1. **Šta je sistem?**

* Sistem je kolekcija međusobno povezanih komponenti koje rade zajedno radi ostvarenja nekog zajedničkog cilja. Pod sistemom podrazumevamo softver, hardver (mehaničke, električne I elektronske komponente) I ljude. Svojstva I ponašanje svake komponente ponaosob utiče na ostale stavke Sistema.

1. **Šta je sistemsko inženjerstvo?**

* Sistemsko inženjerstvo obuhvata projektovanje, implementaciju, isporuku i eksploataciju sistema koji uključuju hardver,softver i ljude. To je inženjerska disciplina koja se bavi svim aspektima razvoja Sistema zasnovanih na računarima, uključujući hardverski, softverski I procesni inženjering. Softversko inženjerstvo je deo tog procesa. Sistem inženjeri su uključeni u aktivnosti specificiranja, arhitekturnog projektovanja, integracije I isporuke Sistema. Sistemsko inženjerstvo je starija disciplina od softverskog inženjerstva.

1. **Šta je softver?**

* Softver je računarski program, pridružena dokumentacija I konfiguracioni podaci neophodni da bi program radio korektno. Softverski *sistem* se obično sastoji od:   
   **-** *određenog broja programa* **-** *konfiguracionih fajlova* - *sistemske dokumentacije koja opisuje strukturu Sistema   
   - korisničke dokumentacije* ***-*** *veb sajtova za podršku korisnicima*

Razvojem softvera se bavi softversko inženjerstvo.

1. **Šta je softversko inženjerstvo?**

* Softversko inženjerstvo je inženjerska disciplina koja se bavi teorijom, metodama I alatima za profesionalni razvoj softvera. Sve više Sistema je softverski kontrolisano.

1. **Po čemu se razlikuje softversko inženjerstvo od informatike (nauke o računarstvu)?**

* *Nauka o računarstvu ili informatika* se bavi teorijom I osnovama računarstva, a softversko inženjerstvo se bavi praktičnom stranom razvoja I isporuke korisnog softvera
* Teorija nauke o računarstvu je trenutno dobro razvijena I obezbeđuje solidnu osnovu za softversko inženjerstvo.

1. **Upravljanje zahtevima (kraće)**

* Upravljanje zahtevima obuhvata prevođenje zahteva korisnika u skup njihovih potreba I funkcija Sistema. Ovaj skup se kasnije pretvara u detaljnu specifikaciju funkcionalnih I nefunkcionalnih zahteva. Detaljna specifikacija se prevodi u test procedure, projekat I korisničku dokumentaciju. U slučaju promene zahteva korisnika potrebno je definisati proceduru.

1. **Funkcionalni I nefunkcionalni zahtevi**

* Postoje 2 vrste zahteva: *funkcionalni I nefunkcionalni.* 
  1. **Funkcionalni** zahtevi su oni zahtevi koji se odnose na konkretne funkcije Sistema koje će biti obavljane I odgovaraju na sledeća pitanja:   
     *KO* (interfejs) – koji ljudi, organizacije ili sistemi smeju (ili ne) da pristupe sistemu.  
     *ŠTA*  (podaci) – koje informacije su potrebne sistemu radi zadovoljenja funkcija.   
     *GDE* (mreža) – koja fizička ili internet lokacija mora imati pristup rešenju.

*KADA* (događaji) – vremensko usaglašavanje poslovnih događaja.

*ZAŠTO (*business rules) – kojim poslovnim pravilima mora biti prilagođen sistem

*KAKO* (process) – koje zadatke ili procesiranje I funkcije sistem mora da izvrši.

