# GENERALNO TEORIJA

1. **Definirajte ulogu odgovornosti u objektno-usmjerenim sustavima? Razmotrite odnos odgovornosti i funkcionalnih zahtjeva. Na koji način se odgovornosti označavaju u dijagramu razreda?**
   Odgovornost (engl. responsibility) je nešto što sustav mora izvršiti. To je viša razina apstrakcije ponašanja od operacije.
   Svaki *funkcionalni zahtjev* mora se pripisati nekom razredu. Ako razred nema odgovornosti, tada je vjerojatno beskoristan.
   Označavaju se unutar uglatih zagrada.

2. **Nužna obilježja koja mora posjedovati programski jezik za svrstavanje u kategoriju objektno usmjerenih jezika su:**
   Apstrakcija (razred), apstraktni tip podataka (objekt), nasljeđivanje, polimorfizam.

3. **Čime su opisani različiti stilovi arhitektura programske potpore? Navedite najmanje 3 primjera stila arhitekture.**
   Opisuju se rječnikom ili topološkim ograničenjima koja moraju zadovoljiti svi članovi stila.
   Protok podataka, objektno usmjeren stil, repozitorij podataka, stil upravljan događajima.

4. **Navedi tri problema evolucijskog modela razvoja i oblikovanja programske potpore.**
   Proces razvoja nije jasno vidljiv, sustavi su loše strukturirani, često su potrebne posebne vještine.

5. **Navedi bar dva razloga za smanjenje međuovisnosti komponenti.**
   Zbog budućih promjena u produktu i zbog potencijalne buduće iskoristivosti samostalne komponente.

6. **Koja je namjena korištenja UML dijagrama?**
   Ujedinjeni jezik za modeliranje (UML) je normirani jezik opće namjene koji se koristi za modeliranje računalnih sustava temeljenih na objektno-orijentiranoj paradigmi.

7. **U koje dvije skupine možemo podijeliti UML dijagrame i koji UML dijagrami pripadaju kojoj skupini?**
   strukturni (statički) dijagrami: razreda, objekata, paketa, komponenti, razmještaja
   ponašajni (dinamički) dijagrami: aktivnosti, stanja, obrazaca uporabe*, sekvencijski, komunikacijski
   (*obrasci uporabe su ponašajni i statički)

8. **Kako se u sekvencijskom dijagramu označava petlja?**
   Na strelicu se napiše *uvjet+povr:=naziv(parametri) gdje zvjezdica označava petlju.

9. **Za razliku od sekvencijskog dijagrama kojem je u fokusu vremenska uređenost između događaja, u fokusu UML kolaboracijsko/komunikacijskog je:**
   Uređen redoslijed slanja poruka - ne prikazuje vremenske odnose. Veći naglasak na pregledu scenarija, odnosno na međusobnoj komunikaciji.

10. **Nasljeđivanje i agregacija, što se od toga treba programirati, a što ne?**
    Agregacija se programira, nasljeđivanje ne.

11. **Oblikovanje PP obuhvaća akciju inženjerstva zahtjeva. Navedi sve načine na koji se mogu izraziti zahtjevi.** *ili druga varijanta pitanja* **Nabrojati barem 3 načina zapisa korisničkih zahtjeva, odnosno zahtjeva sutava.**
    Strukturiranim prirodnim jezikom, specifičnim jezikom za oblikovanje (npr. SDL), grafičkom notacijom (npr. ERA, UML) i matematičkom specifikacijom (npr.vremenska logika).

12. **Na koje dijelove programske potpore je ispitivanje programskog koda usmjereno na otkrivanje pogrešaka u algoritmima, podacima i sintaksi? Navedite i primjere što se ispituje. Što je referenca za provođenje tih ispitivanja?**
    Na komponente.
    Ispituju se sučelja, strukture podataka, granični uvjeti, različiti tokovi izvođenja.
    Reference za provođenje tih ispitivanja su specifikacije zahtjeva.

13. **Navedite vrstu arhitekture programske potpore u koju se svrstava model-pogled-nadglednik (Model View Controller, MVC) i njezine osnovne značajke.**
    MVC se temelji na arhitekturi zasnovanoj na događajima.
    Model implementira centralni repozitorij i sadrži jezgrene funkcionalnosti i podatke.
    Nadglednik eksplicitno upravlja radom, rukuje ulaznim podacima korisnika.
    Pogled korisniku prikazuje informacije dobivene iz modela.

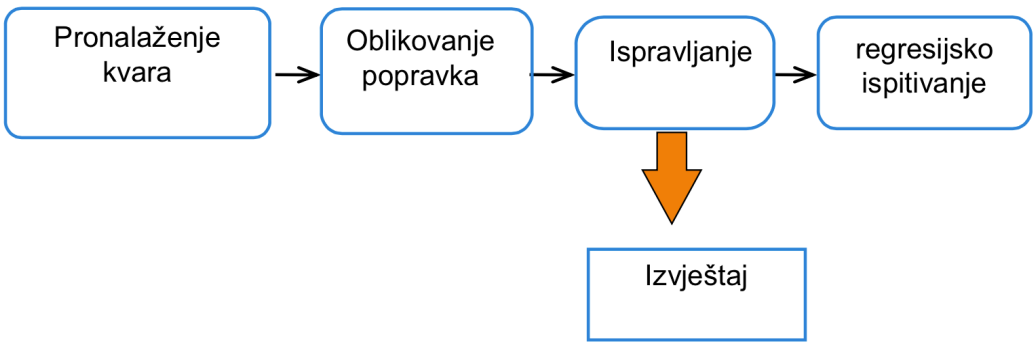
14. **Navedite tri tipična uzroka kvara sučelja komponente.**
    - pogrešna uporaba
    - nerazumijevanje sučelja
    - vremenske pogreške

15. **Definirajte Hipervizorski VM.**
    Hipervizorski VM je virtualan stroj kao dodatni sloj odmah iznad sklopovlja. Na njega se oslanjaju jedan ili više OS-ova gdje svaki smatra da je jedini OS u računalu.

