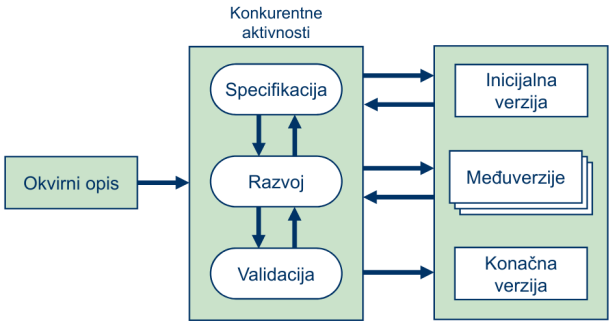
1) Šta je softver i čime se bavi softversko inženjerstvo?
   Softver je računarski program, pridružena dokumentacija i konfiguracioni podaci neophodni za njegovo funkcionisanje. Softversko inženjerstvo je inženjerska disciplina koja se bavi aspektima proizvodnje, koji obuhvataju teoriju, metode i alate za profesionalni razvoj softvera. Postoje dva tipa softvera – generički, namenjen za prodaju na slobodnom tržištu i ugovorni, namenjen za konkretnog korisnika. Osnovna razlika između dva je u tome ko definiše šta softver radi. Kod generičkog to je proizvođač softvera, a kod ugovornog to je naručilac softvera.

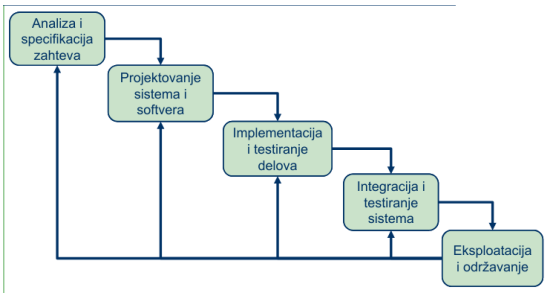
2) Navesti i kratko opisati atribute dobrog softvera
   Softver treba ima funkcionalnosti i performanse u skladu sa zahtevima korisnika. Takođe, mora da bude pogodan za održavanje, da bude upotrebljiv i da uliva poverenje.
   Pogodnost za održavanje – treba da je u stanju da se lako menja
   Stabilnost – podrazumeva da je softver pouzdan, bezbedan i siguran
   Efikasnost – softver mora ekonomično da koristi resurse sistema (procesorsko vreme i memoriju)
   Upotrebljivost – softver je pogodan za korišćenje, ima odgovarajući korisnički interfejs i odgovarajuću dokumentaciju.

3) Po čemu se softversko inženjerstvo razlikuje od informatike?
   Nauka o računarstvu ili informatika se bavi teorijom i osnovama računarstva. Softversko inženjerstvo se bavi praktičnom stranom proizvodnje i isporuke softvera. Teorija nauke o računarstvu je dovoljno razvijena i obezbeđuje solidnu osnovu za softversko inženjerstvo.

4) Koje su razlike između sistemskog i softverskog inženjerstva?
   Sistemski inženjering se bavi svim aspektima razvoja, uključujući hardverski, softverski i procesni inženjering. Softversko inženjerstvo se smatra delom sistemskog inženjerstva i bavi se praktičnim aspetkom softvera. Sistemsko inženjerstvo je starija disciplina od softverskog.

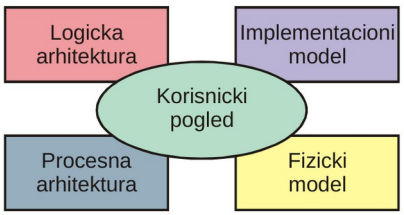
5) Grafički ilustrovati i opisati inkrementalni razvoj. Navesti osnovne prednosti i nedostatke
   Umesto da se ceo sistem isporučuje od jednom, razvoj i isporuku moguće je podeliti na inkremente, pri čemu svaki inkrement obezbeđuje deo tražene funkcionalnosti.
   Prednosti korišćenje inkrementalnog razvoja:
   - Smanjena cena izmene zahteva korisnika u toku razvoja – Količina analize i dokumentacije koju treba ponovo uraditi je manja nego kod modela vodopada.
   - Olakšano dobijanje povratne informacije od korisnika – Korisnik ima uvid i mogućnost komentarisanja trenutno implementiranih funkcionalnisti, a samim tim i uvid u celokupni napredak realizacije projekta.
   - Brže isporučivanje korisnog softvera naručiocu – Korisnici mogu da koriste softver pre nego kod modela vodopada
   Nedostaci korišćenja inkrementalnog razvoja:
   - Proces nije vidljiv – menadžerima je potrebno redovno isporučivanje kako bi merili napredovanje. Ako razvoj teče brzo, nije efikasno praviti dokumentaciju za svaku verziju sistema.
   - Struktura sistema ima tendenciju degradacije sa dodavanjem novih inkremenata – ako se novac ne ulaže u refaktorisanje i unapređenje softvera, izmene sistema za posledicu imaju degradaciju strukture. Nove izmene onda postaju skuplje i složenije za dodavanje.





