SWE – Pitanja i odgovori koji se često padaju na usmenom [2021-2017]

1.1. Navesti i kratko opisati atribute dobrog softvera. (7)

- Pogodnost za održavanje (Maintainability)
  - Treba da je u stanju da se lako menja
- Stabilnost (Dependability)
  - Mora da uliva poverenje što podrazumeva da je:
    - Pouzdan (Reliability),
    - Bezbedan (Security) i
    - Siguran (Safety)
- Efikasnost (Efficiency)
  - Mora da ekonomično koristi resusrse sistema
- Upotrebljivost (Usability)
  - Mora da bude pogodan za korišćenje

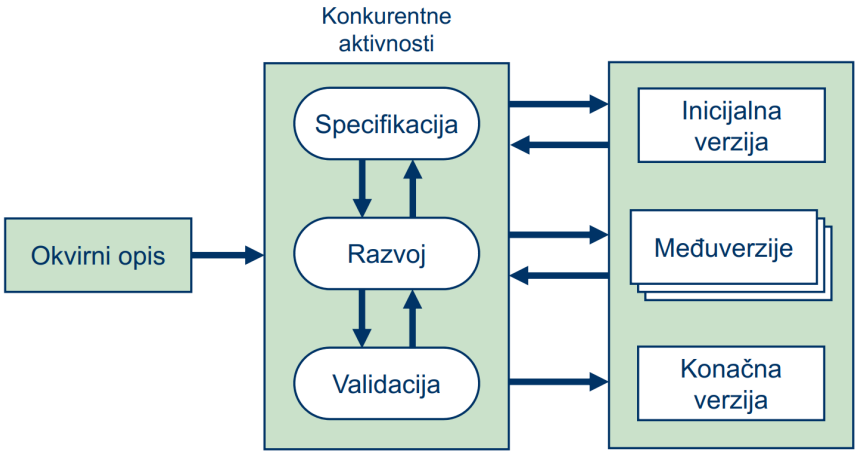
1.2. Po čemu se razlikuje softversko inženjerstvo od informatike? (4)

- Nauka o računarstvu (informatika) se bavi teorijom i osnovama računarstva.
- Softversko inženjerstvo se bavi praktičnom stranom razvoja i isporuke korisnog softvera.
- Teorije nauke o računarstvu je trenutno dobro razvijena i obezbeđuje solidnu osnovu za softversko inženjerstvo.

1.3. Šta je softver i čime se bavi softversko inženjerstvo? (1)

- Softver je računarski program, pridružena dokumentacija i konfiguracioni podaci neophodni da bi softver radio korektno.
- Softverski sistem se obično sastoji od:
  - Određenog broja programa
  - Konfiguracionih fajlova
  - Sistemske dokumentacije koja opisuje strukturu sistema
  - Korisničke dokuemntacije
  - Web site-ova za podršku korisnicima
- Softversko inženjerstvo se bavi razvojem softverskih proizvoda (softver koji se može prodati kupcu)
- Softversko inženjerstvo je inženjerska disciplina koja obuhvata sve aspekte proizvodnje softvera.
- Softverski inženjeri treba da prihvate sistemski organizovan način u svom radu i da koriste odgovarajuće alate i tehnike zavisno od problema koji rešavaju, ograničenja koja nailza u toku razvoja i raspoloživih resursa.

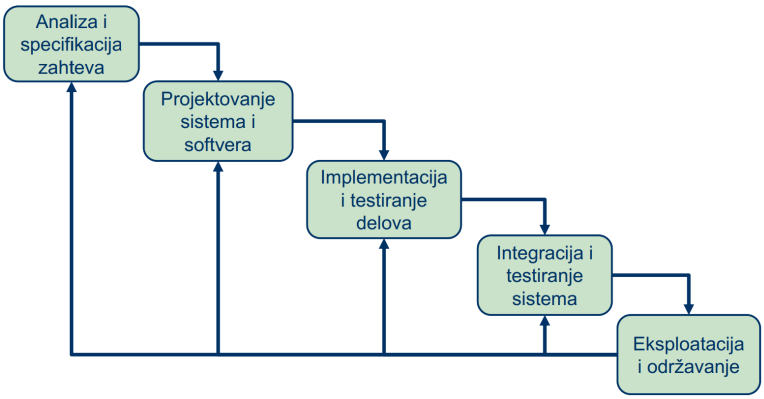
2.1. Grafički ilustrovati i opisati inkrementalni razvoj. Navesti osnovne prednosti i nedostatke. (7)



Inkrementalni razvoj softvera podrazumeva razvoj koji kao ulazne zahteve ima samo okvirni opis krajnjeg proizvoda. Na osnovu njega se onda kreira neka inicijalna verzija softvera nakon čega se kružno vrši specifikacija, razvoj i validacija međuverzija softvera u inkrementima tako da je svaka sledeća međuverzija kompletnija, bliža finalnom proizvodu i u većoj meri zadovoljava realne potrebe zahtevaoca. Sa inkrementalnim razvojem se ciklično nastavlja sve dok se ne dođe do verzije softvera koja se smatra kompletnom i može se finalno isporučiti. Prednosti ovakvog pristupa su što naručilac relativno brzo nakon početka razvoja dobija funkcionalnu verziju softvera koju može da kopristi dok ne stigne finalna verzija, takođe činjenica da naručilac može da dobije svu od međuverzija softvera na korišćenje omogućava veći stepen njegove uključenosti u sam razvoj i time omogućava kreiranje softvera koji je više “po njegovoj meri” jer može da definiše nove, ili izmeni postojeće zahteve, zavisno od iskustva stečenog na interakciji sa nekom od međuverzija. Nedostaci su to što je proces razvoja dinamičan i promenljiv i samim tim nije praćen temeljnom dokumentacijom što čini proces razvoja manje vidljivim menadžerima i drugim osobama na upravljačkim pozicijama, takođe kako se struktura sistema relativno brzo definiše, a često se ne ulaže dovoljno novca u njeno refaktorisanje i prepravku tokom samog rtazvoja to dovodi do toga da se struktura sistema svakim sledećim inkrementom usložnjava što je čini komplikovanom, neefikasnom i samim tim skupom za održavanje.