16. **Navedite osnovne prednosti udomaćenih virtulanih strojeva (hosted virtual machine). Jesu li udomaćeni virtualni strojevi efikasniji od hipervizorskih?**
    Udomaćen virtualni stroj kao i svaki drugi primjenski program izvodi se iznad OS-a u posebnom procesu.
    Domaćinski OS osigurava pristup sklopovlju.
    Manje je efikasan od hipervizorskog VM, ali osnovna prednost odvajanja ostaje.

17. **Nacrtati graf reprezentacije tijeka programa (control flow graph).**
    http://java.dzone.com/articles/how-draw-control-flow-graph

18. **Skicirajte dijagram osnovnih koraka procesa otklanjana pogrešaka u fazi implementacije programiranja.**



**19. Na koji način se opisuje arhitektura programske potpore?**
Arhitektura programske potpore opisuje se modelima koji svaki sadrži jedan ili više pogleda (dijagrama).

**20. Navedite cilj analize zahtjeva u procesu izlučivanja zahtjeva.**
Cilj analize zahtjeva je utvrđivanje problema, nekompletnosti i nejednoznačnosti u izlučenim zahtjevima.

**21. Što je osnovni cilj oblikovanja programske potpore zasnovanog na modelima?**
Cilj: povećanje razine apstrakcije, poboljšanje komunikacije, produktivnost, održavanje

**22. Koje su značajke (atributi) dobre programske potpore?**
- prihvatljivost - razumljiv, koristan i kompatibilan s ostalim sustavima
- pouzdanost - korisnik mora vjerovati u pouzdanost sustava
- održavanje - evolucija je sukladna izmijenjenim zahtjevima
- efikasnost - razumljiva uporaba resursa

**23. Princip ugovornog oblikovanja (design by contract) postavlja neke zahtjeve na pozivanu proceduru. Navedi te zahtjeve.**
- preduvjeti koje pozvana metoda traži da budu ispunjeni prije početka izvođenja (preconditions)
- uvjeti koje pozvana metoda mora osigurati *nakon* završetka izvođenja (postconditions)
- invarijante na koje pozvana metoda neće djelovati (mijenjati) pri izvođenju

**24. Koje su faze u životnom ciklusu RUP procesa?**
1. početak (inception), u kojemu se definira doseg projekta, razvoj modela poslovnog procesa, specifikacija početnih zahtjeva
2. razrada (elaboration), kada se definiraju plan projekta, specifikacija značajki i temelji arhitekture sustava,
3. izgradnja (izrada, construction), koja se sastoji od oblikovanja, programiranja, i ispitivanja
4. prijenos (transition), pri čemu se završni proizvod prenosi korisnicima i postavlja u radnu okolinu.

**25. Nabroji pet jezgrenih (temeljnih) aktivnosti (engl. Core Workflows) RUP modela procesa koje se pojavljuju s različitim intenzitetima u svim fazama.**
Jezgrene aktivnosti: model poslovnog procesa, zahtjevi, analiza i oblikovanje, implementacija, test, razmještaj.
Potporne aktivnosti: upravljanje promjenama, upravljanje projektom, briga o okolini.

**26. Koje su generičke aktivnosti u procesu programskog inženjerstva?**
Specifikacija, razvoj i oblikovanje, validacija i verifikacija, evolucija.

**27. Što je formalna verifikacija (precizno odgovoriti, ne matematičko-logičkim simbolima)?**
To je postupak provjere da formalni model dijela izvedenog sustava (/), odgovara formalnoj specifikaciji (S) sa matematičkom izvjesnošću ("odgovara" = logički zadovoljava).

**28. Koji je model procesa programskog inženjerstva najpogodniji za male i srednje interaktivne sustave?**
Evolucijski.

**29. Koji model ima najmanje novog koda?**
Komponentni.

**30. Koje aktivnosti ima RUP?**
Zahtjevi, analiza, oblikovanje, implementacija, test.

**31. Dva pristupa iteracijama u procesu programskog inzenjerstva?**
Inkrementalni i spiralni.

**32. Svaki proces programskog inženjerstva sadrži iteracije u pojedinim fazama. Iteracije se mogu izvesti inkrementalnim ili spiralnim postupkom. Koje su prednosti (navedi barem tri) inkrementalnog pristupa iteracijama?**
- kupac dobiva vrijednost sa svakim inkrementom
- temeljna funkcionalnost sustava se ostvaruje u ranim fazama projekta
- rani inkrementi služe kao prototipovi na temelju kojih se izlučuju zahtjevi za kasnije implemente
- smanjen rizik neuspjeha projekta
- prioritetne funkcionalne usluge sustava imaju mogućnost detaljnijeg ispitivanja

**33. Objasni razliku između proširenog obrasca extend i uključenog obrasca include.**
Vezom uključivanja se povezuju dva obrasca uporabe na način da jedan obrazac u tijeku svog izvođenja u potpunosti izvede uključeni obrazac uporabe pri čemu trenutak izvođenja uključenog obrasca nije specificiran dijagramom.
Vezom proširenja se povezuju dva obrasca uporabe pri čemu jedan proširuje funkcionalnost drugog. Proširenje se ostvaruje ukoliko je zadovoljen određeni uvjet definiran u točki proširenja.

