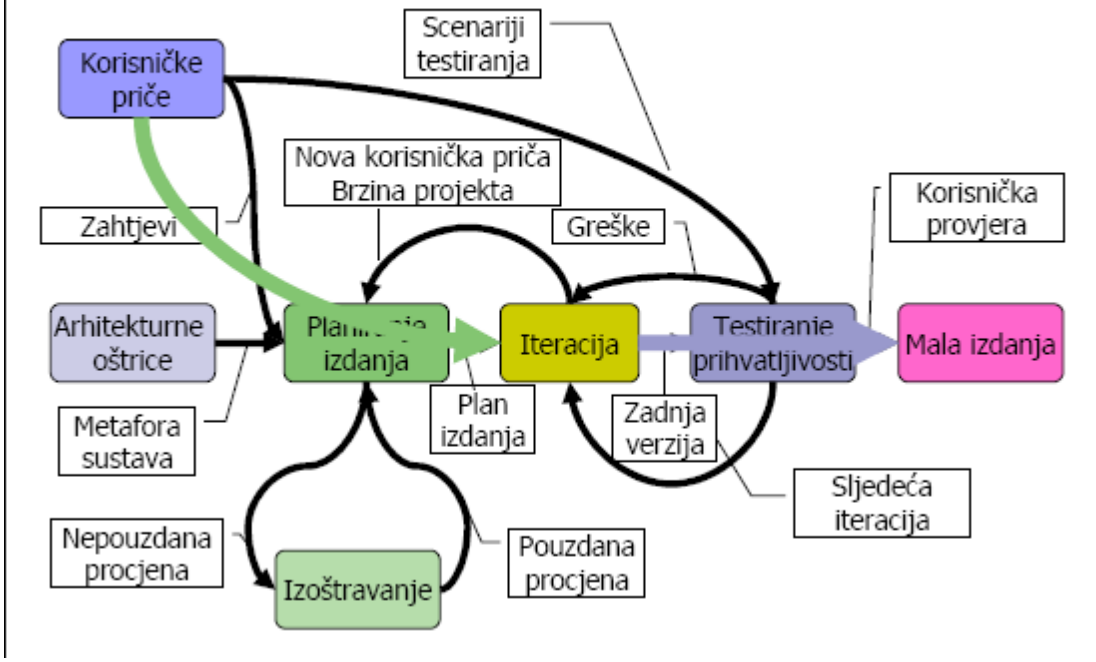
1. Jednostavnost
2. Korištenje metafora
3. CRC Cards (Class, Responisbilities, Collaboration) – osnovna tehnika objektno-orijentiranog dizajna
4. Izoštravanje rješenja kao mjera za reduciranje rizika
5. Što kasnije dodavanje funkcionalnosti
6. Refaktoriziranje – poboljšanje koda uklanjanjem redundancija i pojednostavljivanjem

Pravila kodiranja

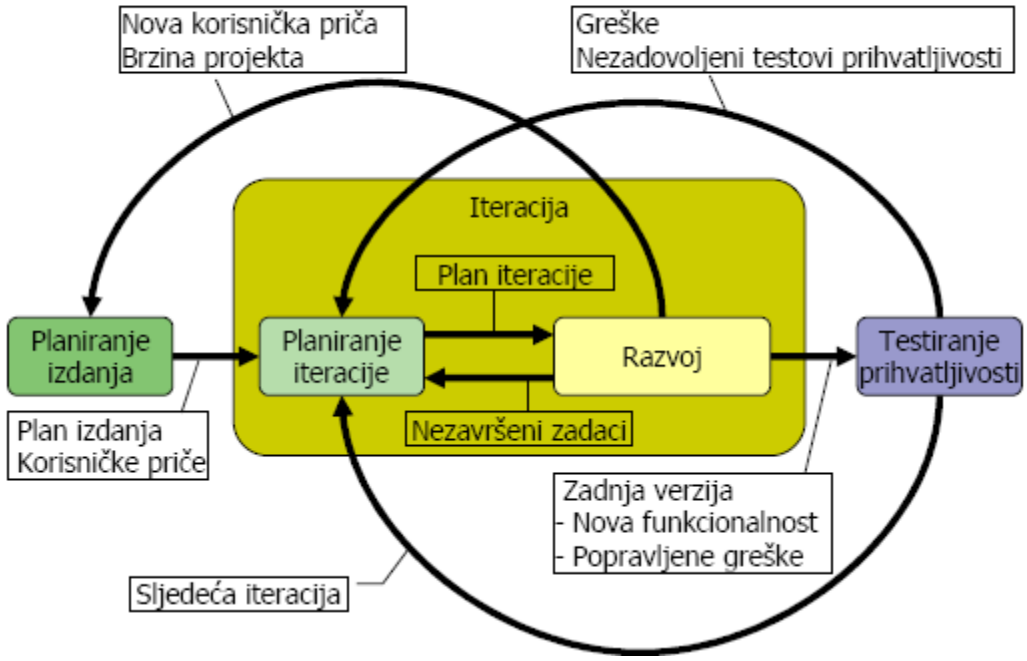
1. Korisnik je uvijek dostupan
2. Poštivanje dogovorenih standarda
3. Testiranje jedinica koda (Unit testing)
4. Programiranje u paru (Pair programming)
5. Samo jedan par integrira kod u jednom trenutku
6. Ucestalo integriranje
7. Zajednicko vlasništvo nad kodom
8. Optimiranje što kasnije
9. Obavljanje poslova u roku

još jedno moguće pitanje u stilu nabroji 6 ili svih 9 pravila LOL

XP projekt



Iteracija



Provjera sustava

ako se sustavi ne provjere prije nego udu u uporabu, korisnici otkrivaju greške tijekom rada, funkcionalnost im ne odgovara

ovakav pristup ne valja, sustav je nužno provjeriti prije isporuke

provjeru ne valja obavljati niti neposredno prije isporuke, što dovodi do prekoracenja rokova

XP i testiranje sustava

XP favorizira povratnu vezu (feedback) na nacin da se u svim iteracijama provjerava kako sustav radi i da li radi kako korisnik ocekuje

Provjera:

1. Težnja za postizanjem zadane kvalitete, uz planirane troškove i na vrijeme
2. Inženjersko projektiranje cija je osnova modeliranje
3. Specijalizacija i podjela posla
4. Interdisciplinarnost i suradnja s korisnicima
5. Primjena pravila struke, metodika i normi
6. Modularni pristup
7. Dokumentiranje, provjera i vrednovanje rezultata rada
8. Upravljanje rezultatima
9. Organizacija poduhvata razvoja (projektna, matricna ...) i upravljanje poduhvatom

   još jedno moguće pitanje u stilu nabroji 6 ili svih 9 pravila LOL