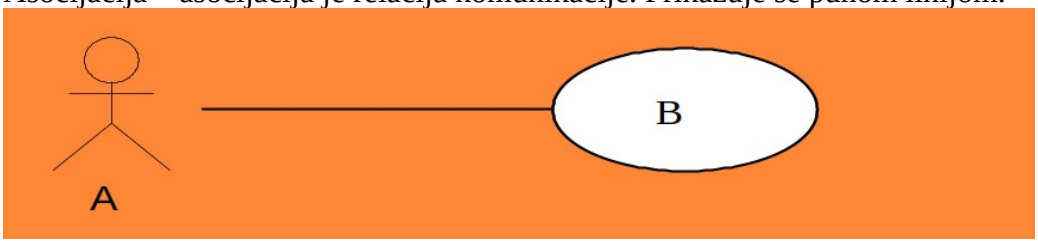
6) Razvoj softvera po modelu vodopada (dijagram, prednost i problemi)
Prednost modela vodopada – model vodopada se najčešće koristi kod velikih sistema čiji razvoj je podeljen po različitim lokacijama. U takvim situacijama planom vođena priroda modela pomaže u koordinisanju posla.
Nedostatak modela vodopada – osnovni nedostatak modela vodopada je nemogućnost efikasnog prihvatanja izmena u zahtevima korisnika kada je proces u toku. U principu, jedna faza mora biti završena, kako bi naredna otpočela, pa izmene zahteva resetuju celi proces. Takođe, nefleksibilna podela projekta na disjunktne faze otežava mogućnost reakcije na izmene korisničkih zahteva. Ovaj model jedino je pogodan za primenu kada su zahtevi dobro poznati na početku projekta, sa malom verovatnoćom promene.

7) Grafički ilustrovati i kratko opisati 4+1 model sistema.
Logička arhitektura opisuje najvažnije klase u sistemu i njihovu organizaciju u pakete i podsisteme, kao i organizaciju paketa i podsistema u nivoe (layers). Za prikazivanje logičke arhitekture koriste se dijagrami klasa. Postoji mogućnost automatskog generisanja koda na osnovu dijagrama klasa.
Procesna arhitektura opisuje najvažnije procese i niti (threads) u sistemu. Procesi se izvršavaju paralelno u odvojenim adresnim prostorima, dok su niti lightweight procesi koji mogu da se izvršavaju paralelno sa drugim procesima ili nitima, ali dele adresni prostor sa drugim nitima koje pripadaju istom procesu. Za prikazivanje procesne arhitekture koriste se dijagrami klasa.
Implementacioni model – za prikazivanje implementacionog modela koriste se dijagrami komponenti.
Fizički model – opisuje fizičke čvorove sistema i njihovo razmeštanje u prostoru. Za prikazivanje fizičkog modela koriste se dijagrami raspoređivanja.

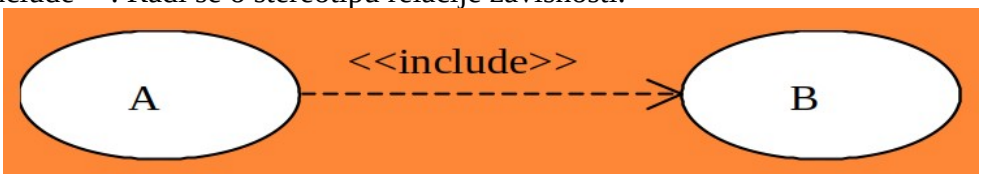


8) Navesti osnovne principe agilnog razvoja softvera izražene kroz Agile Manifesto
- Jednostavnost
- Samoorganizovani timovi
- Kontinualno usmeravanje pažnje na tehničke veštine i dobar dizajn
- Prilagođavanje promenljivim okolnostima
- Često isporučivanje softvera, u razmaku od par nedelja
- Ispravnost softvera je osnovna mera napretka
- Najbolji tip komunikacije je komunikacija "licem u lice"
- Bliska saradnja između projektanata i poslovnih saradnika
- Zadovoljstvo korisnika čestim isporukama softvera
- Mogućnost izmene zahteva korisnika, čak i u poodmaklim fazama razvoja
- Razvoj koji može da održi konstantan tempo

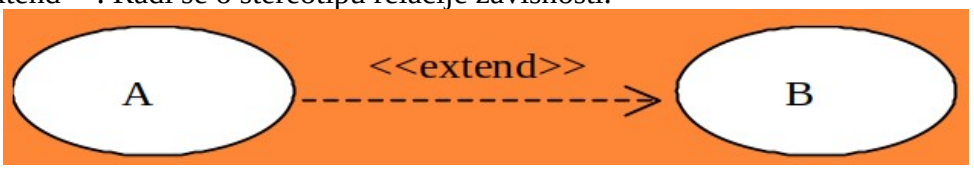
9) Objasniti osnovne tipove veza kod Use-case dijagrama i ilustrovati jednim dijagramom
a) Asocijacija – asocijacija je relacija komunikacije. Prikazuje se punom linijom.