* 1. **Nefunkcionalni zahtevi –** neka ograničenja u projektu koja se moraju definisati a ne sadrže konkretne informacije o funkcijama Sistema. Kategorije tih ograničenja su:
     + *Nivo kvaliteta*
     + *Nivo cene*
     + *Funkcionalnost*
     + *Dizajn*
     + *Politička ograničenja*
     + *Kulturološka*
     + *Pravna*

1. **Lažni zahtevi**
   1. *Zahtevi koji nisu iskazani na kvantifikovan I merljiv način.* Npr: (“**redukovana cena” , “poboljšanje kontrole upravljanja”**

**, “visoka upotrebljivost”**

**,“povećano zadovoljstvo kupca”**) I sl.

* 1. *Ideje o projektovanju umesto stvarnih zahteva. Npr:* ( **“zahteva se objektno orijentisana baza podataka”**

**, “potrebno je poboljšano uputstvo za rukovanje”**

**, “obezbediti Windows XP intefejs”**).

1. **Extreme programming (XP)**

* Najpopularnija agilna metoda projektovanja softvera, nastala 1999. Godine. Timovi koji se bave razvojem softvera se sastoje od malog broja članova. Proizvodi se dugotrajni softver I karakteriše je sposobnost reagovanja na promene zahteva. Životni ciklus XP metode se sastoji od:
  + ***Istraživanja*** *(pisanje priča, upoznavanje tehnologija)*
  + ***Planiranje*** *(utvrđivanje potrebnog vremena, prioriteti, stvaranje)*
  + ***Iteracije*** *(pisanje, procena I dodeljvivanje prioriteta novim zadacima, testiranje na kraju svake iteracije)*
  + ***Proizvodnja*** *(testiranje pre isporuke, evidentiranje novih zahteva)*
  + ***Održavanje*** *(dodavanje novih funkcionalnosti, ispravke grešaka)*
  + ***Kraj*** *(zahtevi ispunjeni, završetak projekta)*

XP metodologija se temelji na 5 osnovnih principa: ***jednostavnost, komunikacija, povratna sprega, hrabrost I poštovanje.***

Da bi se dostigle vrednost ektremnog programiranja sprovoede se četiri vrste aktivnosti pomoću kojih se te vrednosti implementiraju u ponašanje I rad tima: ***kodiranje, slušanje, testiranje I projektovanje.***

Postoji 6 osnovnih uloga u XP-u: ***menadžer (****formira tim I upralja njime),* ***trener*** *(uči članove tima o XP metodologiji),* ***tragač*** *(prikuplja korisničke zahteve I prati napredak testova),* ***programmer (****piše testove, projektuje I kodira) ,* ***tester*** *(pomaže kupcu da piše I razvija testove) I* ***kupac (****piše zahteve I testove, te bira zahteve koji će se raditi u određenoj iteraciji).*

1. **“Agile manifesto”**

* *Manifest* – proglas u kom su date osnovni principi na kojima se zasniva agilno programiranje, a oko kojih su se složili mnogi vodeći kreatori I inženjeri ovih metoda. Dotični manifest glasi:
  + *“ Mi otkrivamo bolje načine razvijanja softvera kroz praksu I pomažući drugima da ih primenjuju (rade) I kroz ovaj rad došli smo do nekoliko zaključaka*:
    - *Pojedinci I interakcija su bitniji od procesa I alata*
    - *Softver koji radi je bitniji od obimne dokumentacije*
    - *Saradnja sa klijentima je bitnija od pregovora o ugovorima*
    - *Odgovaranje na promene je bitnije od praćenja plana.*

*I uprkos tome što postoji vrednost u stvarima koje su desno date u ovim poređenjima, mi više vrednujemo stvari navedene na levoj strani.”*

* Gornja definicija principa agilnog programiranja je dovela do definicije sledećih zahteva koji su njime ispunjeni:
  + - *Zadovoljstvo korisnika brzom isporukom korisnog softvera*
    - *Mogućnost promene zahteva, čak I upoodmakloj fazi razvoja*
    - *Česta isporuka softvera, u razmaku od par nedelja*
    - *Ispravan softver je osnovna mera napretka*
    - *Razvoj koj je u stanju da održi konstantan tempo*
    - *Bliska saradnja između projektanata I poslovnih saradnika*
    - *Najbolji tip komunikacije je komunikacija “licem-u-lice”*
    - *Projekti se izvode u okruženju u kojem su motivisani pojedinci, u koje se može imati poverenje*
    - *Kontinualno usmeravanje pažnje ka tehničkoj veštini I dobrom dizajnu*
    - *Jednostavnost*
    - *Samoorganizovani timovi*
    - *Prilagođavanje promenljivim okolnostima*