**34. Jedan od 12 principa oblikovanja i izbora arhitektura programske potpore traži povećanje kohezije komponenata (modula). Zašto se nastoji ostvariti taj princip (navedi barem dva osnovna razloga)?**
Veća je kohezija ako se grupira međusobno povezane elemente, sve ostalo je izvan. Razlozi: olakšano razumijevanje, povećana ponovna uporabljivost modula, lakša zamjena.

**35. (Kako izražavamo korisničke) Nabroji četiri metode u izlučivanju zahtjeva u oblikovanju programske potpore.**
Intervjuiranje, scenarij, obrasci uporabe (use cases), specificiranje dinamičkih interakcija u sustavu (sekvencijski dijagrami), promatranje rada, izrada prototipa.

**36. Koji su mogući pogledi (viewpoints) i koje korisnici sustava koriste?**
Pogledi interakcije, indirektni pogledi, pogledi domene primjene. Korisnici koriste poglede interakcije.

**37. U inženjerstvu zahtjeva u oblikovanju programske potpore, navedi klasifikacije zahtjeva s obzirom na razinu detalja:**
Korisnički zahtjevi, zahtjevi sustava, specifikacija programske potpore.

**38. U inženjerstvu zahtjeva u oblikovanju programske potpore, navedi klasifikacije zahtjeva s obzirom na sadržaj:**
Funkcionalni, nefunkcionalni, zahtjevi domene primjene.

**39. Obrasci uporabe koriste se pri modeliranju funkcionalnih / nefunkcionalnih zahtjeva (zaokruži).**

**40. U okviru procesa ispitivanja programske potpore poredajte (zadane, op. autora) faze.**

1. ispitivanje komponenti
2. integracijsko ispitivanje
3. ispitivanje sustava
4. ispitivanje prihvatljivosti
5. ispitivanje instalacije
6. alfa ispitivanje
7. beta ispitivanje

**41. Rezultat procesa inženjerstva zahtjeva je:**

Specifikacija programskog proizvoda.

**42. U arhitekturi PP koja se zasniva na repozitoriju podataka postoje 2 velike podskupine: baze podataka i oglasna ploča. Navedi temeljnu razliku između ovih podskupina.**

Baza podataka - vanjski procesi iniciraju promjenu sadržaja.
Oglasna ploča - promjena sadržaja inicira vanjske promjene.

**43. U arhitekturi protoka podataka pažnja je usredotočena na prijenos podataka između aktora. Upravljački tok je implicitan. Navedi barem 3 mehanizma upravljanja u protoku podataka.**

Guranje (PUSH), povlačenje (PULL), guranje/povlačenje (PUSH/PULL), pasivni.

**44. Program izgrađen u objektnoj arhitekturi je brži ako ima više/manje (zaokruži) dinamičkih povezivanja?**

**45. OCSF**

**a) Koje metode se obavezno moraju implementirati?**

HandleMessageFromServer(), HandleMessageFromClient()

**b) Što se događa sa klijentima kada se pokrene metoda stoplistening() na serveru?**

Poslužitelj prestane osluškivati za nove klijente, sa trenutnima i dalje komunicira.

**46. Korištenjem OCSF izgrađen je dio klijentske aplikacije, tj. razred TClient, koji je nastao tako da je:** naslijeđen razred AbstractClient **te je implementirana metoda:** handleMessageFromServer()**. Pokaži fragmentom koda kako se razred koristi ako bi se poslužitelju koji sluša na adresi 10.0.0.33:12345 poslala poruka sadržaja: „test“. Na ispitu je bio zadan popis razreda poslužitelja i klijenta.**

```
TClient a("10.0.0.33", 12345);
a.openConnection();
a.sendToServer("test");
```

**47. Upisati DA/NE ovisno o standardizaciji na binarnoj razini i standardizaciji na razini izvornog koda.**

| | **binarni kod** | **izvorni kod** |
|---|---|---|
| **CORBA** | NE | DA |
| **.NET** | DA | NE |

**48. Što je programsko inženjerstvo? Koja je razlika između programskog inženjerstva i računalne znanosti?**

Programsko inženjerstvo je disciplina koja se bavi metodama i alatima za profesionalno oblikovanje i proizvodnju programske potpore uzimajući u obzir cjenovnu efikasnost. Računarska znanost se bavi teorijom i temeljima; programsko inženjerstvo se bavi praktičnim problemima razvoja, oblikovanja i isporuke korisnog računalnog programa.

**49. Što je CASE (engl. Computer-Aided Software Engineering)?**

To su programski produkti namijenjeni automatiziranoj podršci aktivnostima u procesu programskog inženjerstva (specifikacija, oblikovanje i evolucija).