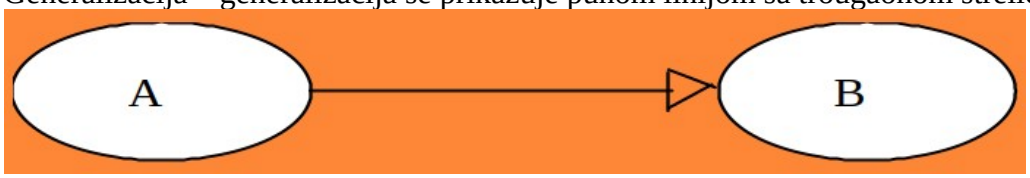
Na strai A je akter, a na strani B slučaj korišćenja. Komunikaciju može inicirati i strana A i strana B, što znači da je veza bidirekciona.
b) Uključivanje – uključivanje se prikazuje isprekidanom linijom sa strelicom i natpisom <<include>>. Radi se o stereotipu relacije zavisnosti.



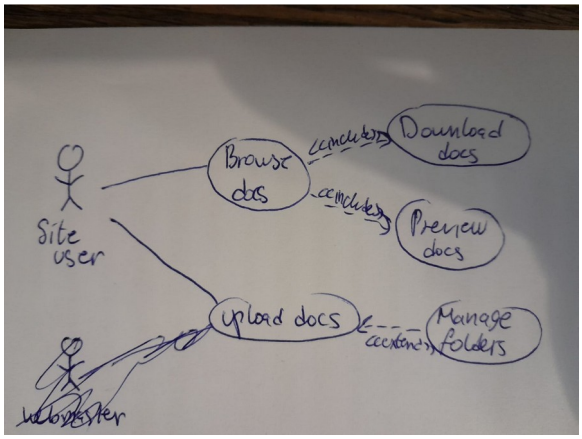
Relacija uključivanja od slučaja korišćenja A prema slučaju korišćenja B ukazuje da će slučaj korišćenja A uključiti i ponašanje slučaja korišćenja B. Ponašanje opisano u B je obavezno za A.
c) Proširivanje – proširivanje se prikazuje isprekidanom linijom sa strelicom i natpisom <<extend>>. Radi se o stereotipu relacije zavisnosti.



Relacija proširivanja od slučaja korišćenja A prema slučaju korišćenja B ukazuje da slučaj korišćenja B može obuhvatiti ponašanje specificirano slučajem korišćenja A. Praktično, slučaj korišćenja B se može proširiti i ispoljiti celokupno ponašanje opisano u A. Relacija proširivanja se koristi da ukaže na opciono ponašanje osnovnog slučaja korišćenja. A je opciono ponašanje, a B osnovno. Tačka u kojoj se slučaj korišćenja proširuje naziva se tačka ekstenzije.
d) Generalizacija – generalizacija se prikazuje punom linijom sa trougaonom strelicom.



Relacija generalizacije od slučaja korišćenja A prema slučaju korišćenja B ukazuje da je slučaj korišćenja A specifičniji slučaj od opštijeg slučaja B.



10) Šta predstavlja upravljanje zahtevima?

Svaki od funkcionalnih i nefunkcionalnih zahteva evoluira kroz različite fokuse i perspektive definišući različite nivoe detalja u zahtevima. Svaka slučajna praznina u zahtevima povećava rizik propadanja softvera i dodavanja novih mana.

Upravljanje zahtevima je više od dokumentovanja istih. Šest preporuka "dobre prakse" prilikom upravljanja zahtevima su:

1) Odvojite dovoljno vremena za proces definisanja zahteva
2) Izaberite pravi način za iznošenje zahteva
3) Prenesite – saopštite zahteve svim interesnim stranama
4) Razvrstajte zahteve po prioritetu
5) Zahtevi za ponovno korišćenje – reuse
6) Pratite zahteve kroz životne cikluse softvera

Proces upravljanja zahtevima se sastoji od nerazdvojivih podprocesa, koji se ne mogu zasebno i sekvencijalno odvijati:

1) Prikupljanje zahteva
2) Analiza zahteva
3) Specifikacija zahteva
4) Validacija zahteva

11) Navesti i kratko opisati kategorije zahteva

Funkcije – "šta" sistem mora da bude sposoban da uradi
Osobine – "koliko dobro" će sistem izvršavati funkcije
Cena – koliko će koštati (bilo koji ulazni resurs – novac, vreme ili ljudi) kreiranje i održavanje funkcija i njihovih osobina
Ograničenja – bilo koja restrikcija u slobodi definisanja zahteva ili dizajnu

