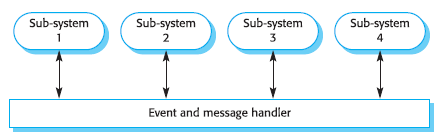
Softversko inženjerstvo - Kolokvijum II

1. **Navesti i kratko opisati osobine dobrog projekta.**

* **Osobine dobrog projekta su:**
  + **Hijerarhija -** dobar projekat bi trebalo da bude organizovan u dobro projektovanu hijerarhiju komponenata
  + **Modularnost -** treba izvršiti dekompoziciju sistema u posebne celine - module (na primer, razdvojiti podatke i obradu) sa jasno definisanim interfejsom.
  + **Nezavisnost -** treba grupisati slične stvari u nezavisne module. Ako se kasnije menjaju neke bitne odluke u projektu, posledice će biti lokalizovane.
  + **Jednostavan interfejs -** treba izbeći komplikovan korisnički interfejs, interfejse sa velikim brojem mogućnosti upotrebe kao i interfejse čija bi izmena mogla izazvati neželjene efekte.

1. **Navesti osnovne karakteristike i grafički ilustrovati *broadcast* arhitekturne modele.**

* Kod ove grupe modela, generisani događaj (event) se prosleđuje svim komponentama i podsistemima u sistemu. Svaka komponenta koja upravlja generisanim događajem može da obradi događaj. Efikasni su kod integracije podsistema koji se nalaze na različitim računarima u mreži. Podsistemi ne znaju da li će i kada eventi biti obrađeni. Podsistemi se registruju za određene događaje i kada se oni generišu, upravljanje se prenosi na podsistem koji upravlja tim događajem.



1. **Šta predstavlja i koji su ciljevi inspekcije softvera.**

* **Inspekcija softvera** predstavlja analizu statičkih reprezentacija sistema da bi se otkrili problemi. Cilj inspekcije sosftvera je ispitivanje izvorne reprezentacije SW radi otkrivanja anomalija i defekata. Inspekcija ne zahteva izvršenje SW tako da se može izvoditi pre njegove implementacije. Može biti primenjena na bilo koju reprezentaciju SW a prema istraživanjima, najefikasnija primena je na slučajeve korišćenja.

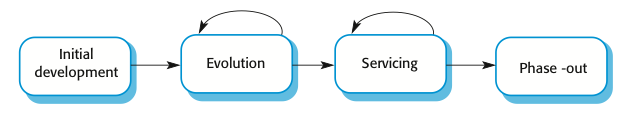
1. **Opisati i kratko objasniti strukturu test slučajeva.**

* U SWE **test slučaj** je skup uslova na osnovu kojih tester može utvrditi da li SW delimično ili potpuno ispunjava neki zahtev. Za testiranje SW potreban je veliki broj test slučajeva. RUP preporučuje kreiranje bar po 2 test slučaja za svaki zahtev. Test slučaj treba da sadrži opis funkcionalnosti koja se testira i kako treba pripremiti okruženje da bi bili sigurni šta testiramo.
* **Struktura test slučaja:**
  + **Uvod** - sadrži opšte informacije o test slučaju (Identifikator, Vlasnik ili kreator test slučaja, Verzija definicije, Naziv, Identifikator zahteva, Namena, Zavisnosti)
  + **Aktivnost test slučaja** 
    - Okruženje/konfiguracija test slučaja
    - Inicijalizacija
    - Finalizacija
    - Akcije
    - Ulazni podaci
  + **Očekivani rezultati** - opisuje šta će tester videti posle izvođenja svih koraka

1. **Navesti i kratko objasniti koje sve kategorije strategija za upravljanje rizikom postoje.**

* **Kategorije strategija za upravljanje rizikom:**
  + **Strategija izbegavanja** 
    - Smanjiti verovatnoću pojave rizika
    - Primenjeno kod defekta u komponentama
  + **Strategije minimizacije**
    - Smaniti uticaj rizika na projekat ili proizvod
    - Primenjeno kod bolesti osoblja
  + **Planovi za nepredviđene događaje**
    - Ako se rizik pojavi, ovi planovi definišu kako se radi sa rizikom
    - Primenje kod finansijskih problema organizacije

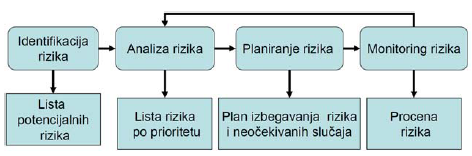
1. **Grafički ilustrovati i kratko opisati odnos između evolucije i servisiranja.**



* **Evolucija** predstavlja fazu u životnom ciklusu softverskog sistema gde se softver nalazi u operativnoj upotrebi i evoluira kako se novi zahtevi implementiraju u sistem.
* **Servisiranje** je faza u kojoj se softver i dalje operativno koristi, a izmene koje se vrše imaju za cilj održavanje njegove operativnosti. To znači da se ne dodaju nove funkcionalnosti, već se samo otklanjaju greške i vrše izmene usled promena u okruženju softvera.