**50. Koje su vrste projekata (nabroji bar 4 tipa) u programskom inženjerstvu? Što znači pojam legacy system?**

*Korektivni projekti* (otklanjanje pogrešaka).
*Adaptivni projekti* (izmjene temeljem promjene operacijskog sustava, baza podataka, pravila i zakonskih odredaba, …).
*Unapređujući*, aditivni (dodavanje nove funkcionalnosti).
*Re-inženjerstvo* (npr. unutarnje izmjene kako bi se olakšalo održavanje).
*Sasvim novi projekti* (engl. green-field) – u manjini.
*Integrativni* (oblikovanje novoga okruženja iz postojećih programskih komponenata i cjelina, engl. framework).
Većina projekata je evolucijska i vezana je uz održavanje starijih sustava (engl. legacy).

**51. Nabroji generičke aktivnosti inženjerstva zahtjeva.**

Studija izvedivosti, izlučivanje zahtjeva, analiza i specifikacija zahtjeva, validacija zahtjeva, upravljanje zahtjevima.

**52. Cilj validacije zahtjeva i zašto se provodi?**

Cilj validacije je pokazati da dokument zahtjeva predstavlja prihvatljiv opis sustava koji naručitelj doista želi.

**53. Koja je temeljna karakteristika modela PI zasnovanog na komponentama koja ga razlikuje od ostalih modela?**

Sustav se integrira višestrukom uporabom postojećih komponenata ili uporabom komercijalnih, gotovih komponenata.

**54. Ukratko i precizno objasni ekstremno programiranje, koje su prednosti, koje nedostaci tog pristupa oblikovanja programske potpore**

Ekstremno programiranje je vrsta inkrementalnog postupka razvoja i isporuke koji se bazira na razvoju, oblikovanju i isporuci vrlo malih inkremenata funkcionalnosti.
Prednosti: kontinuirano poboljšanje koda, sudjelovanje korisnika u razvojnom timu
Nedostataci: nerazumljivost, ne podupire ponovnu uporabu rješenja.

**55. Navedi tri vrste verifikacije PP. U koju spada testiranje (ispitivanje)?**

Dinamička, statička i formalna, valjda u dinamičku (ispitivanje izvođenjem programa sa stvarnim podacima).

**56. Tko testira komponente prije integracije?**

Programer.

**57. Dio nekog problema riješen je na sljededi način: *catpopis.txt|grupa z1|sort|analiza>rez.txt* Koji je to tip arhitekture?**
Protok podataka, cjevovodi i filtri.

**58. Cilj, čemu služi i na koji se konkretan način koristi konceptualna razina arhitekture sustava?**

**59. Povećanje kohezije je jedan od temeljnih principa razvoja PP i kriterija. Objasni funkcijsku koheziju!**
Podsustav ili modul ima veliku koheziju ako grupira međusobno povezane elemente, a sve ostalo stavlja izvan grupe.
Funkcijska kohezija: Kôd koji obavlja pojedinu operaciju je grupiran, sve ostalo izvan, npr. modul obavlja jednu operaciju, vraća rezultat bez popratnih efekata.

**60. Kod OO arhitekture, objasni metodu razreda.**
Metode definirane unutar nekog razreda i deklarirane ključnom riječi static.

**61. Objasni nasljeđivanje i njegova svojstva u OO programiranju.**
Nasljeđivanje je proces kod kojeg se jedan razred stvara na temelju drugog tako da se dodaju specifična obilježja i ponašanje.
Svojstva nasljeđivanja
Prednosti:
- mehanizam apstrakcije pogodan za organizaciju
- mehanizam ponovne uporabe u oblikovanju i implementaciji
- organizacija znanja o domeni i sustavu

Nedostaci:
- razredi se ne mogu razumjeti bez poznavanja nadrazreda
- nasljeđivanja uočena u fazi analize mogu dati neučinkovita rješenja

**62. Razlika tradicionalnog pristupa ispitivanja programa temeljenog na pronalaženju pogreške i modernog pristupa ispitivanju, posebno u svijetu moderne tehnologije razvoja PP.**
U modernom ispitivanju se uz metode tradicijskog ispitivanja uvode formalne metode ispitivanja(poglavito formalna verifikacija modela).

**63. Što je potpuno ispitivanje? Kako se provodi?**
Potpuno ispitivanje je ispitivanje po završetku kojeg znamo da nema neotkrivenih pogrešaka. Ispitati sve moguće vrijednosti varijabli, sve kombinacije ulaza, svih sekvenci izvođenja, svih hardware/software konfiguracija, svih načina uporabe.

**64. Od čega se sastoji ispitni slučaj?**
Sastoji se od određenog para ispitnih podataka i očekivanog izlaza.

**65. Neka je sustav koji prima na ulazu cijele brojeve u rasponu 10-20 podvrgnut funkcijskom ispitivanju (kao crna kutija). Navedi pogodne ispitne slučajeve tako da se pokriju sve ekvivalentne podjele.**
0-9, 10-20, 21-inf.
napomena: ispitni slučaj je par (ulaz, očekivan izlaz) - ispituju se brojevi iz svake skupine te je potrebno ispitati granične slučajeve
npr. 0, 9, 10, 15, 20, 21, 25

**66. Navedi koje bi vrijednosti trebalo isprobati kako bi se provelo uspješno funkcijsko ispitivanje metode koja na ulazu treba primiti cijeli broj u rasponu [-100, 100].**
npr**.** -200(može bilo koji broj manji od -101); **-101**; **-100**; 0(bilo koji broj između -100 i 100); **100**; **101**; 200(može bilo koji broj veći od 101)

**67. Objasni što je to i kako funkcionira implicitno pozivanje kod arhitekture zasnovane na događajima.**
Primjenski program ne poziva rutinu za obradu događaja izravno niti neizravno. Prekidna rutina (odziv na iznimku) se aktivira preko prekidnog vektora.