12) Tipovi zahteva

Nije kvantifikovan, ali je testiranjem moguće utvrditi da li je realizovan
Kvantifikovan (na mernoj skali)
Funkcije se ne mogu kvantifikovati, ili su ili nisu prisutne
Osobine i cena mogu i moraju biti kvantifikovane
Ograničenja mogu biti bilo kog tipa

13) Lažni zahtevi

1. Zahtevi koji nisu iskazani na kvantifikovan i merljiv način
- "Smanjena cena"
- "Poboljšana kontrola upravljanja"
- "Povećano zadovoljstvo korisnika"
- "Visoka upotrebljivost"

2. Ideje o projektovanju umesto stvarnih zahteva
- "Obezbediti Windows Xp interfejs"
- "Koristiti objektno orijentisanu bazu podataka"
- "Poboljšano korisničko uputstvo"

14) Navesti i kratko opisati kategorije strategija za upravljanje rizikom

1. Strategija izbegavanja
- Smanjiti verovatnoću pojave rizika
- Primenjeno kod defekata u komponentama

2. Strategija minimizacije
- Smanjiti uticaj rizika na projekat ili proizvodn
- Primenjeno kod bolesti osoblja

3. Planovi za nepredviđene događaje
- U slučaju nastupanja rizika, ovi planovi definišu kako raditi sa rizikom
- Primenjeno kod finansijskih problema organizacije

15) Use-case metoda i scenariji događaja. Ilustrovati na primeru

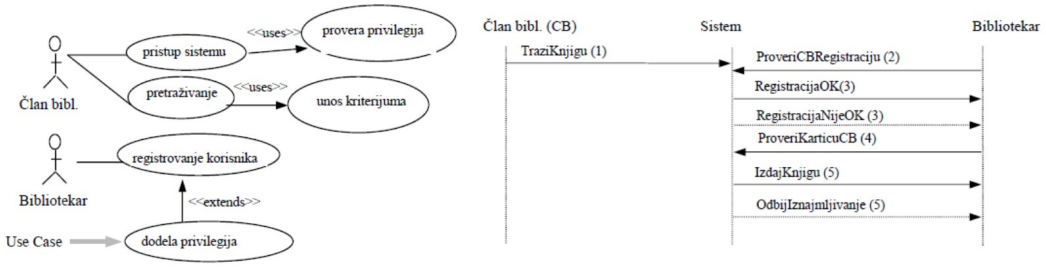
Slučajevi korišćenja predstavljaju tehiniku, baziranu na scenarijima i zapisanu pomoću UML dijagrama, koja identitfikuje sve aktere u sistemu i slučajeve korišćenja sistema.

Use-case metoda:

- Sistem se posmatra sa stanovišta korisnika sistema
- Opisuju se slučajevi korišćenja sistema i scenarija ponašanja
- Koristi se UML notacija
- Koristi se kod RUP modela razvoja softvera
- Servisi objekata mogu biti otkriveni i modeliranjem scenarija događaja za različite funkcije sistema
- Događaji se prati sve do objekata koji reaguju na njih
- Tipičan model scenarija događaja je interakcija korisnika i sistema

Scenarija događaja su primeri kako će sistem biti korišćen u realnom životu. Oni treba da uključe:

- Opis početne situacicje
- Opis normalnog toka događaja
- Opis izuzetka (ako se izađe iz normalnog toka)
- Informacije o konkurentnim aktivnostima
- Opis stanja kada se scenario završi



16) Navesti koji sve sastanci po Scrum-u postoje i kratko ih opisati

Postoje četiri vrste sastanaka:

1) Planiranje sprinta

- Tim bira stavke iz product backloga za koje može da se obaveže da može da ih završi
- Kreiranje product backloga (Identifikacija zadataka)
- uz saradnju, ne samo scrum master
- uzima se u obzir i high level dizajn rešenja

2) Dnevni scrum sastanak

- Parametri - Dnevno - Ne rešavaju se problemi
- 15 minuta - Stand up - Svi su pozvani
- Samo članovi tima, scrum master i vlasnik proizvoda smeju da pričaju
- Služi kako bi se izbegli nepotrebni sastanci