1. **SCRUM**

* SCRUM metoda je generalna agilna metoda upravljanja koja se fokusira na upravljanje iterativnim razvojem više nego na specifični tehnički pristup agilnom softverskom inženjerstvu. Propisuje načine upravljanja zahtevima, formiranja iteracija, kontrole implementacije I isporuke klijentu. Postoje 3 faze u SCRUM-u:

1. *Prva faza* je okvirna faza planiranja gde se ustanovljuju generalni ciljevi za projekat I dizajn arhitekture softvera
2. *Druga faza* projekta obuhvata seriju sprintova, gde se nakon svakog sprinta dobija inkrement Sistema.
3. *Treća faza – finalna*  je kraj projekta koji upakuje ceo projekat, dovršava potrebnu dokumentaciju I ocenjuje stvari naučene iz projekta.

Suština SCRUM procesa jeste njegova centralna faza – sprintovi. SCRUM sprint je faza u razvoju u kojoj se sav rad koji treba da bude obavljen ocenjuje, odabrana su svojstva za razvoj I implementiran je softver. Na kraju svakog sprinta komplenta funkcionalnost je dostavljena zainteresovanim stranama. Ključne karakteristike ovog procesa su:

1. Sprintovi su fiksne dužine, obično 2-4 nedelje.
2. Početna tačka planiranja su “zalihe proizvoda” tj. Lista poslova koji treba da se obave na projektu. Tokom faze procene u sprintu, ove zalihe se pregledaju I prioriteti I rizici im se dodeljuju. Korisnik je dosta uključen u ovaj process I može da zada nove zahteve ili zadatke na početku svakog sprinta.
3. Faza selekcije obuhvata selektovanje poslova koji će biti obaljveni od strane celog tima, tj. Svakog člana pojedinačno tokom sprinta
4. Kada se dogovore, tim se organizuje za razvoj softvera I sprintovi počinju. Tokom ove faze nema kontakta sa korisnikom niti sa bilo kojim subjektima spolja, sem preko člana tima koji se zove “Scrum master”, koji ima ulogu da zaštiti tim od mešanja sa strane tokom sprintova I lično interaguje sa članovima tima kao posrednik.
5. Na kraju svakog sprinta ceo rad se pregledava I prezentuje zainteresovanim stranama. Sledeći sprint počinje nakon toga.

* Ilustracija koja prikazuje SCRUM process data je ispod:

1. **Lean development (LD)**

* Metodologija inspirisana sistemom proizvodnje automobila u japanskim fabrikama. Npr.vreme razvoja u Tojoti je duplo kraće, a produktivnost radnika je 4 puta veća, uz znatno manji procenat grešaaka nego u fabrikama u SAD-u. LD je skoncentrisan na vrh preduzeća I strategiju upravljanja, a ne na detalje razvojnog procesa na niskom nivou.

Detalji razvoja nisu precizno definisani, kao ni veličina tima I način njegovog organizovanja. Osnovni principi LD-a su:

* + - *Najveći prioritet je zadovoljiti očekivanja korisnika*
    - *Potrebno je obezbediti najbolje moguće rešenje za novac korisnika*
    - *Uspeh zavisi od aktivnog učešća korisnika*
    - *Svaki LD projekat je rezultat timskog rada*
    - *Sve se može menjati*
    - *Rešenje primenljivo na širi problemski domen, a ne samo za konkretan problem*
    - *Bolje je prilagoditi gotovo rešenje, nego raditi svaki put Iz početka*
    - *Bolje je imati 80% rešenja danas, nego 100% rešenja u budućnosti*
    - *Minimalizam je esencijalan za uspeh projekta*
    - *Potreba određuje tehnologiju, a ne obrnuto*

1. **Šta je softverski proces I koje su aktivnosti zajedničke za različite procese. Opisati 4 generička modela softverskih procesa**

**Softverski proces**  je struktuiran skup aktivnosti neophodan za razvoj softverskog Sistema. Zajedničke aktivnosti za sve softverske procese su:

* ***Specifikacija* –** definisanje šta sistem treba da radi
* ***Projektovanje I implementacija*** – definisanje organizacije Sistema I njegova implementacija
* ***Validacija*** – provera da li sistem radi ono što naručilac želi
* ***Evolucija*** – promena Sistema kao odgovor na promenu potreba naručioca

Četiri osnovna modela softverskih procesa su:

1. **Model vodopada –** planom vođen model. Potpuno su odvojene faze specifikacije I razvoja
2. **Inkrementalni razvoj –** specifikacija, razvoj I validacija se poklapaju. Može biti planom vođen ili agilan.
3. **Razvoj zasnovan na korišćenju gotovih komponenti –** sistem se sklapa od govovih komponenti. Može biti planom vođen ili agilan.
4. *U praksi se veliki sistemi najčešće razvijaju procesima koji inkorporiraju elemente sva tri prethodna modela.*
5. **Navesti I kratko opisati atribute dobrog softvera**

* **Pogodnost za održavanje (Maintainability)** – softver treba da bude napravljen tako da se može lako održavati I menjati
* **Stabilnost (Dependability)** – softver mora da uliva poverenje, što podrazumeva da je: *pouzdan, bezbedan I siguran.*
* **Efikasnost (Efficiency) –** mora da ekonomično koristi resurse Sistema
* **Upotrebljivost (Usability) –** mora da bude pogodan za korišćenje

1. **Kategorije zahteva**

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

1. **Tipovi zahteva**

* **Nekvantifikovani –** onaj za koji je testiranjem moguće utvrditi da li je realizovan
* **Kvanitifikovan (na mernoj skali)** – zahevi definisani na neki način koji se može izmeriti – uporediti sa nečim za šta postoji neka merna skala

***Funkcije*** *se ne mogu kvantifikovati, one jesu ili nisu prisutne. Sve* ***osobine I cena*** *se mogu I moraju kvantifikovati.* ***Ograničenja*** *mogu biti bilo kog tipa.*

1. **Najpoznatije RE (request engineering) metode**

* Procesi inženjeringa zahteva (RE) su obično “vođeni” odgovarajućom metodom. Metode **inženjeringa zahteva** definišu načine za dobijanje modela Sistema. Najpoznatije RE metode su:
  + - **DFD (data-flow-diagram)**
    - **Metode koje koriste relacioni model podataka**
    - **Objektno-orijentisane metode**
    - **Formalne metode**
    - **Metode zasnovane na ponašanju Sistema**
      * *Use-case specifikacija*
      * *Viewpoint orijentisane metode*
  + *Kratki opisi ovih metoda biće dati u nastavku:*

1. ***DFD*** *–* metoda zasnovana na prikazu zahteva I projektovanja Sistema kroz dijagram toka podataka. (I sl.)
2. ***Metode koje koriste realacioni model*** *–* za opis zahteva se koristi prošireni ER model
3. ***Formalne metode –*** bazirane sun a matematičkoj formalnoj sintaksi I semantici. Poznate metode: Z, B, VDM, LOTOS. Obezbeđuju načine za postizanje visoke sigurnosti da će sistem odgovarati specifikaciji zahteva. Ne garantuju korektnost Sistema. Nisu veoma popularne zbog teškog razumevanja I notacije, kao I zbog toga što učinak nije očigledan.
4. ***Objektno-orijentisane metode –*** nasledile su ER model. Bazirane su na mnoštvu modela nastalim tokom godina, a čija je notacija različita, a semantika ista. Osnovni koncepti su: **enkapsulacija zahteva, klase, nasleđivanje, atributi I servisi, poruke.**
5. **Faze u RUP-u I odnos iteracija I faza, kao I rezultati svake od faza**

* Postoji 4 faza u RUP-u (rational unified process):