**68. Navedite 3 entiteta koja čine temeljni model web uslužne arhitekture (ne treba protokole).**
Registar usluga i posrednik, ponuditelj usluge, tražitelj usluge.

**69. Navedi barem dvije razlike između programske komponente u komponentno temeljenoj arhitekturi i objekta u objektno usmjerenoj arhitekturi programske potpore.**
Definicija objekta je tehnička; ne uključuje pojam nezavisnosti.
Kompozicija objekta nije namijenjena širokom krugu korisnika.
Ne postoji, niti će postojati, tržište objekata.
Objekti ne podržavaju paradigmu „plug-and-play“.

**70. Prema pravilima ekvivalencije pretvoriti u CNF oblik: (P ∧ ¬Q) ∨ (¬R ∨ P).**
(P ∧ ¬Q) ∨ (¬R ∨ P) =
= [(P ∧ ¬Q) ∨ ¬R] ∨ P =
= [(¬R ∨ P ) ∧ (¬R ∨ ¬Q)] ∨ P =
= [P ∨ (¬R ∨ P ) ] ∧ [P ∨ (¬R ∨ ¬Q)] =
= (P ∨¬R) ∧ (P ∨ ¬R ∨ ¬Q)
kaj se, ako se ne varam, može skratit na (P ∨ ¬R)
(P ∧ ¬Q) ∨ (¬R ∨ P) = P ∧ (¬Q ∨ true) ∨ ¬R = (P ∨ ¬R) - ovo je u redu jer je CNF oblik "logičkih umnožaka" (povezani su "i" operatorom) i ovo se može gledati kao umnožak s jednim članom. Oblik gore zapisan je također dobar i ova dva izraza su ekvivalentna. U CNF-u je bitno samo da se dobiju "umnošci" pa stoga nakon što se dobije gornji izraz nije ga potrebno kratiti u donji, ali meni izgleda ljepše. Isto tako se moglo dobiti i (P ∨¬R ∨ Q) ∧ (P ∨ ¬R ∨ ¬Q) što bi isto vrijedilo.

**71. Što znači ispravan (sound), a što kompletan (complete) formalni logički sustav koji sadrži skup logičkih izraza {T} i skup dozvoljenih pravila {L}?**
Ispravan kada je svaka pravilima dokazana formula ujedno i logička posljedica. Kompletan kada je svaku logičku posljedicu moguće dokazati pravilima.

**72. Zaokruži netočno:**
a) A [p U EF ¬r]
b) AEFr – kvantifikatori ne mogu biti u niz
c) FGr – operatori ne mogu biti u nizu
d) AF [(r U g) ∧ (p U r)] – operatori ne mogu biti u nizu
e) A [¬p U A[q U r]]
f) A ¬G ¬p – operatori se ne mogu negirati

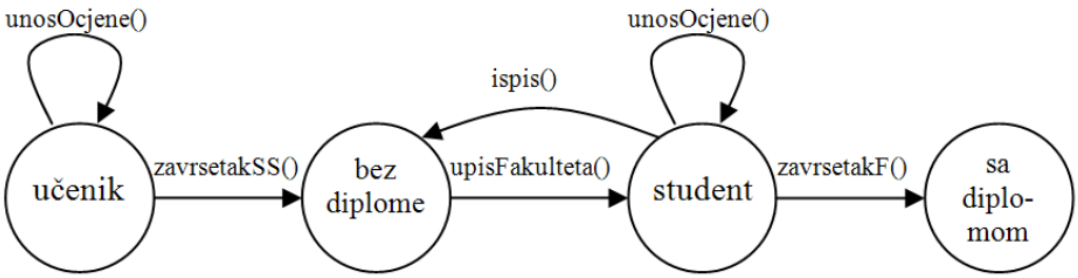
**73. Ako su istinite formule: (1) ¬P, (2) Q, (3) (P ili ¬Q ili R), općim postupkom dokazivanja teorema (opovrgavanje, obaranje) pokažite je li R logička posljedica navedenih formula.**

- R logička posljedica skupa, (1), (2), (3) - ako je svaki model tog skupa ujedno i model formule R
- model formule R je bilo koji u kojem je R=T
- iz (1) P=F, iz (2) Q=T, iz (3) (P v ¬Q v R) ako uvrstimo P i Q=>F v F v R=> da bi bilo točno, mora R biti T
-dakle, jedini model za skup, (1), (2), (3) - je: P=F, Q=T, R=T, što je i model za R, konačno, R je logička posljedica skupa {(1), (2), (3)}

**74. Neka je poznat skup stanja u kojima je istinit propozicijski simbol p. Objasniti riječima za koja stanja je istinita formula vremenske logike EXp.**
To su sva stanja iz kojih se može doći, tj. iz kojih postoji prijelaz u stanje u kojem p=T.

**75. U ispitivanju programske potpore koristi se termin "testni slučaj". Što je to?**
Jedan par: ulaz i očekivani izlaz.
Testni slučaj je određen scenarij koji testira sustav ili komponentu na određen način, koji može ili ne mora biti specifičan.