- Prednosti inkerementalnog razvoja:
  - Smanjena cena izmene korisničkih zahteva u toku razvoja
    - Količina analiza i dokuemntacije koju treba ponovo uraditi je mnogo manja u odnosu na model vodopada.
  - Olakšano dobijanje povratne informacije od korisnika u toku razvoja
    - Korisnici imaju uvid i mogućnosti komentarisanja trenutno implementiranih funkcionalnosti, a samim tim i u proges realizacije projekta
  - Moguća brža isporuka korisnog softvera naručiocu
    - Korisnici mogu da koriste softver pre nego u slučaju modela vodopada.
- Nedostaci inkrementalnog razvoja:
  - Proces nije vidljiv
    - Menadžerima su potrebne redovne isporuke kako bi merili napredovanje. Ako se sistem brzo razvija nije efikasno praviti dokuemntaciju za svaku verziju sistema.
  - Struktura sistema iam tendenciju degradacije sa dodavanjem novih inkremenata
    - Ukoliko se ne ulaže u refaktorisanje i unapređenje softvera, periodićne izmene imaju za posledicu degradaciju strukture. Nove i nove izmene softvera postaju sve složenije i skuplje.

2.2. Razvoj softvera po modelu vodopada (dijagram, prednosti i problemi). (4)



Nalaže razvoj softvera koji prolazi kroz faze strogo sekvencijalno, bez povratnih sprega i redosled tih faza tj. Aktivnosti u razvoju softvera je strogo definisan. Kako bi se prošlo u sledeću fazu razvoja, potrebno je da se prethodna faza završi i da svi njeni izlazni rezultati budu dostupni. Ovaj princip čini ovaj model jako nefleksibilnim i nepogodnim u slučajima kada je potrebno da postoji mogućnost izmene postojećih ili dodavanja novih zahteva u procesu razvoja, jer faza analize i specifikacije predstavlja prvu fazu razvoja i zahteva se da svi njeni izlazni rezultati, koji uključuju čitav spisak definisanih zahteva, budu potpuno definisani da bi uopšte moglo da se pređe na sledeću fazu razvoja. Ovaj model smatra se delimično zastarelim i kosti se samo u slučajevima kada se radi na razvoju izuzetno velikog sistema gde se ravoj odvija paralelno na više različitih lokacija pa je ovakav strogo planom vođen razvoj poželjan.

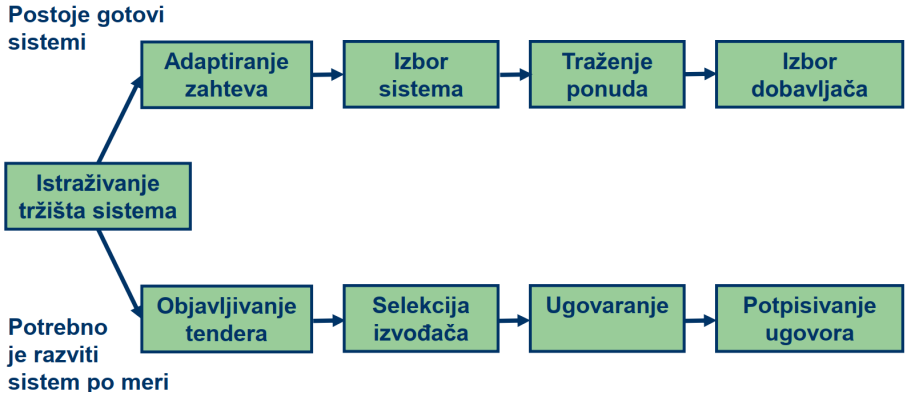
- Faze u modelu vodopada:
  - Analiza i specifikacija zahteva
  - Projektovanje sistema i softvera
  - Implementacija i testiranje delova
  - Integracija i testiranje sistema
  - Eksploatacija i održavanje
- Prednosti:
  - Dobro definisani i usaglašeni zahtevi na početku projekta sa malom verovatnoćom izmene u toku razvoja.
  - Koordinisanje posla kada se projektuje veliki sistem gde je razvoj razdeljen na nekoliko lokacija
  - Pogodan pri paralelnom razvijanju različitih delova sistema
- Problemi:
  - Nefleksibilna deoba projekta na disjunktivne faze otežava odgovor na promenu korisničkih zahteva
  - Nemogućnost efikasnog prihvatanja izmena u zahtevima korisnika kada je proces u toku. U principu, jedna faza mora biti završena da bi se otpočela naredna, pa izmene zahteva resetuju čitav proces.
  - Analiza i dokumentacija su mnogo veće u odnosu na inkrementalni model

2.3. Šta je softverski proces? Navesti i kratko opisati zajedničke aktivnosti za različite softverske procese. (2)