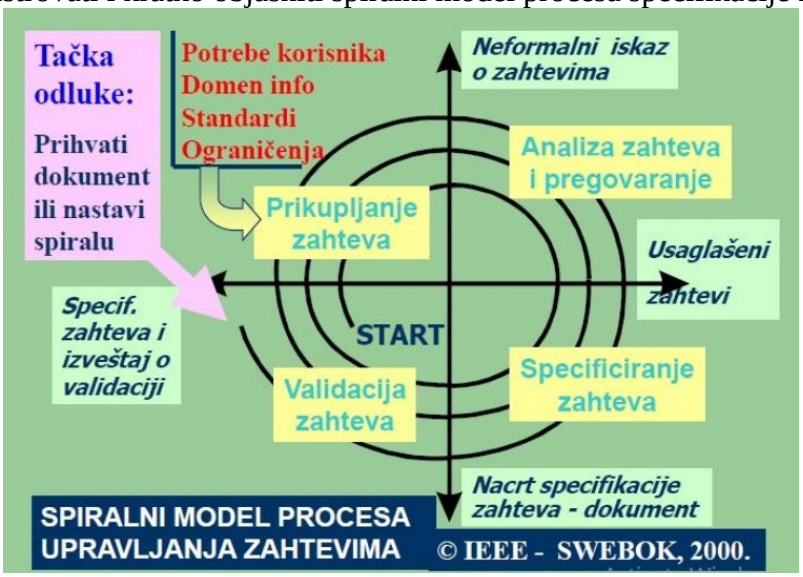
3) Pregled sprinta

- Tim prezentuje šta je urađeno za vreme sprinta
- Tipično u formi demonstracije novih funkcionalnosti ili upotrebljene arhitekture
- Neformalno
- Pravilo 2-satne pripreme
- Bez slajdova
- Svi su pozvani
- Učestvuje ceo tim

4) Retrospektiva sprinta

- Periodično razmatranje šta je dobro, a šta ne
- 15-30 minuta
- Učestvuje ceo tim
- Obavlja se nakon svakog sprinta

17) Grafički ilustrovati i kratko objasniti spiralni model procesa specifikacije zahteva



1. Prikupljanje zahteva – prikupljaju se informacije o potrebama korisnika, informacije o domenu, standardi i ograničenja. Kao posledica ove faze dobija se neformalni iskaz o zahtevima.
2. Analiza zahteva i pregovaranje – u ovoj fazi se analiziraju prikupljeni zahtevi iz prethodne faze i pregovara se sa korisnikom u cilju usaglašavanja zahteva i poboljšavanja komunikacije. Kao posledica ove faze dobijaju se usaglašeni zahtevi između korisnika i proizvođača.
3. Specificiranje zahteva – usaglašeni zahtevi iz prethodne faze se formalizuju i dobija se nacrt specifikacije zahteva.
4. Validacija zahteva – formalizovani zahtevi iz predhodne faze se proveravaju i validiraju i kao posledica dobijaju se specifikacija zahteva i izveštaj o validaciji.

Nakon 4. faze potrebno je odlučiti da li je dokument dovoljan ili je još jednom potrebno proći kroz sve faze od 1 do 4.

18) Šta je Agile Manifesto?

Više vrede
- Pojedinci i interakcije nego procesi i alati
- Programska podrška koja radi nego dokumentacija
- Saradnja sa klijentima nego ugovori
- Odgovor na promene nego praćenje plana

19) Šta je sistem?

Sistem je kolekcija međusobno povezanih komponenata koje zajedno rade ka postizanju zajedničkog cilja. Sistem obuhvata hardver, softver i ljude. Svojstva i ponašanje komponenata u sistemu utiče na druge komponente sistema.

20) Zahtevi

Funkcionalni zahtevi
- KO (interfejs) – koji ljudi, organizacije i sistemi smeju (ili ne smeju) da pristupaju sistemu
- ŠTA (podaci) – informacije potrebne sistemu u smislu zadovoljenja funkcija
- GDE (mreža) – fizička ili internet lokacija koja mora imati pristup rešenju
- KADA (događaji) – vremensko usaglašavanje poslovnih događaja
- KAKO (proces) – zadaci i funkcije koje sistem mora da izvrši
- ZAŠTO (business rules) – poslovna pravila kojima sistem mora biti prilagođen

Nefunkcionalni zahtevi

U projektu takođe moraju biti definisana ograničenja. Kategorije rešenja: kulturološka, pravna, politička, nivo kvaliteta, nivo cene, funkcionalnost i dizajn.

21) Navesti i kratko opisati faze u RUP-u. Objasniti odnos faza i iteracija u RUP-u

22) Navesti faze RUP metodologije i objasniti šta se dobija kao rezultat svake od faza

Faze u RUP-u su:

1) Početna faza
2) Faza razrade
3) Faza izrade
4) Faza isporuke

Svaka faza može imati proizvoljan broj iteracija, a svaka iteracija (osim, naravno, početne) treba da rezultira izvršnom verzijom koju je moguće testirati.

1) Početna faza obuhvata analizu problema, razumevanje potreba (potencijalnih) korisnika, generalno definisanje sistema, upravljanje kad dođe do promene korisničkih zahteva. Rezultat početne faze je dokument – Vizija sistema. Vizija sistema ne sadrži puno tehničkih detalja kako bi bila razumljiva i korisnicima i razvojnom timu. Samo se koriste blok dijagrami za šematski prikaz sistema. Vizija sistema sadrži: pozicioniranje proizvoda, opis korisnika, opis proizvoda, funkcionalne zahteve, nefunkcionalne zahteve, kvalitet i ograničenja.