**76. Za razred StatusOsobe zadan je dijagram stanja prema slici, gdje su prijelazi uzrokovani pozivima metoda razreda.**




**a) Kako oblikovati testni slučaj?**
**b) Navedite testni slučaj.**
a) Tako da se prođu sva stanja i prijelazi.
b) unosOcjene → zavrsetakSS → upisFakulteta → ispis → upisFakulteta → unosOcjene → zavrsetakF
ili
unosOcjene → zavrsetakSS → upisFakulteta → unosOcjene → ispis → upisFakulteta → zavrsetakF

**77. Neka je stvorena raspodijeljena arhitektura prema objektnom klijent-poslužiteljskom radnom okviru (OCSF). Ako poslužitelj ima spojena dva klijenta i provodi interakciju sa svojim administratorom, koliko je ukupno aktivnih dretvi na poslužiteljskoj strani?**
2 klijenta + 1 admin + 1 koja čeka nove konekcije

**78. Koliko je najmanje potrebno testova za broj_lukova=12, broj_čvorova=8 Nema dodatnih komponenti?**
CV(G) = Lukovi – Čvorovi + 2*P = 6

**79. UML dijagram razreda povezuje (asocijacijom) razrede A i B tako da postoji brojnost: [A] 1..4_____3..5 [B]**
**Neka se tijekom izvođenja u jednom času u sustavu nalazi 3 objekta nastala iz razreda.**

**Koliko se najmanje, a koliko najviše objekata razreda može u tom času nalaziti u sustavu?**
MIN: 3 (Jer sve tri instance od A mogu referencirati isti B)
MAX: 3*5=15

**80. Kada kažemo da je {Γ,L} konzistentan?**
Skup Γ je konzistentan akko (ako i samo ako) ne sadrži formule na temelju kojih bi ωi i ¬ωi istovremeno bili teoremi.

# PREDIKATNA LOGIKA

Znakovi: ∧ ∨ ⟶ ⇒ ↔ ¬ ∃ ∀

1. **Svako dijete je mlađe od svoje majke.**
   ∀x(dijete(x) ⇒ ∃y(majka(y,x) ∧ mlađi(x,y)))

2. **Tko god ima majku ima i oca.**
   ∀x(∃y majka(y,x) ⇒ ∃z otac(z,x))

3. **Ante ima najviše jednu sestru.**
   ∀x(sestra(x,Ante) ⇒ ¬∃y(sestra(y,Ante)∧ ¬=(x,y)))

4. **Ante ima točno jednu sestru.**
   ∃x(sestra(x,Ante) ∧ ∀y(sestra(y,Ante) ⇒ (x=y)))

5. **Ante ima barem dvije sestre.**
   ∃x∃y(sestra(x,Ante) ∧ sestra(y,Ante) ∧ ¬=(x,y))

6. **Ako Ana voli Milovana, tada postoji netko tko voli Anu a ne voli Milovana.**
   voli(Ana, Milovana) ⇒ ∃x (voli(x,Ana) ∧ ¬voli(x,Milovana))

7. **U Hrvatskoj postoje najmanje dva nacionalna parka.**
   ∃x∃y[postojiu(x,Hrvatska) ∧ postojiu(y,Hrvatska) ∧ ¬=(x,y)]
   ili
   ∃x∃y[NP(x) ∧ HR(x) ∧ NP(y) ∧ HR(y) ∧ ¬=(x,y)]

8. **Svaku štetu plaća neka osoba.**
   ∀x∃y(plaća(y,x))

9. **Najveći miš se ne boji najmanje mačke.**
   ∀x∀y ((miš(x) ∧ najveći(x) ∧ mačka(y) ∧ najmanji(y)) ⇒ ¬boji(x,y))
   ili
   ∃x∃y (miš(x) ∧ najveći(x) ∧ mačka(y) ∧ najmanji(y) ∧ ¬boji(x,y))
   ili
   ∀x∀y (miš(x) ∧ najveći(x) ⇒ (mačka(y) ∧ najmanji(y) ∧ ¬boji(x,y)) - ovo vrijedi samo u slučaju da je skup y skup veličine jednog člana i taj član je najmanja mačka. Inače je izraz neistinit.
   Ispravno bi bilo sljedeće(za bilo koju veličinu skupa y):
   ∀x∀y (miš(x) ∧ najveći(x) ⇒ ¬(mačka(y) ∧ najmanji(y) ∧ boji(x,y))

10. **Svaki student je mlađi od nekog nastavnika**
    ∀x(student(x) ⇒ ∃y(nastavnik(y) ∧ mlađi(x,y))

11. **Svaki sin mog oca je moj brat.**
    ∀x(sin_mog_oca(x) ⇒ moj_brat(x))

12. **Ana voli svu Marijinu braću.**
    ∀x[brat(x,Marija) ⇒ voli (Ana, x)]

13. **Ana voli jednog Marijinog brata.**
    ∃x[brat(x, Marija) ∧ voli(Ana, x)]

14. **Među životinjama nalazi se mačak koji jede sir.**
    ∃x(zivotinja(x) ∧ macak(x) ∧ jede_sir(x))

**15. Samo mrtav vanzemaljac je dobar vanzemaljac.**
    ∀x (DobarVanzemaljac(x) → ( Vanzemaljac(x) ∧ Mrtav(x) ) )

    *u zadatku nam je izrečena pogodba na izvrnut način te moramo paziti šta je uvjet, a šta posljedica. U zadatku nam kaže da zapravo ne postoji vanzemaljac koji je istovremeno dobar i ne-mrtav(živ)-samo mrtav je dobar :D- to ne mora značiti da ako je mrtav onda mora biti i dobar jer nam o tom ništa ne govori. Sve što znamo o njemu je to da ako je dobar onda sigurno mora biti mrtav.
    *(Vanzemaljac(x) ∧ Mrtav(x)) sam mogao napisati samo kao MrtavVanzemaljac(x).