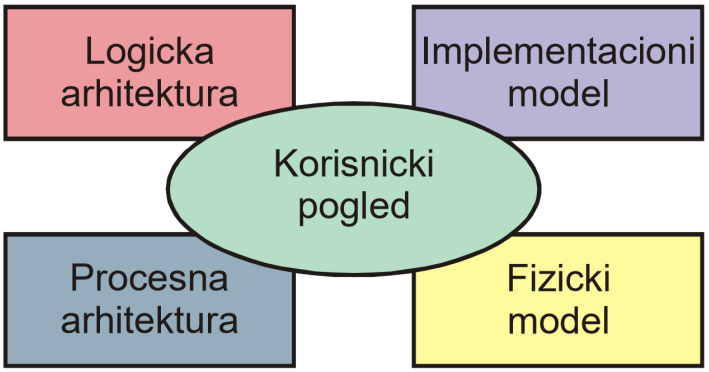
- Softverski proces je skup aktivnosti i pridruženih rezultata čiji je cilj proizvodnja softvera
- Aktivnosti zajedničke za sve softverske procese su:
  - Specifikacija softvera (Software Specification)
    - Definisanje šta sistem treba da radi
  - Razvoj softvera (Software Development)/Projektovanje i implementacija
    - Definisanje organizacije sistema i njegova implementacija
  - Validacija softvera (Software Validation)
    - Provera da li sistem radi ono što naručioc želi
  - Evolucija softvera (Software Evolution)
    - Promena sistema kao odgovor na promenu potreba naručioca
- Model softverskog procesa predstavlja apstraktnu reprezentaciju procesa. Reč je o opisu procesa iz određene perspektive.

2.4. Grafički ilustrovati i objasniti proces nabavke sistema. (1)

- Opis:
  - Traženje sistema koji zadovoljava potrebe organizacije
  - Pre nabavke je neophodno izvesti u nekom obliku specijalno i arhitekturno projektovanje sistema
    - Potrbna je specifikacija da bi se ugovorio razvoj sistema
    - Specifikacija može omogućiti kupovinu komercijalnih COTS (Commercial Off The Shelf) sistema. Gotovo uvek je jeftnije nego razviti sistem.
- Problemi:
  - Možda je potrebno modifikovati zahteve da bi se prilagodili mogućnostima COTS komponenata
  - Specifikacija zahteva može biti deo ugovora za razvoj sistema
  - Obično postoji period rada na ugovoru da bi se naručilac i izvođač usaglasili kakav sistem razvijati.



3.1. Grafički ilustrovati i kratko opisati “4+1” model sistema. (7)



- Logička arhitektura sistema:
  - Opisuje najvažnije klase u sistemu; njihovu organizaciju u pakete i podsisteme kao i organizaciju paketa i podsistema u nivoe (layers)
  - Za predstavljanje logičke arhitekture se koriste dijagrami klasa
  - 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 nititima, ali u adresnom prostoru nekog od procesa.
  - Za prikaz procesne arhitekture sistema koriste se dijagrami klasa
- Implementacioni model sistema:
  - Za prikaz implementacionog modela koriste se dijagrami komponenti
  - Predstavlja fizičke komponente u smislu posebnih fajlova koji čine sistem (dll, exe, config).
  - Treba navesti i veze između komponenata.

- Fizički model sistema:
  - Opisuje fizičke čvorove u sistemu i njihov razmeštaj u prostoru
  - Za prikaz fizičkog modela se korste dijagrami razmeštaja
  - Kockama se predstavljaju procesni elementi, a linijama se navode veze određenog tipa (LAV, interget).
  - Kao fizički čvorovi u ovom dijagramu javljaju se svi računari sistema, ali i ostale fizičke komponente. Za manje sisteme, moguće je kombinovati fizički i implementacioni model, tako što se u kocku procesnog elementa fizičkog modela ubacuje dijagram komponenata koje se fizički nalaze na računaru na koji se ta kocka odnosi.
- Korisnički pogled
  - Ne utiče na stvari kao što su projektovanje klasa, modula i tome slično
  - Za ovaj pogled vezano je definisanje Use Case-ova sistema, odnosno sastoji se iz kompletnog opisa toga što će sistem korektno da radi i na koji način (šta radi korsinik i koji odgovor daje sistem na njegove akcije)
  - Ovaj pogled predstavlja dobru specifikaciju za projektovanje.

3.2. Navesti faze RUP metodologije i objasniti šta se dobija kao rezultat svake od faza. (5)

- RUP (Rational Unified Process) metodologija se sastoji od faze:
  - Započinjanje (Inception)
    - Analiza poslovne perspektive
    - Rezultat ove faze je dokument Vizija sistema
  - Razrada (Elaboration)
    - Razumevanje domena problema i razvoj arhitekture sistema
    - Rezultat ove faze su:
      - Plan projekta
      - Use-case specifikacija
      - Arhitekturni projekat sistema
  - Izrada (Construction)
    - Projektovanje sistema, implementacija i testiranje
    - Rezultati ove faze su:
      - Plan testiranja
      - Test specifikacija
      - Detaljni projekat sistema
  - Okončanje (Transition)
    - Isporuka i instalacija sistema
    - Rezultati ove faze su:
      - Test izveštaji
      - Korisničko uputstvo

3.3. Navesti i kratko opisti faze u RUP-u. Objasniti odnos faza i iteracija u RUP-u. (1)

- Početna faza (Inception)
  - Analiza problema
  - Razumevanje potreba (potencijalnih) korisnika

    - Generalno definisanje sistema
    - Upravljanje kod promena korisničkih zahteva
- Faza elaboracije (Elaboration)
    - Izrada plana projekta
    - Organizacija i ekipni rad
    - Detaljna definicija zahteva
    - Definisanje arhitekture sistema
    - Rezultat ove faze su:
        - Plan projekta
        - Use-case specifikacija
        - Arhitekturni projekat sistema
- Faza izrade (Construction)
    - Realizacija sistema
    - Testiranje
    - Rezultati ove faze su:
        - Plan testiranja
        - Test specifikacija
        - Detaljni projekat sistema
- Faza isporuke:
    - Rezultati ove faze su:
        - Test izveštaji
        - Korisničko uputstvo
        - Instalacija sistema
- Svaka faza može imati proizvoljan broj iteracija i svaka iteracija (osim početne) treba da rezultira izvršnom verzijom koja se može testirati