2) Faza elaboracije ili razrade obuhvata izradu plana projekta, organizaciju i ekipni rad, detaljno definisanje zahteva i definisanje arhitekture sistema. Kao rezultat ove faze dobijaju se:

- Plan projekta (plan faza, plan izrade, rezultat projekta, kontrolne tačke (milestones) i resursi)
- Use-case specifikacija (opis slučajeva korišćenja sistema, definisanje svih aktera u sistemu, određivanje arhitekturno najvažnijih slučajeva korišćenja)
- Arhitekturni projekat sistema (definisanje arhitekture sistema, definisanje najvažnijih klasa u sistemu, realizacija arhitekturno najvažnijih slučajeva korišćenja korišćenjem sequence dijagrama, dijagrami klasa)

3) Faza izrade obuhvata realizaciju sistema i testiranje. Rezultati ove faze su:

- Plan testiranja (zahtevi testiranja, strategije testiranja, tehnike testiranja, alati za testiranje, resursi i kontrolne tačke)
- Specifikacija testiranja (svaki od test case-ova treba da sadrži opis, ulaze, očekivane izlaze i završne radnje)
- Detaljni projekat sistema (arhitekturni projekat sistema razvijen u detalje, dijagrami klasa, "4 + 1" model sistema)
- Softverski proizvod

4) Faza isporuke uključuje finalizaciju softverskog proizvoda, alfa (beta) testiranje, izradu korisničke dokumentacije (uputstvo), obuka korisnika, uvođenje sistema kod korisnika. Rezultati ove faze su

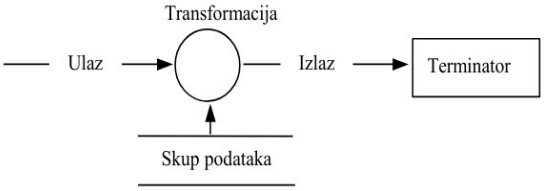
- Test izveštaj (rezultati izvršavanja test case-ova, pregled test rezultata (generalni pregled testiranog softvera, uticaj test okruženja, predložena poboljšanja))
- Korisničko uputstvo
- Instalacija sistema

23) Metode inženjeringa zahteva

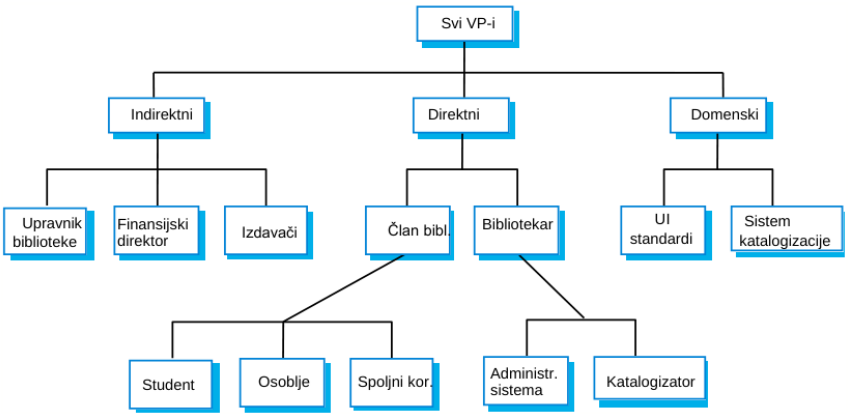
Metode inženjeringa zahteva definišu sistematski način za dobijanje modela sistema. Najznačajnije metode inženjeringa zahteva su:

1) DFD (data-flow-diagram) dijagrami
2) Metode koje koriste relacioni model podataka
3) Objektno orijentisane metode
4) Formalne metode
5) Metode zasnovane na ponašanju sistema
    - Use-case specifikacija
    - Viewpoint orijentisane metode

DFD dijagram



Metode koje koriste relacioni model podataka – za opisivanje zahteva koristi se prošireni ER model.
OO metode – nasledile su ER model. U procesu prikupljanja zahteva najvažniji elementi su objekti. Objekti se dobijaju analizom domena.
Formalne metode – Formalne metode su bazirane na matematičkoj formalnoj sintaksi i semantici. Najpoznatije formalne metode su B, Z, VDM, LOTOS metode. Ove metode obezbeđuju postizanje visoke sigurnosti da će sistem ispunjavati specifikaciju zahteva, ali ne garantuju apsolutnu korektnost sistema. Ove metode nisu doživele uspeh među softverskim inženjerima zato što su teške za razumevanje i iskodi nisu očigledni.
Viewpoint orijentisane metode - Osnovna ideja je da se sistem posmatra iz različitih tačaka posmatranja kako bi se pokušalo sagledavanje sistema sa svih strana, pri čemu se identifikuju svi validni zahtevi i razrešavaju se kontradiktorni zahtevi. Prednosi vp metoda su da se eksplicitno identifikuju svi izvori zahteva, dobija se osećaj kompletnosti, obezbeđuju načine za struktuiranje zahteva.
Tipovi vp-a:
Direktni aspekti posmatranja – ljudi ili sistemi koji imaju direktnu komunikaciju sa sistemom
Indirektni aspekti posmatranja – korisnici koji ne koriste sistem, ali imaju direktan utican na zahteve
Domenski aspekti posmatranja – karakteristike i ograničenja domena koji imaju uticaj na zahteve