1. **Početna faza (Inception) –** obuhvata analizu problema, razumevnaje potreba potencijalnih korisnika, generalno definisanje Sistema I upravljanje kod promena korisničkih zahteva. Rezultat ove faze je dokument ***Vizija Sistema****.*
2. **Faza razrade (Elaboration) –** obuhvata izradu plana rpojekta, organizaciju I ekipni rad, detaljnu definiciju zahteva I definisanje arhitekture Sistema. Rezultati ove faze su: ***Plan projekta, Use-case specifikacija I Arhitekturni projekat Sistema.***
3. **Faza izrade (Construction) –** obuhvata realizaciju Sistema I testiranje. Kao rezultat se dobijaju: ***Plan testiranja, Test specifikacija, Detaljni projekat Sistema I Softverski proizvod.***
4. **Faza isporuke (Transition) –** obuhvata finalizaciju softverskog Sistema, alfa (beta) testiranje, izradu korisničke dokumentacije, obuku korisnika I uvođenje Sistema kod korisnika. Kao rezultat ove faze dobijaju se: ***Test izveštaji, Korisničko uputstvo I Instalacija Sistema.***

**s**

* **Iteracije u RUP-u mogu biti u okviru faze I međufazne iteracije.** Kod *onih u okviru faze* – svaka faza je iterativna sa rezultatima koji se razvijaju inkrementalno, a kod *međufazne iteracije moguće je čitav skup faza ponavljati.*

1. **Grafički ilustrovati i opisati proces nabavke sistema**

Izbor   
dobavljača

Postoje gotovi sistemi

Adaptiranje  
 zahteva

Potpisivanje ugovora

Ugovaranje

Selekcija  
tendera

Objavljivanje tendera  
 zahteva

Traženje  
ponuda

Izbor sistema

Adaptiranje  
 zahteva

Potrebno je razviti sistem po meri

* Traženje Sistema koji zadovoljava potrebe organizacije se podrazumeva pod *Nabavkom Sistema.* Pre nabavke je neophodno izvesti u nekom obliku specijalno I arhitekturno projektovanje Sistema, što znači da vam je: *Potrebna specifikacija da bi ste ugovorili razvoj Sistema I Specifikacija koja vam može omogućiti kupovinu komercijalnih COTS (commercial off-the-shelf) Sistema. Gotovo uvek je to jeftinije nego razvijati sistem.*
* Problemi koji se javljaju prilikom nabakve Sistema su:
  + Možda je potrebno modifikovati zahteve da bi se prilagodili mogućnostima COTS kompoenenti.
  + Specifikacija zahteva može biti deo ugovora za razvoj Sistema.
  + Obično posotji period rada na ugovoru da bi se naručilac I izvođač usaglasili kakav sistem treba razvijati.

1. **Koraci pri modelovanju Sistema I prikupljanju zahteva primenom objektno-orijentisanih metoda**

* Većina OO metoda ima sledeće zajedničke korake u modeliranju Sistema I prkiupljanju zahteva:

1. ***Identifikacija osnovnih objekata (I klasa)***
2. ***Definisanje objektne strukture I veza između klasa***
3. ***Definisanje atributa I servisa objekata*** ( atributi se dobijaju analizom zahteva; a pod servisima se obično podrazumevaju servisi za pristup I izmenu atributa *(get I set metode),* a jedan od načina da se identifikuju servisi je praćenje poruka koje se razmenjuju između objekata)
4. ***Definisanje poruka koje objekti razmenjuju***
5. **Navesti I objasniti 6 osnovnih principa za uspešan razvoj SW-a po RUP-u**

* ***Iterativni razvoj*** *(svaka iteracija ima za rezultat izvršnu verziju softvera)*
* ***Upravljanje zahtevima*** *(opisano ranije)*
* ***Komponentna arhitektura softvera*** *(zadovoljava I trenutne I buduće zahteve, poboljšava proširljivost, obezbeđuje višestruko korišćenje I enkapsulira zavisnosti)*
* ***Vizuelno modeliranje*** *(koristi se UML)*
* ***Kontinualna verifikacija kvaliteta***
* ***Upravljanje promenama*** *(obuhvata izveštavanje o status proizvoda, upravljanje konfiguracijom, praćenje promena I odlaganje izvornog koda I kontrola verzija)*