4.1. Navesti koji sve sastanci po Scrum-u postoje i kratko ih opisati. (8)

- U SCRUM metodologiji postoje 4 tipa sastanka. Pre početka sprinta, organizuje se planiranje sprinta. Kada sprint počne, svakog dana postoji dnevni sastanak. Nakon završetka sprinta organizuju se 2 tipa sastanka:
    - Pregled sprinta, u kome se posmatra šta je urađeno u sprintu i
    - Retrospektiva sprinta, koja služi za unapređenje procesa
- Planiranje sprinta
    - Definiše se cilj sprinta i na osnovu njega biraju se stavke iz Product backlog-a na osnovu kojih se kreira sprint backlog. Mapiranje stavke iz Product backlog-a na stavku iz sprint backlog-a nije 1:1, jer kada smo već došli do trenutka da nešto moramo da implementiramo, često se desi da nešto što smo smatrali za jednu stavku u Product backlog-u predstavlja više različitih stavki u sprint backlog-u. Kada se identifikuju zadaci sprinta, svaki od njih se procenjuje po tome koliko će vremena da oduzme, ovo je potrebno zbog praćenja progresa sprinta. Ta procena se ne postiže tako što scrum master naloži procenu, već svi učestvuju i daju sopstvene procene pa se na osnovu njih dogovaraju koncezusom o nekoj finalnoj odluci o proceni svakog zadatka.
- Dnevni sastanak
    - Održava se svakog dana tokom trajanja sprinta, to je kratak sastanak koji traje otprilike 15 minuta i uslovno rečeno je “usputan”, njemu mogu da prisustvuju svi, ali najviše razloga za pričanje imaju članovi razvojnog tima, scrum master i vlasnik proizvoda. Na ovom sastanku treba svako da kaže šta je radio juče, šta planira da radi danas i da li ima neke probleme koji mu stoje na putu. Ovo ne predstavlja neki raport scrum masteru, već ima za cilj da se svi upoznaju sa time ko na čemu trenutno radi, šta je već urađeno i sa time da li ima nekih problema. Istovremeno, obzirom na to da se problem tako javno iznese, bilo ko ima ideju ili način da ga

reši, može da se javi i time se olakšava saradnja i pronalaženje adekvatne pomoći. Pomaže da se izbegnu ostali, nepotrebni sastanci.

- Pregled sprinta
  - Prezentuje se ono što je urađeno za vreme sprinta. Sadrži prezentaciju nove verzije proizvoda i funkcionalnosti koje su urađene, ali se ta prezentacija ne priprema mnogo. Postoji takozvano pravilo dvostrane primedbe, dakle kratko se pripremi i prezentuje ono što je urađeno bez neki slajdova ili previše planiranja. U prezentaciji učestvuje ceo tim, a svi su pozvani da prisustvuju po želji.
- Retrospektiva sprinta
  - Predstavlja preiodično razmatranje toga šta je dobro, a šta nije. Posmatra se način na koji se rade određene stvari i to da li može nekako da se unapredi taj način. Ovaj sastanak ne traje dugo, obično 15-30 minuta i obično se organizuje preko svakog sprinta. Ceo tim učestvuje u ovom sastanku, a mogu da se priključe i klijenti i ostali po želji. Cilj je unaprediti proces rada, možda nešto ukinuti, uvesti nešto novo ili izmeniti postojeće.

4.2. Navesti osnovne principe agilnog razvoja softvera izražene kroz “Agile Manifesto”. (4)

- Agilno = aktivnost, hitnost, spremnost za pokret
- Inkrementalne metode (3-6 meseci) vs. Agilne metode (1-4 nedelje)
- 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
  - Principi:
    - Zadovoljstvo korisnika brzom isporukom korisnog softvera
    - Mogućnost promene zahteva, čak i u poodmakloj fazi razvoja
    - Česta isoruka softvera je osnovna mera napretka
    - Razvoj koji 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 poverenja
    - Kontinualno usmeravanje pažnje ka tehničkoj veštini i dobrom dizajnu
    - Jednostavnost
    - Samoorganizvoani timovi
    - Prilagođavanje promenljivim okolnostima

4.3. Objasniti osnovne tipove veza kod Use-case dijagrama i ilustrovati jednim dijagramom. (1)

- Relacija komunikacije (asocijacija) => puna linija
- Relacija zavisnosti => isprekidana linija sa strelicom
  - <<include>> stereotip koji označava da slučaj korišćenja uključuje i ponašanje slučaja korišćenja na koji pokazuje
  - <<extend>> stereotip proširivanja označava da slučaj korišćenja na koji pokazuje veza može obuhvatiti i slučaj korišćanja od koga polazi veza
- Relacija generalizacije => puna linija sa trougaonom strelicom na vrhu
  - Ukazuje da je slučaj korišćenja na koga pokazuje strelica generalizacija slučaja korišćenja od koga polazi strelica (važi i za aktere)

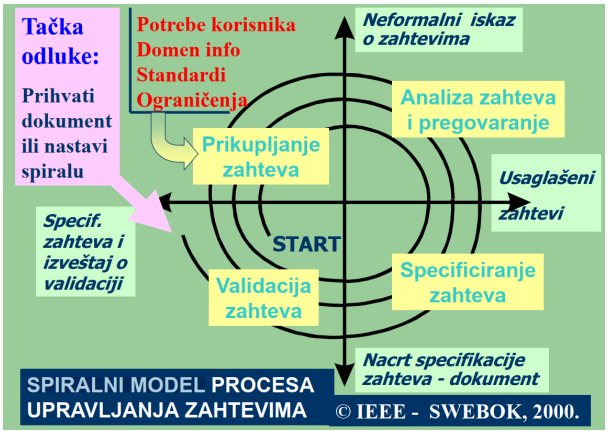
5.1. (5) Use-case metoda i scenariji događaja. Ilustrovati na primeru.