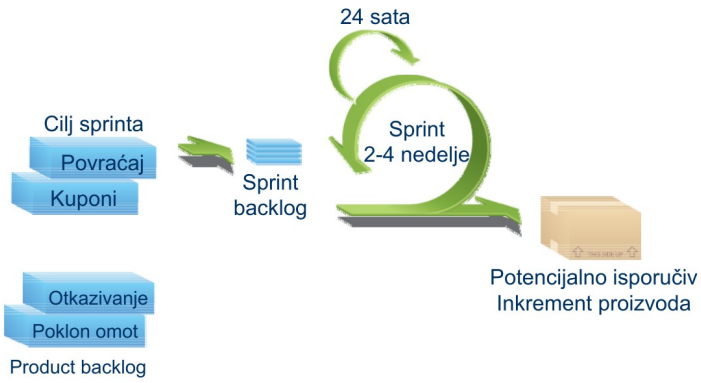


24) Šta je RUP?
Rup je skup parcijalno uređenih koraka namenjenih osnovnom cilju – da se efikasno i u predviđenom terminu korisniku isporuči kompletan sistem koji u potpunosti zadovoljava njegove potrebe. Rup je iterativni i inkrementalni proces, što znači da se do konačnog proizvoda dolazi nakon niza iteracija, pri čemu svaka iteracija inkrementalno dodaje funkcionalnosti do konačne verzije. Dobra osobina rup-a je da je izuzetno dobro dokumentovan.

25) Šest principa dobre prakse po RUP-u?
Šest osnovnih principa za uspešan razvoj softvera po RUP-u su:
- Iterativni razvoj softvera
- Upravljanje zahtevima – prevođenje zahteva u skup potreba korisnika i funkcija sistema koji će biti detaljno specificirani kroz funkcionalne i nefunkcionalne zahteve, zatim test

procedure, projekat i dokumentaciju. Takođe, potrebno je definisati procedure za slučaj izmene korisničkih zahteva.
- Komponentna arhitektura softvera – zadovoljava trenutne i buduće potrebe, povećava proširljivost, omogućava ponovno korišćenje i enkapsulira zahteve
- Vizuelno modeliranje – koristi se UML
- Kontinualna verifikacija kvaliteta – jeftinije je izvršiti verifikaciju faze nakon završetka, nego raditi ispravke nakon isporuke sistema
- Upravljanje promenama – praćenje statusa proizvoda, praćenje promena, odlaganje izvornog koda i kontrola verzija

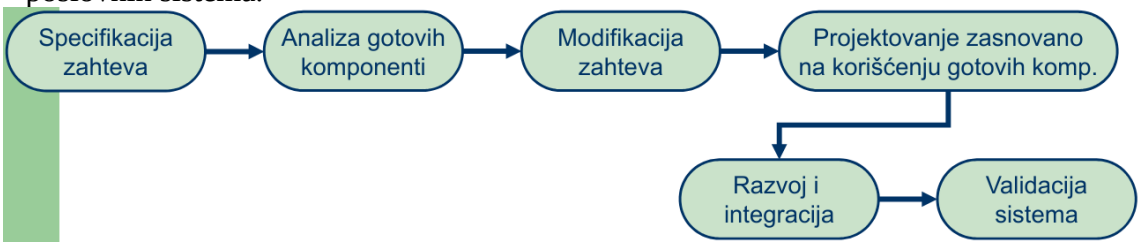
26) Grafički ilustrovati i opisati scrum



Scrum je agilni proces koji nam omogućava da se fokusiramo na isporuku najviših poslovnih vrednosti u najkraćem vremenskom roku. Takođe, omogućava često unapređenje aplikacije, svakih dve nedelje do mesec dana. Posao određuje prioritete. Timovi se samoorganizuju tako da mogu da odluče koji način je najpovoljniji za isporuku funckionalnosti za najvišim prioritetom.

27. Razvoj zasnovan na korišćenju gotovih komponenti

Razvoj zasnovan na sistematskom korišćenju ranije razvijenih komponenti ili komercijalno dostupnih komponenti. Korišćenje komponenata je standardni pristup za razvoj mnogih poslovnih sistema.