1. **Kratko opisati i ilustrovati proces upravljanja rizikom.**

* **Proces upravljanja rizikom se sastoji od 4 faza:**
  + **Identifikacija rizika** - Identifikovati projektne, proizvodne i poslovne rizike
  + **Analiza rizika** - Proceniti verovatnoću i posledice identifikovanih rizika
  + **Planiranje rizika -** Skicirati planove da se izbegnu ili minimiziraju uticaji identifikovanih rizika
  + **Monitoring rizika -** Nadgledati rizike u toku projekta



1. **Ukratko opisati i navesti prednosti i nedostatke slojevitog (eng. *layered*) arhitekturnog modela.**

* **Slojeviti (Layered) model** 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.
* **Prednosti Layered modela:**
  + 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 Layered modela:**
  + Ne mogu svi sistemi da se lako organizuju po ovom modelu

1. **Šta predstavlja verifikacija, a šta validacija softvera?**

* **Verifikacija** odgovara na pitanje „Da li na pravi način gradimo proizvod?“ i ispituje da li je softver usaglašen sa svojom specifikacijom.
* **Validacija** odgovara na pitanje „Da li gradimo pravi proizvod?“ i ispituje da li softver radi ono što korisnik stvarno traži.
* Dva glavna cilja V&V su otkrivanje defekata u sistemu i procena da li je sistem upotrebljiv i koristan u realnim uslovima.

1. **Navesti, kratko opisati i uporediti različite metode testiranja softvera.**

* **Metode testiranja se mogu svrstati u 3 grupe:**
  + **Metode crne kutije -** Test inženjer pristupa SW koji testira preko interfejsa koji je namenjen krajnim korisnicima
  + **Metode bele kutije -** Test inženjer pristupa izvornom kodu i može pisati kod koji linkuje sa bibliotekama koje su linkovane sa softverom koji se testira. Obično se koristi kod komponentnog testiranja.
  + **Metode sive kutije -** Test inženjer može postaviti ili manipulisati nekom okolinom za testiranje i može videti stanje SW posle svake akcije. Koriste ih gotovo isključivo klijent-server test inženjeri ili inženjeri koji koriste bazu podataka kao repozitorijum informacija.
* Kriterijumi za ovu kategorizaciju metoda su: da li se pri razvoju test slučajeva pristupa izvornom kodu SW koji se testira i da li se testiranje vrši preko korisničkog interfejsa (UI) ili preko programskog interfejsa (API).

1. **Opisati i uporediti *top-down* i *bottom-up* pristupe projektovanja softvera.**

* **Top-down projektovanje SW-a:**
  + 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 se 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.
* **Bottom-up projektovanje SW-a:**
  + 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 podsistema (a koje korisnik direktno vidi) dobijaju u kasnim fazama implementacije.

1. **Navesti i kratko opisati tipove održavanja softvera.**

* **Tipovi održavanja softvera su:**
  + **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 prilagođavanja 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.

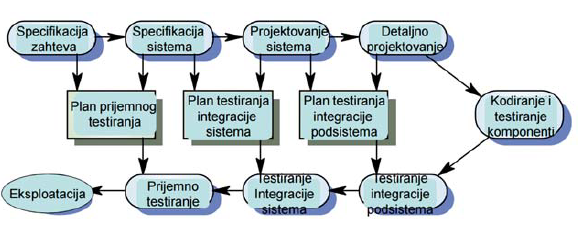
1. **Šta predstavlja upravljanje projektima i po čemu su softverski projekti osobeni?**

* Pod upravljanje projektima se podrazumeva **organizovanje**, **planiranje** i **raspoređivanje** softverskih projekata. **Upravljanje softverskim projektima** odnosi se na aktivnosti koje treba da osiguraju da se softver isporuči na vreme i po planu, a saglasno zahtevima organizacije koja ga razvija i organizacije koaj ga nabavlja.
* Softverski projekti su specifični zbog toga što:
  + Proizvod je nevidljiv
  + Proizvod je fleksibilan
  + Softversko inženjerstvo nema status zdrave inženjerske discipline kao što su mašinstvo, energetika, itd.
  + Proces razvoja softvera nije standardizovan
  + Mnogi softverski proizvodi su unikatni

1. **Ukratko opisati i navesti prednosti i nedostatke klijent/server arhitekturnog modela.**

* **Client-Server model** se koristi kod distribuiranih sistema. Sastoji se od skupa stand-alone servera koji obezbeđuju specifične servise (štampa, Web, baza podataka, ...), skupa klijenata koji pozivaju te servise i mreže koja omogućuje udaljeni pristup. Komponente ovog modela su serveri i klijenti a konektori mreža i servisi servera.
* **Prednosti ovog modela su:**
  + Efikasno korišćenje mrežnih sistema.
  + Omogućava korišćenje slabije HW-a za klijente, obzirom da server odrađuje većinu posla.
  + Lako dodavanje novih servera i upgrade postojećih.
* **Nedostaci ovog modela su:**
  + Neefikasna razmena podataka između klijenata (moraju da idu preko servera).
  + Redundantnost podataka.
  + Ne postoji centralni registar imena servera i servisa. Nije lako otkriti koji serveri i servisi su na raspolaganju.