- Sistem se posmatra sa stanovišta korisnika sistema.
- Opisuju se slučajevi korišćenja sistema i scenariji 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 prate do objekata koji reaguju na njih
- Tipičan model scenarija događaja je interakcija između korisnika i sistema.
- Scenariji su primeri kako će ssitem biti korišćen u realnom životu
- Oni treba da uključe:
    - Opis početne situacije
    - Opis normalnog toka događaja
    - Opis izuzetaka (ako se izađe iz normalnog toka)
    - Informacije o konkurentnim aktivnostima
    - Opis stanja gde se scenario završava
- Slučajevi korišćenja (Use cases) predstavljaju tehniku baziranu na scenarijima i zapisanu pomoću UML dijagrama koja identifikuje aktere u sistemu i slučajeve korišćenja sistema
- Potpuni skup slučajeva korišćenja opisuje sve moguće interakcije korisnika sa sistemom
- UML dijagrami sekvenci mogu biti korišćeni za detaljni opis slučaja korišćenja. Oni daju sliku o obradi događaja u sistemu za izabrani slučaj korišćenja.

5.2. Šta predstavlja upravljanje zahtevima? (1)

- Upravljanje zahtevima predstavlja 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. Potrebno je definisati proceduru u slučaju promene zahteva korisnika.

6.1. Grafički ilustrovati i kratko objasniti spiralni model procesa specifikacije zahteva. (4)



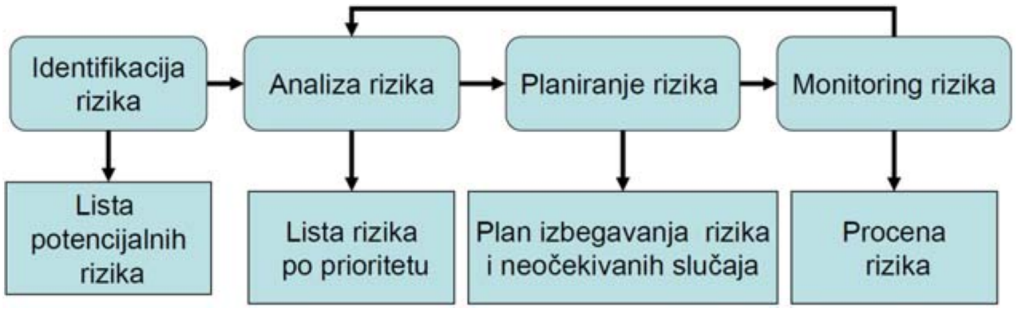
- Spiralni model procesa upravljanja zahtevima podrazumeva kružni prolaz kroz 4 definisane faze upravljanja zahtevima. Svaki put kada se prođe pun krug, potrebno je odlučiti da li smo zadovoljni kreiranim dokumentom specifikacije zahteva ili želimo da prođemo još jedan krug kroz sve 4 faze.
- Faze koje su sastavni deo kruga:
    - Prikupljanje zahteva – Sakupljaju se informacije o potrebama korisnika, informacije o domenu, o standardima i ograničenjima. Nakon ove faze postoji definisan neformalni iskaz o zahtevima.
    - Analiza zahteva i pregovaranje – Vrši se analiza zahteva prikupljenih u prethodnoj fazi i pregovara se sa korisnikom u cilju usaglašenja mišljenja i boljeg razumevanja. Kao rezultat ove faze potrebno je da izvršilac i korisnik imaju usaglašene zahteve.
    - Specificiranje zahteva – Zahtevi usaglašeni u prethodnoj fazi se formiraju. Kao rezultat ove faze imamo nacrt dokumenta specifikacije zahteva.

- Validacija zahteva – Zahtevi formalizovani u prethodnoj fazi se validiraju, tj. proverava se da li odgovaraju očekivanjima korisnika. Kao izlaz iz ove faze imamo specifikaciju zahteva i izveštaj o validaciji.

6.2. Šta predstavlja upravljanje projektima i po čemu su softverski projekti osobeni? (3)

- Pod upravljanjem projektima se podrazumeva organizovanje, planiranje i raspoređivanje softverskih projekata. Odnosi se na aktivnosti koje treba da osiguraju da se softver isporuči na vreme i po planu u skladu sa zahtevima organizacije koja ga kreira i organizacije koja ga nabavlja. Predstavlja bitan aspekt jer je razvoj softvera uvek limitiran budžetom i raspoloživim vremenom za rad. Upravljanje softverskim projektom je specifično i razlikuje se od upravljanja projektima u drugim inženjerskim disciplinama zato što:
  - Proizvod je nevidljiv – Za razliku od drugih inženjerskih disciplina u kojima je lako videti dokel se stiglo sa razvojem (na primer kada gradimo kuću, možemo na osnovu toga koliko smo sagradili da znamo tačno dokle smo stigli). Kod softverskog proizvoda je mnogo teže odrediti dokle se stiglo sa razvojem, postojanje neke funkcionalnsoti ili nekog skupa funkcionalnosti ne podrazumeva nužno da se daleko stiglo, može da postoji neki nefunkcionalni zahtev koji nije ispunjen ili još neki funkcionalni koji zavisi od navedenog skupa ili od kojeg navedeni skup zavisi. Ta veza između implemntacionih delova je znatno kompleksna i nije tako lako odrediti dokle se stiglo kada se posmatra procentualno ispunjenje finalnog cilja. Nekada može da nam deluje da imamo dosta, ali da zapravo zbog neispunjenosti nekih nefunkcionalnih zahteva sistem u globalu nije funkcionalan po standardima koje je trebao da zadovolji i da mi zapravo nemamo ništa.
  - Proizvod je fleksibilan – Možemo menjati proizvod u srži u bilo kom trenutku razvoja, često i kupac koji je zahtevao kreiranje može u toku samog razvoja da menja zahteve i da traži da se oni ispune u bilo kom trenutku.
  - Softversko inženjerstvo je mlada oblast – Disciplina je skoro otkrivena u odnosu na ostale inženjerske discipline, što čini njeno polje jako promenljivim i u stalnom razvoju.
  - Sam proces razvoja softvera nije standardizovan – Postoji mnog različitih načina da se pristupi samom razvoju u zavisnosti od samog proizvoda.
  - Mnogi softverski proizvodi su unikatni – Sam pojam softverskog proizvoda je toliko širok da može da ne postoji apsolutno ništa zajedničko za neka dva softverska proizvoda, što dovodi do toga da i njihovi sami procesi razvoja moraju korenito da se razlikuju jedan od drugog, što nije slučaj kod drugih inženjerskih disciplina.