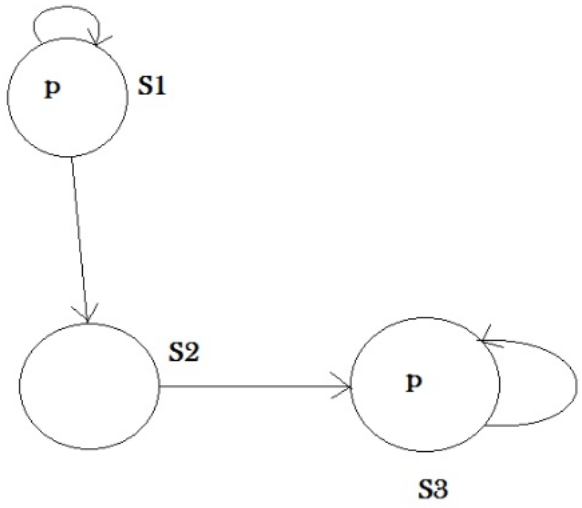
**16. Za svako brdo u HR postoji brdo u BIH koje je više.**
    ∀x ( ( Brdo(x) ∧ UnutarHR(x) ) → ∃y ( Brdo(y) ∧ UnutarBiH(y) ∧ VišiOd(y,x) ) )

**17. Brijač brije sve ljude osim onih ljudi koji se sami briju.**
    ∀x ¬ ( Brije(x,x) ↔ Brije(Brijač,x) )
    ili
    ∀x ( ( Brije(x,x) → ¬Brije(Brijač,x) ) ∧ ( ¬Brije(x,x) → Brije(Brijač,x) ) )

# CTL, Kripke

Znakovi: ∧ ∨ ⟶ ⇒ ↔ ¬ ∃ ∀

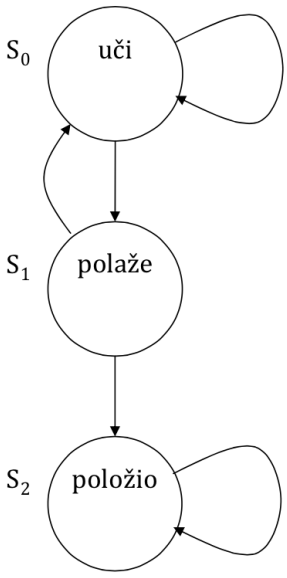
1. **Za (S1, S2, S3) sa donje slike odrediti da li je istinita CTL formula AF (AG p).**




S1: ne, S2: da, S3: da

Objašnjenje:
1. p - čvorovi sa p su S1 i S3
2. AG p - čvorovi koji duž svih svojih puteva imaju s1 i/ili s3 te sami jesu s1 ili s3: samo s3 (s1 može preći u s2, a to nam se ne sviđa)
3. AF AG p - čvorovi koji svim svojim putevima uvijek dođu u s3: s2 i s3 (s1 beskonačno mnogo puta može ići sam u sebe, a to nam se ne sviđa)

2. **Kripke struktura zadana je slikom desno. Za koja stanja grafa vrijede formule:**
   **a) AG(uči ⇒ AF(položio))**
   **b) E(uči U položio)**




   a) S2
   b) S2

3. **Napišite CTL formulu za sljedeću izjavu: Iz svih stanja gdje je p=istinito (ima stanje u kojima je p=neistinito) izvođenje programa će svakako konačno završiti u stanju gdje je q=istinito.**
   AG(p ⇒ AF q)

4. **Preslikaj rečenicu prirodnog jezika u dobro definiranu formulu CTL vremenske logike: "Proces uvijek može iz nekritičnog stanja u slijedećem trenutku zahtijevati ulazak u kritično stanje“.**