1. **Grafički ilustrovati I opisati “4+1” model Sistema**

* „4+1“ model sistema obuhvata 4 odvojene komponente i jedan sveobuhvatni korisnički pogled na njih. Ilustracijom je dat ispod:

Implementacioni  
 model

Logička   
arhitektura

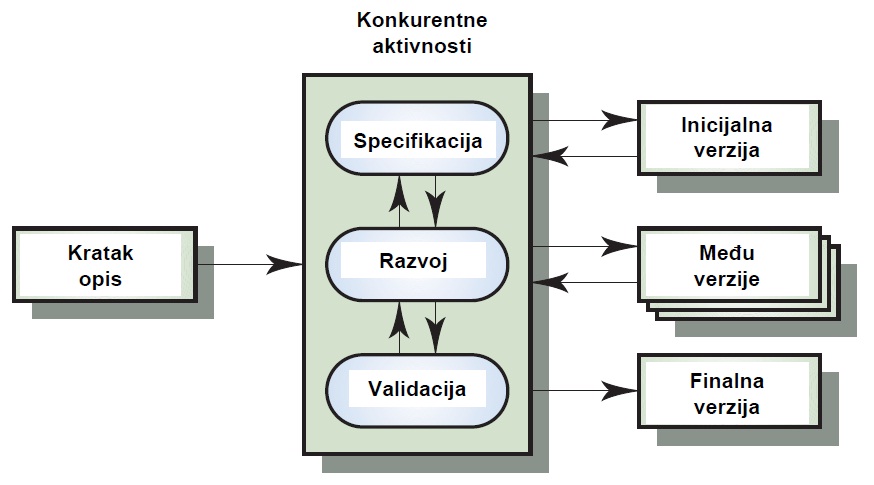
Fizički   
model

Procesna   
arhitektura

* **Logička arhitektura Sistema –** opisuje najvažnije klase u sistemu, njihovu organizaciju u pakete I podsisteme kao I organizaciju paketa I podsistema u nivoe. Za predstavljanje logičke arhitekture koriste se **dijagrami klasa,** kod kojih postoji mogućnost automatskog generisanja koda na osnovu dijagrama klasa.
* **Procesna arhitektura Sistema –** opisuje najvažnije procese I niti (threads) u sistemu I njihovu organizaciju. Procesi se izvršavaju u nezavisnim adresnim prostorima računara, dok su niti procesi koji se izvršavaju paralelno sa procesima ili drugim nitima ali u adresnom prostoru nekog od procesa. Za procesne arhitekture se koriste **dijagrami klasa.**
* **Implementacioni model –** Za prikaz ovog modela koriste se **dijagrami komponenti,** I on prikazuje kako su realne komponente iz Sistema implementirane I u kakvom međusobnom odnosu.
* **Fizički model –** opisuje fizičke čvorove u sistemu I njihov razmeštaj u prostoru. Za prikaz fizičkog modela se koriste **dijagrami razmeštaja.**

1. **Evolutivni model**

* **Evolutivni razvoj** - Zasniva se na ideji da se razvije inicijalna implementacija, da se ona pruži na uvid korisniku i da se na osnovu komentara korisnika vrši rafinacija i da se kroz više verzija dođe do konačnog rešenja. Postoje dva osnovna tipa evolutivnog razvoja:
  + ***Istraživački razvoj* (Exploratory development)** - Cilj je raditi sa klijentima i evolucijom doći do finalnog sistema polazeći od kratke inicijalne specifikacije. Startuje se sa dobro shvaćenim zahtevima.
  + ***Prototipovanje sa odbacivanjem prototipova*** (Throw-away prototyping) - Evolutivni razvoj služi da se dobro shvate zahtevi korisnika i da se potom razvije bolja definicija sistemskih zahteva. Startuje se sa slabo shvaćenim zahtevima
    - *Problem*i: – *Proces nije vidljiv, Sistemi su obično slabo struktuirani, m*ogu biti potrebne specijalne veštine (npr. Jezici za brzo prototipovanje)
    - Aplikativnost: za male i srednje (do 500 000 linija koda) interaktivne sistemem za delove velikih sistema (npr. za korisnički interfejs) , za sisteme sa kratkim životnim vekom