6.3. Kratko opisati i grafički ilustrovati proces upravljanja rizikom. (1)



- Proces upravljanja rizikom se sastoji od 4 koraka:
  - Identifikacija rizika – Podrazumeva identifikovanje svih rizika iz kategorija rizika projekta, proizvoda i poslovanja. Kao izlaz daje listu potencijalnih rizika. Nema nekih striktnih pravila kako se obavlja ova faza, bitno je da onaj ko radi ima neko

iskustvo i da zna otprilike koji se to problemi često javljaju i da ne bude previše optimističan povodom toka projekta, već realno sagledava šta bi sve moglo da pođe po zlu. Dobra polazna tačka za ovu fazu je da imamo neku unapred definisanu listu mogućih rizika kroz koju možemo da prođemo i da čekiramo koji od tih rizika bi mogli da se dese.
    - Analiza rizika – Vrši se procena verovatnoće i posledice identifikovanih rizika, kao i ozbiljnost svakog od rizika, tj. koliko je on opasan. Kao izlaz daje listu rizika po prioritetu (definisano je koliko je svaki od njih bitan). Verovatnoća u ovoj fazi se računa nekom procenom, tj. daje joj se neka lingvistička vrednost – vrlo nisak, nisak, srednja, visoka, vrlo visoka. Takođe, posledice rizika se označavaju nekim semantičkim nivoom značaja – katastrofalni, ozbiljni, tolerišući, beznačajni itd...
    - Planiranje rizika – Ako se deis neki od rizika šta raditi povodom toga, odnosno kako izbeći ili makra minimizovati uticaj tog identifikovanog rizika na projekat. Kao izlaz daje plan izbegavanja rizika i neočekivanih slučaja. Ne postoji neki prost proces koji će da dozvoli lako planiranje rizika, već u mnogome zavisi od samog menadžera projekta.
    - Monitoring projekta – Nadgledamo rizike u toku samog projekta. To je neka kontinuirana aktivnost u kojoj pokušavamo da uočimo da li se neki od predviđenih rizika dešava. Kao izlaz ima procenu rizika u odgovarajućem trenutku. Ako se uoči neki rizik, onda se posmatraju njegovi efekti u odnosu na očekivane i to kako njegovo pojavljivanje utiče na verovatnoću pojavljivanja tog rizika.

7.1. Navesti i kratko opisati osobine dobro projektovanog softvera. (8)

- Hijerarhija – softver bi trebalo da bude oranizovan u dobru hijerarhiju komponenata. Nije pogodno da postoji mnogo komponenata sa previše veza, koje nije moguće ispratiti.
- Modularnost – sistem treba da bude dekomponovan u posebne celine (module, podsisteme) sa jasno definisanim interfejsima, tako da svaka celina zna šta može da koristi od drugih. Primer dekompozicije može biti podela u posebnu celinu koja sadrži podatke i celinu koja te podatke obrađuje.
- Nezavisnsot – Pri dekompoziciji, treba grupisati slične stvari u isti modul. Na ovaj način, ako se menja nešto bitno,

7.2. Navesti osnovne karakteristike, kao i prednosti i nedostatke event-driven arhitekturnih modela. (2)

- Ovaj model se koristi kod sistema koji su upravljani eksterno generisanim događajima (events)
- Postoje dve osnovne grupe ovih modela:
    - Broadcast modeli
    - Interrupt-driven modeli
- Komponente:
    - Komponente i podsistemi koji generišu ili obrađuju događaje
- Konektori:
    - Broadcast sistem i event procedure
- Prednosti:
    - Podrška višestrukom korišćenju softvera (reuse)
    - Laka evolucija sistema
    - Lako uvođenje nove komponente u sistem (jednostavno se registruje za neki event)
- Nedostaci:

    - Kada komponenta generiše događaj, ona ne može da zna da li će neka komponenta da odgovori na njega i kada će pbrada događaja biti završena

7.3. Navesti i kratko opisati kategorije zahteva. (2)

- Funkcije
    - "Šta" sistem mora da bude sposobvan da uradi
- Osobine
    - "Koliko dobro" će funkcije biti izvršene
- Cena
    - Koliko će koštati (bilo koji ulazni resurs: novac, ljudi ili vreme) kreiranje i održavanje funkcija i njihovih osobina
- Ograničenje
    - Bilo koja restrikcija u slobodi definisanja zahteva ili dizajnu

8.1. Ukratko opisati i navesti prednosti i nedostatke slojevitog (eng. layered) arhitekturnog modela. (7)

- Ovaj odel se koristi kod modeliranja interfejsa među podsistemima
- Sistem se organizuje u skup slojeva (layera) od kojih svaki obezbeđuje jedan skup funkcionalnosti sloju iznad i služi kao klijent sloju ispod.
- Omogućava inkrementalni razvoj podkomponenti u različitim slojevima:
    - Komponente:
        - Slojevi
    - Konektori:
        - Interfejsi
- Prednosti:
    - Promena interfejsa jednog sloja može da utiče na maksimalno još dva sloja
    - Laka zamena jednog sloja drugim, ukoliko su im interfejsi identični
    - Baziran je na visokom nivou apstrakcije
- Nedostaci:
    - Ne mogu svi sistemi da se lako organizuju po svom modelu