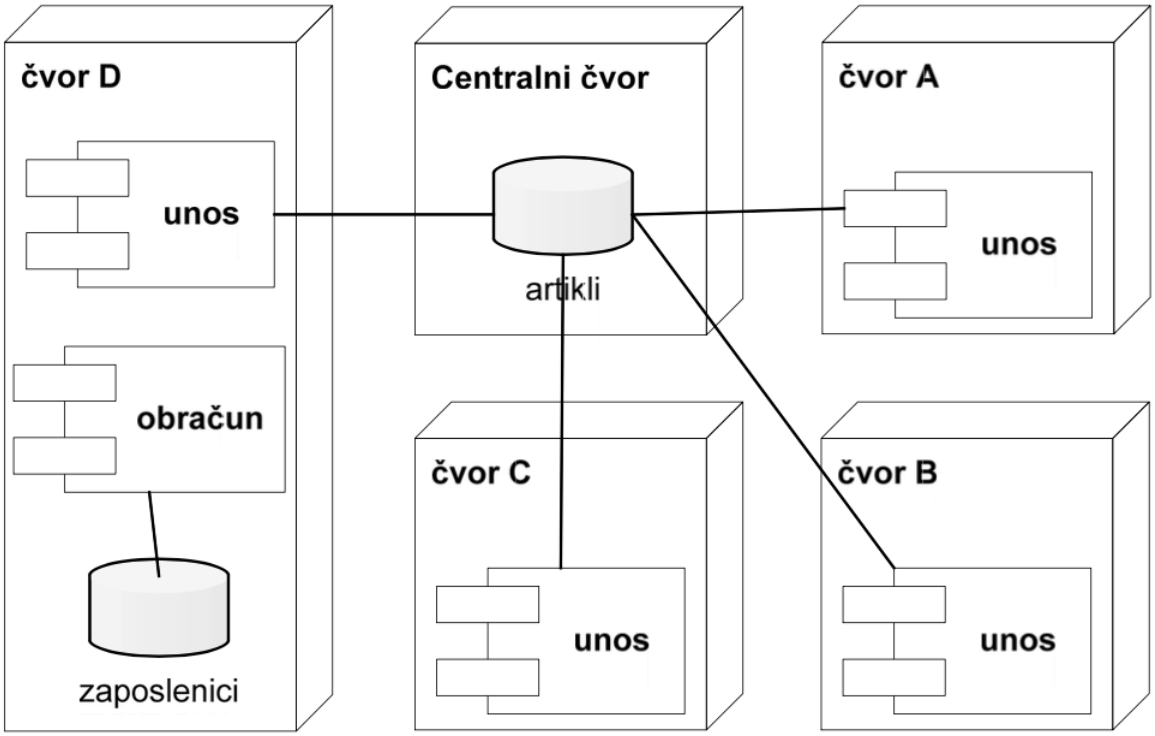
Neka su oznake za propozicijske simbole: n-proces nije u kritičnom stanju; c – proces je u kritičnom stanju; t – proces zahtijeva ulaz u kritično stanje.
AG (n => EX (t))

5. **Preslikaj u vremensku logiku CTL:**
   **a) Sustav će, gledajući iz bilo kojeg stanja uvijek konačno doći u stanje "odobreno".**
   **b) Iz početnog stanja, postoji put u kojem se konačno ulazi u stanje gdje vrijedi "p"; od tog stanja dalje to "p" vrijedi uvijek i u idućem stanju.**
   **c) Uvijek vrijedi da ako vrijedi "p" tada će na svim putevima vrijediti "q" dok ne počne vrijediti "r".**
   a) AG(AF(odobreno))
   Gledajući iz bilo kojeg stanja (AG) sustav će uvijek konačno doći (AF) u stanje u kojem vrijedi odobreno = true.
   b) EF(p $\wedge$ AG p)
   c) AG(p => A(q U r))
   Uvijek vrijedi (AG) da ako vrijedi p (p =>) tada će na svim putevima (A) vrijediti q dok ne počne r (q U r).

6. **U CTL-u prikazati rečenicu "Svjetlo može biti ili upaljeno ili ugašeno (ex-ili) korištenjem (oba) atoma: *upaljeno* i *ugašeno*.**
   AG((upaljeno $\wedge$ ¬ugašeno) $\vee$ (¬upaljeno $\wedge$ ugašeno))

# UML

1. **"Baza artikli se nalazi u centralnom računalu, a aplikacija unos koja koristi tu bazu nalazi se na računalima A, B, C i D. Program obračun nalazi se na računalu D i koristi zasebnu bazu zaposlenici koja se tamo nalazi."**
   **Prikazati rečeno odgovarajućim UML dijagramom.**



# LITERATURA


1. Dijagrami stanja - http://www.anylogic.com/anylogic/help/index.jsp?topic=/com.xj.anylogic.help/html/discrete/Statecharts.html

2. Graf reprezentacije tijeka programa (control flow graph) - http://java.dzone.com/articles/how-draw-control-flow-graph

# NEDOREČENI GENERALNI ZADACI

1. Zadan sekvencijski dijagram pretvoriti u komunikacijski.

2. Bilo je neko stablo i nekoliko izraza i vidit dal vrijede, ono s CTL logikom, tipa AF q i tako to.

3. Taj nisam zapisao, ali bila je slika klijentskih dretvi i serverskih te njihova komunikacija i pitanja o tome.

4. OCSF, bio je dijagram razreda, predavanje o OCSF-u str 62., tri potpitanja, koje sucelje implementira neki razred, koju metodu zove metoda HandlemessageFromServer i jos neko s main

5. Nacrtan neki dijagram razreda, napisat sta je sta i sta dijagram predstavlja.

6. Bilo je zadan dijagram klasa i napisat za koje se izvodi dinamicko za koje staticko povezivanje, razredi su bili Shape2D, Rectangle i tako nesto

7. Trebalo je nacrtati dijagram razreda knjiga, sadržaj knjige, indeks pojmova, poglavlje u knjizi.

8. Tri zadatka s crtanjem dijagrama, jedan usecase:administrator i korisnik pa zadano sta koji radi, drugi sekvencijski:klijent A i B salju poruke Posluzitelju, i treci:nacrtat dijagram razreda za alarmni sustav, auto vrata na auto,sirena tako neke stvari su bile.

9. Zadan dijagram razreda i treba napisati što znamo o sustavu.