1. **Nacrtati dijagram V-modela razvoja softvera.**



1. **Šta predstavlja i šta se dobija reinženjeringom sistema.**

* **Reinženjering sistema** predstavlja restruktuiranje ili ponovno pisanje dela ili celog nasleđenog sistema bez promene njegove funkcionalnosti. Primenljiv je onda kada neki ali ne svi podsistemi većeg sistema zahtevaju često održavanje. Reinženjering utiče na smanjenje rizika i smanjenje cene. Kod razvoja novog softvera postoji veliki rizik, a cena razvoja novog softvera je najčešće znatno veća nego cena reinženjeringa sistema.

1. **Objasniti razlike između statičke i dinamičke verifikacije softvera.**

* **Statička verifikacija** se odnosi na analizu statičkih reprezentacija sistema da bi se otkrili problemi. Mogu se koristiti alati za analizu koda i alati za pregledavanje dokumentacije koja je predmet inspekcije.
* **Dinamička verifikacija** se odnosi na izvršenje i posmatranje ponašanja proizvoda. Sistem se izvršava sa test podacima i posmatra se kako se ponaša u radu.

1. **Koje su prednosti a koji nedostaci kod Event-driven arhitekturnog modela?**

* Ovaj model se koristi kod sistema koji su upravljani eksterno generisanim događajima (events). Postoje 2 osnovne grupe ovih modela:
  + Broadcast modeli
  + Interrupt-driven modeli
* **Prednosti Event-driven modela:**
  + Podrška višestrukom korišćenju SW-a (reuse).
  + Laka evolucija sistema.
  + Lako uvođenje nove komponente u sistem (jednostavno se registruje za neki event).
* **Nedostaci Event-driven modela:**
  + Kada komponenta generiše događaj ona ne može da zna da li će neka komponenta da odgovori na njega i kada će obrada događaja biti završena.

1. **Navesti osnovne metode za projektovanje softvera i objasniti svaku od njih.**

* Postoje 3 osnovne grupe metoda za projektovanje SW-a:
  + **Strukturne metode**
  + **OO metode**
  + **Kombinovane metode**
* **Strukturne metode projektovanja** su funkcionalno orijentisane jer se softverski sistem projektuje sa funkcionalnog stanovišta u vidu osnovnih funkcija koje obavlja nad podacima koji čine stanje sistema. Podržane su od strane mnogih CASE alata i grafičkih notacija.
* **OO metode projektovanja -** Softverski sistem se posmatra kao kolekcija objekata u međusobnoj interakciji, pri čemu svaki objekat obuhvata određene podatke i funkcije za manipulisanje tim podacima. Uglavnom se koristi UML grafička notacija.
* **Kombinovane (hibridne) metode projektovanja** uključuju dobre osobine i strukturnih i OO metoda kako bi se došlo do dobro projektovanog softverskog sistema.

1. **Navesti osnovne arhitekturne modele za projektovanje softvera i objasniti Control arhitekturni model.**

* Osnovni arhitekturni modeli za projektovanje softvera:
  + **Repository (Skladište)**
  + **Pipe and Filter (Protočna obrada)**
  + **OO model**
  + **Client/Server model**
  + **Slojeviti model (Layered)**
  + **Event-driven model (Implicitno pozivanje)**
  + **Control model**
* **Control model** koristi se kod sistema gde je potrebna centralizovana kontrola. Kontrolni podsistem upravlja tokom informacija između ostalih podsistema. Postoje 4 osnovne grupe ovih modela: Call-return modeli, Manager modeli, Feed-back modeli, Open-loop modeli.

1. **Objasniti Alfa-testiranje i Beta-testiranje.**

* **Alfa testiranje** vrši se u alfa fazi životnog ciklusa softvera, tj. u toku razvoja softvera. Koriste se uglavnom metode bele kutije. Dodatna inspekcija se može vršiti metodama crne ili sive kutije, to rade posebni timovi za testiranje i to se naziva drugi stepen alfa testiranja.
* **Beta testiranje** je testiranje koje dolazi posle Alfa testiranja, u tzv. beta fazi razvoja SW proizvoda. Beta verzije SW proizvoda se distribuiraju ograničenom broju krajnjih korisnika koji koriste SW uz obavezu slanja izveštaja razvojnom timu o uočenim nedostacima. Alternativa je da se beta verzija SW javno publikuje čime se povećava broj testera i povećava verovatnoća da defekti u programu budu uočeni. Koriste se uglavnom metode crne kutije.