8.2. Navesti i kratko opisati kategorije strategija za upravljanje rizikom. (5)

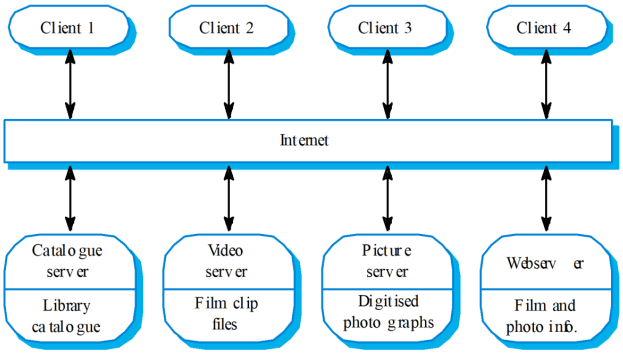
- Strategija izbegavanja
    - Smanjiti verovatnoću pojave rizika
    - Primenjeno kod defekata u komponentama
- Strategije minimizacije
    - Smanjiti uticaj rizika na projekat ili proizvod
    - Primenljeno kod bolesti osoblja
- Planovi za nepredviđene događaje
    - Ako se rizik pojavi, ovi planovi definišu kako se radi sa rizikom
    - Primenjeno kod finansijskih problema organizacije

8.3. Šta predstavlja i koji su ciljevi inspekcije softvera. (3)

- Inspekcija softvera predstavlja statičku verifikaciju softvera i podrazumeva proveru statičkih reprezentacija sistema i potragu za greškama u okviru njih. Statičke reprezentacije sistema su na primer kod i dokuemnti.
- Često se u okviru inspekcije softvera koriste alati za analizu softvera ili za pregeldavanje dokumentacije koja je predmet testiranja.

- Cilj inspekcije softvera je ispitivanje izvorne reprezentacije softvera i traženje anomalija i grešaka u okviru nje.
- Ne podrazumeva pokretanje porgrama tako da se može vršiti pre implementacije.
- Može biti primenjena na bilo koju reprezentaciju softvera (zahteve, projekta, konfiguracione podatke...)
- Inspekcija softvera se pokazala kao vrlo efikasna tehnika u otkrivanju grešaka u softveru.

8.4. Ukratko opisati i navesti prednosti i nedostatke klijent/server arhitekturnog modela. (2)



- Ovaj model se koristi kod distribuiranih sistema
- Sastoji se od skupa stand-alone servera koji obezbeđuju specifične servise (štampa, Web, baza podataka, ...), skupa klijentata koji pozivaju te servise i mreže koja mogućava udaljeni pristup.
- Komponente:
  - Klijenti, serveri
- Konektori:
  - Mreža, servisi servera
- Prednosti:
  - Efikasno korišćenje mrežnih sistema
  - Omogućava korišćenje slabijeg hardvera za klijente, obzirom da server obrađuje većinu posla
  - Lako dodavanje novih servera i unapređenje postojećih
- Nedostaci
  - Neefikasna razmena podataka između klijenata (moraju da idu preko servera)
  - Redudantnost podataka
  - Ne postoji centralni registar imena servera i servisa, pa nije lako otkriti koji serveri i servisi su na raspolaganju

9.1. Šta predstavlja verifikacija, a šta validacija? (6)

- Verifikacija predstavlja pitanje da li na pravi način razvijamo softver, tj. Da li softver koji pravimo ispunjava sve zahteve navedene u specifikaciji.
- Validacija predstavlja pitanje da li pravimo pravi proizvod, tj. Da li je proizvod koji razvijamo saglasan sa onim što od njega očekuje korisnik za kojeg ga razvijamo.
- Ukoliko specifikacija nije dobro napisana može da se desi da proizvod prođe verifikaciju i zadovolji sve uslove iz specifikacije, ali ne prođe validaciju, jer korisnik jednostavno nije zadovoljan i proizvod ne predstavlja ono što je on očekivao, osnosno ono što mu je zapravo potrebno.

9.2. Navesti, kratko opisati i uporediti različite metode testiranja softvera. (5)

- Metode testiranja se svrstavaju u tri gurpe:
  - Metode crne kutije
    - Ne znamo kako je softver iznutra urađen, već mu pristupamo kao bilo koji drugi korisnik, preko korisničkog interfejsa, ili ukoliko je to neka serverska komponenta koju testiramo i ona ima neki API, mi onda testiramo taj API. Ali suština je da test inženjer ne zna šta se odvija unutra.
  - Metode sive kutije

        - Podrazumevaju da nemamo uvid u kod, ali možemo da pratimo neke međuprezentacije. Na primer, imamo neki sistem koji radi sa bazom podataka pa mi samom sistemu pristupamo preko korisničko interfejsa, ili API-ja, a onda vršimo monitoring baze podataka.
    - Metode bele kutije
        - Obrnuta situacija od crne kutije, osoba koja testira ima uvid i kontrolu nad izvornim kodom u toku testiranja. Ovu vrstu testiranja uglavnom ne obavljaju testeri, već developeri prilikom razvoja odgovarajućih komponenti.
- Metode bele i crne kutije su krajnji slučajevi, a metode sive predstavljaju neku mešavinu prethodno pomenuta dva tipa. Kriterijumi za kategorizaciju su to da li se pri razvoju test slučajeva pristupa izvornom kodu softvera koji se testira ili se testiranje vrši preko korisničkog interfejsa ili nekih predviđenih API-ja.