28. Navesti i kratko opisati zajedničke aktivnosti za različite softverske procese

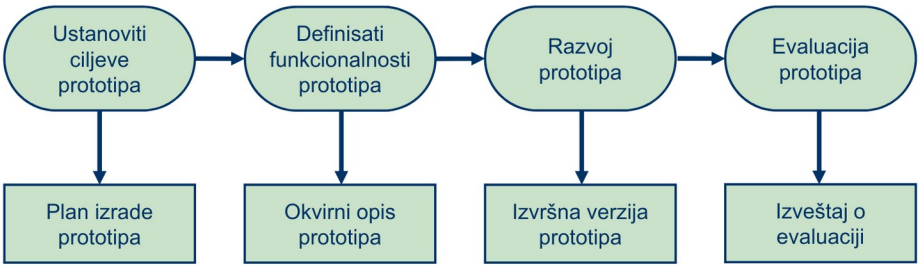
Specifikacija softvera – je proces određivanja servisa zahtevanih od strane korisnika, kao i ograničenja u pogledu funkcionisanja i razvoja sistema. Proces inženjeringa zahteva:
- Studija izvodljivosti – proverava se da li je razvoj tehnički i finanskijski izvodljiv
- Prikupljanje i analiza zahteva – određuje se šta zainteresovane strane žele i očekuju od sistema
- Specifikacija zahteva – detaljno definisanje zahteva
- Validacija zahteva – provera validnosti specificiranih zahteva

Projektovanje i implementacija softvera – je proces prevođenja specifikacije sistema u realan upotrebljiv sistem. Projektovanje se odnosi na projektovanje strukture sistema tako da realizuje zadatu specifikaciju. Implementacija se odnosi na prevođenje projektovane strukture sistema u izvršni program. Aktivnosti vezane za implementaciju i projektovanje su blisko povezane i često se preklapaju.
- Arhitekturno projektovanje se odnosi na projektovanje osnovne strukture sistema, osnovnih komponenata i način na koji su povezane i način distribucije
- Projektovanje interfejsa se odnosi na projektovanje interfejsa između komponenata
- Projektovanje komponenata se odnosi na definisanje željenog funkcionisanja za sve identifikovane komponente u sistemu
- Projektovanje baze podataka se odnosi na projektovanje struktura podataka i određivanje njihove reprezentacije u bazi podataka

Validacija softvera – Verifikacija i validacija softvera (V&V) za cilj ima da pokaže da sistem odgovara sopstvenoj specifikaciji i zahtevima naručioca. Uključuje proveru i pregled procesa, kao i testiranje softvera. Testiranje uključuje izvršavanje test slučajeva izvedenih iz specifikacije realnih podataka koje sistem obrađuje.
Evolucija softvera – softver je podrazumevano fleksibilan i podložan izmenaka. Kako se zahtevi menjaju promenom poslovnih okolnosti, tako i softver odgovoran za određene poslove mora da evoluira i da se menja. Često dolazi do mešanja između novog razvoja i evolucije postojećeg sistema, ali razlika postaje sve manje relevantna obzirom da imamo sve manje potpuno novih sistema.

29. Prototipovanje softvera

Prototip je inicjalna verzija sistema koja za cilj ima demonstraciju koncepata i proveravanje projektnih ideja. Prototip olakšava prikupljanje zahteva, projektovanje sistema, testiranje i otkrivanje grešaka. Prototip obično ne implementira sve funckionalnosti sistema. Prednosti korišćenja prototipovanja su povećana proširljivost sistema i lakše prikupljanje korisničkih zahteva. Protitipe treba odbaciti nakon razvoja, i iskoristiti ideje za razvoj sistema, zato što je zbog čestih izmena struktura prototipa degradirala i teško ju je uključiti u sistem.



30. Dodatna svojstva sistema

Radi se o svojstvima sistema kao celine. Dodatna svojstva su posledica međusobnih veza između komponenata sistema. Kako bi se dodatna svojstva određivala i merila prvo je potrebno izvršiti integraciju komponenti. Primeri dodatnih svojstava:
- Težina sistema (nije moguće odrediti je na osnovu sastava sistema)
- Upotrebljivost (ne zavisi samo od hw i sw već i od operatera koji upravlja sistemom)
- Pouzdanost (zavisi od pouzadnosti pojedinačnih komponenti ali i od njihovih međ. Veza)

31. Nepoželjne karakteristike ("Shall-not" svojstva)

Svojstva sistema kao što su performanse i pouzdanost se mogu meriti. Međutim postoje i svojstva sistema koja nije moguće meriti niti ih sistem može pokazivati (bezbednost, sigurnost). Ovakva svojstva je veoma teško određivati i meriti pa se ona definišu preko stvari koje sistem ne sme da dopusti.

32. Proces sistemskog inženjerstva

Najčešće se koristi model vodopada jer je potrebno paralelno razvijati različite delove sistema. U razvoju učestvuju inženjeri in različitih disciplina koji moraju da rade zajedno.