1. **Prikazati spiralni model specifikacije zahteva I objasniti ga, kao I upravljanje zahtevima**

* Svaki od funkcionalnih i nefunkcionalnih zahteva evoluira kroz različite fokuse i perspektive, definišući različite nivoe detalja u zahtevima. Svaka nenamerna praznina u zahtevima povećava *rizik* kraha projekta ili dodavanja drugog defekta.
* Upravljanje zahtevima je više od dokumentovanja istih. Šest preporuka “dobre prakse” za upravljanje zahtevima:

1. Odvojite dovoljno vremena za proces definisanja zahteva.
2. Izaberite pravi pristup 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 sastoji se od međusobno nerazdvojivih podprocesa, koji se ne mogu zasebno i sekvencijalno odvijati:
  + *Prikupljanje zahteva*
  + *Analiza zahteva*
  + *Specifikacija zahteva*
  + *Validacija zahteva*



1. **Objasniti osnovne elemente Use-case dijagrama I osnovne tipove veza kod ovih dijagrama**

* Dijagrami slučajeva korišćenja prikazuju skup slučajeva korišćenja, aktere i relacije između njih. Oni modeluju kontekst sistema i zahteve sistema i služe da specificiraju neku funkcionalnost i ponašanje aplikativnog sistema. Ukratko, oni nam govore šta sistem radi. Preko njih možemo da prikažemo ponašanje sistema, podsistema, klasa ili interfejsa.
* Elementi ovih dijagrama su:

1. ***Slučajevi korišćenja*** – opisuju se tokom događaja i to se opisuje kada slučaj korišćenja počinje i kada se završava, kada slučaj korišćenja interaguje sa akterima i kada se razmenjuju objekti. Tok događaja može biti primarni (osnovni) i alternativni i opisuje se na neki od sledećih načina: neformalan struktuiran tekst, formalan struktuiran tekst (sa pred i post uslovima), pseudokod ili dijagramima interakcije (jedan dijagram za primarni i dodatni za alternativne tokove). Obično se predstavlja kao tabela koja sadrži ime, aktere, trigere, preduslove, postuslove, ispravan scenario i alterantivni tok.
2. ***Akteri*** – skup uloga koji može biti korisnik ili neki sistem sa kojim se interaguje. Sistem može da interaguje sa jednim ili više spoljašnjih aktera. Akter predstavlja standardni stereotip klase koji se predstavlja posebnim grafičkim simbolom u vidu čiča gliše.
3. ***Relacije*** – predstavljaju povezanost aktera i slučajeva korišćenja.
   1. ***Asocijacije – relacija komunikacije; puna linija od aktera do slučaja*** korišćenja; komunikaciju iniciraju i jedna i druga strana.
   2. ***Zavisnosti*** – konkretno se koriste dva stereotipa ove relacije: uključivanje (<<include>>) i proširivanje (<<extend>>). Predstavlja se isprekidanom linijom sa strelicom. Uključivanje od jednog ka drugom slučaju korišćenja (od A ka B) govori o tome da će ponašanje A da uključi i ponašanje B (ne važi obrnuto). Uključivanjem se opisuje zajedničko ponašanje između dva ili više slučaja korišćenja. Proširivanje od A ka B znači da B može da obuhvati i ponašanje iz A (može da se proširi). Proširivanje se događa u tačno određenim tačkama koje se nazivaju tačke ekstenzije.
   3. ***Generalizacije*** – nasleđivanje; puna linija sa trougaonom strelicom. Generalizacija od A ka B kaže da je A specifičan slučaj korišćenja u odnosu na B.
4. ***Paketi*** – moguće je grupisanje aktera, slučajeva korišćenja i njihovih veza u pakete i kasnije pozivanje na te pakete u nekom dijagramu.
5. **Objasniti softversko inženjerstvo zasnovano na komponentama**

* Bazira se na sistematskom ponovom korišćenju komponenti ili gotovih sistema, pa se novi sistem razvija tako što se integrišu postojeće komponente ili postojeći sistemi – *COTS (Commercial-off-the-shelf) systems*