9.3. Opisati i uporediti top-down i bottom-up pristupe projektovanja softvera. (1)

- Top-down projektovanje softvera
    - Polazi se od vrha sistema, odnosno od najviših slojeva i onda se polako dolazi do podsistema koji su na nižim nivoima.
    - Loša strana ove tehnike je ta što forsira razvoj pojedinih grana sistema, dok neke druge nisu ni započete.
    - Takođe, ova tehnika ne sagledava na pravi način već postojeće komponente koje se mogu iskoristiti.
    - Dobra strana ovog pristupa je to što se kreće sa vrha. Na najvišim slojevima je korisnički interfejs, odnosno jedini slojevi sa kojima korisnik direktno dolazi u kontakt. Zbog toga, u ranim fazama mogu da se razreše nedoumice sa korisnikom, jer on rano dobija prototip i inicijalni izgled sistema, na koji može da uloži primedbu. Niže komponente još uvek nisu ubačene u prototip, jer se do njih još uvek nije stiglo u projektovanju. Zbog toga se prave stub-ovi koji predstavljaju zamene za module projektovane tako da imaju isti interfejs kao modul koji menjaju, ali u pozacdini funkcije stub-ova ne rade ono za šta je modul namenjen, već samo pružaju mogućnost poziva.
- Bottom-up projektovanje softvera
    - Polazi se sa dna sistema, odnosno od najnižih slojeva i onda se polako dolazi do podsistema koji su na višim nivoima
    - Obično se kreće od gotovih komponenata, koje se povezuju kako bi se realizovali neki delovi sistema
    - Loša strana ove tehnike je ta što se najviši slojevi sistema, a koje korisnik direktno vidi, dobijaju u kasnim fazama implementacije.
    - Dobra strana je što se mogu na višim nivoima koristiti gotove komponente sa nižih nivoa.
    - Na najnižim nivoima kreiraju se moduli. S obzirom na to da još uvek nemamo projektovane više nivoe koji bi pozivali funkcije nižih modula, kreiraju se takozvani drajveri, koji predstavljaju zamenu za više module i vrše pozive funkcija nižih modula. Drajver nema funkcionalsti pravog modula čiju zamenu vrši. Kako se penjemo, jedan po jedan drajver menja se pravim modulima. Zamena drajvera je bezbolna, jer ne menja ni na koji način niže, već postojeće module.

9.4. Opisati strukturu test slučajeva i ilustrovati na jednom primeru. (1)

- U osnovi ima 3 celine:
    - Uvod
        - Sadrži opšte informacije o test slučaju:
            - Identifikator – jedinstvenu oznaku test slučaja

        - Vlasnik ili kreator test slučaja
        - Verzija definicije test slučaja
        - Naziv test slučaja
        - Identifikator zahteva koj ije pokriven ovim test slučajem
        - Namena – koja se funkcionalnost testira
        - Zavisnosti
    - Aktivnost test slučaja:
        - Okruženje/konfiguracija test slučaja – potreban HW i SW koji mora biti obezbeđen da bi se izvršio test slučaj
        - Inicijalizacija – šta treba da se obezbedi pre izvršenja test slučaja (npr. da se otvori datoteka)
        - Finalizacija – Opis akcije koja treba da se uradi nakon izvršavanja test slučaja (npr. ako test slučaj obori bazu treba je oporaviti pre nego što se test slučaj ponovo izvrši)
        - Akcije – šta treba uraditi korak po korak da bi se test slučak kompletirao
        - Ulazni podaci
    - Očekivani rezultati
        - Opisuje šta će tester videti posle izvođenja svih koraka

10.1. Navesti i kratko opisati tipove održavanja softvera. (10)

- Održavanje programa predstavlja bilo koju izmenu programa nakon puštanja u upotrebu i najčešće ne uključuje velike izmene u arhitekturi sistema.
- Tipovi održavanja softvera:
    - Održavanje u cilju ispravke softverskih grešaka
        - Izmena sistema kako bi se otklonili nedostaci koji sprečavaju da sistem radi u skladu sa svojom specifikacijom.
    - Održavanje sa ciljem prilagođenja softvera za drugačije radno okruženje
        - Promena sistema tako da može da radi u drugačijem okruženju (računar, OS, itd.) od onog za koje je inicijalno implementiran
    - Održavanje u cilju dodavanja nove ili izmene postojeće funkcionalnosti sistema
        - Modifikacija sistema kako bi zadovoljio nove zahteve

10.2. Šta je refaktoring, a šta reinženjering? Navesti sličnosti i razlike. (1)

- Re-engineering sistema
    - Reč je o restrukturianju ili ponovnom pisanju dela ili celog nasleđenog sistema bez promena njegove funkcionalnosti
    - Primenljiv je onda kada neki ali ne svi podsistemi većeg sistema zahtevaju često održavanje
    - Predstavlja ulaganje truda sa ciljem lakšeg održavanja. Sistem može biti restruktuiran i redokumentovan.
    - Obavlja nakon što je sistem bio održavan neko vreme i cena održavanja je porasla.
    - Re-engineering nasleđenih sistema se vrši korišćenjem automatskih alata, kako bi novodobijeni sistem bio lakši za održavanje.
- Refactoring sistema
    - Proces unapređenja programa kako bi se smanjila njegova degradacija kroz izmene, tj. predstavlja “preventivno održavanje”, koje smanjuje probleme pri budućim izmenama.

    - Uključuje izmenu programa kako bi se poboljšala njegova struktura, smanjila složenost ili povećala razumljivost. Kada se refaktoriše program, treba izbegavati dodavanje novih funkcionalnosti, već se skoncentrisati na njegovo unapređenje.
    - Kontinuirani proces unapređivanja kroz razvoj i evoluciju.
    - Koristi se kako bi se izbegla degradacija strukture i koda, a u cilju umanjenja cene i složenosti sistema.
- Re-engineering je stepen jači od refractoring-a, prosto kada je nemoguće snaći se u kodu, piše se deo složenog koda ispočetka, šta znam...