4 boda) Definirajte ulogu odgovornosti u objektno-usmjerenim sustavima? Razmotrite odnos odgovornosti i funkcionalnih zahtjeva. Na koji način se odgovornosti označavaju u dijagramu razreda?

2b) Nužna obilježja koja mora posjedovati programski jezik za svrstavanje u kategoriju objektno usmjerenih jezika su:

-Identitet, Objekt, Razredi, Nasljeđivanje, Polimorfizam (prezentacija 6, Modularizacija i objektno usmjerena paradigma, str 17)

2b)Čime su opisani različiti stilovi arhitektura programske potpore? Navedite najmanje 3 primjera stila arhitekture

- rječnikom (tipovima komponenata i konektora) i topološkim ograničenjima koja moraju zadovoljiti svi članovi stila. Primjeri: protok podataka, objektno usmjereni stil, repozitorij podataka, upravljan događajima (prezentacija 5, Arhitektura programske potpore, str 14)

2b)Na koje dijelove progr. pot. je usmjereno 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?

-To je ispitivanje komponenti, ispituju se sučelje, podaci struktura, rubni uvjeti, nezavisni putovi, iznimke

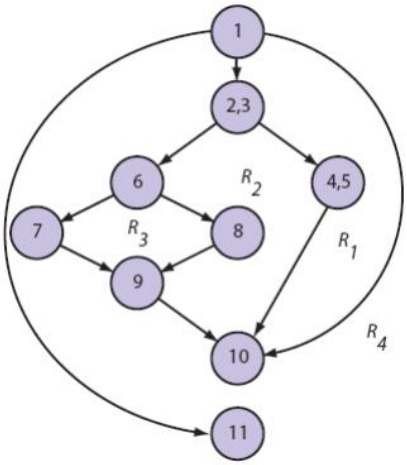
-referenca?

2b) Navedite vrstu arhitekture prog. pot. u koju se svrstava model-pogled-nadglednik (Model View Controller, MVC) i njezine osnovne značajke.

-Arhitektura zasnovana na događajima. Komponente se međusobno ne pozivaju eksplicitno, neke komponente generiraju signale (događaje), neke komponente su zainteresirane za pojedine događaje te se prijavljuju na strukturu za povezivanje komponenta, događajima se upravlja asinkrono... (prezentacija 9,Primjeri arhitekturnih obrazaca , str 29

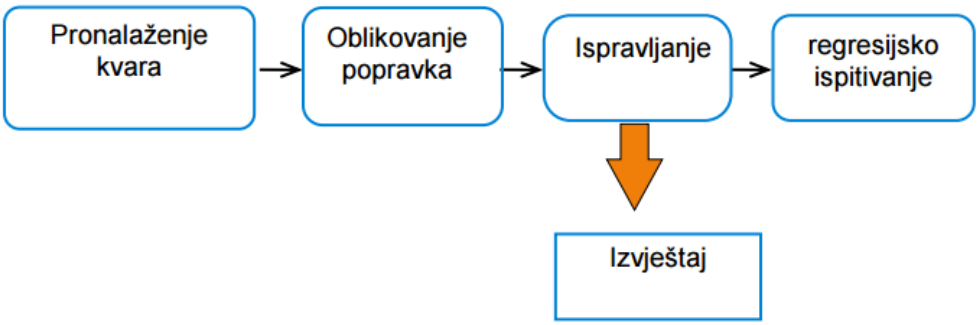
2b) Navedite osnovne prednosti udomaćenih virtulanih strojeva (hosted virtual machine). Jesu li udomaćeni virtualni strojevi efikasniji od hipervizorskih?
-Ovdje mi čak ni Google nije pomogao.
4b) Control flow graph nacrtati

-čisto opće informacije radi, jedan primjer kako to inače izgleda:



2b) Skicirajte dijagram osnovnih koraka procesa otklanjana pogrešaka u fazi implementacije programiranja.

Otkrivanje kvara (engl. debugging):



1b) Na koji način se opisuje arh. prog. pot?

- rječnikom (tipovima komponenata i konektora), topološkim ograničenjima koja moraju zadovoljiti svi članovi stila

1b) Navedite cilj analize zahtjeva u procesu izlučivanja zahtjeva..

-cilj izlučivanja zahtjeva je saznanje što više informacija o domeni primjene i o tome koje usluge trebaju biti podržane s takvim performansama i ograničenjima

1b) Što je osnovni cilj oblikovanja prog. pot. zasnovanog na modelima?
Model opisuje faze od začetka projekta do kraja životnog vijeka proizvoda
1b) .. osnovne značajke dobrog programskog produkta
-prihvatljivost za korisnika, pouzdanost, mogućnost laganog održavanja

1b) RUP- 3 stvari u fazi razrade (elaboration)

-plan projekta, specifikacija značajki, temelji arhitekture sustava

može ovo neko poredat točnim redosljedom, iz završno 08/09 (navodno dolazi na rokovima?)

test komponenti
test integracije
funkcijski test
test performansi
test prihvatljjivosti
test uporabe
test instalacije

(tema 10, slajd 76-77):

Ispitivanje komponenti

Ispitivanje integracije

Ispitivanje sustava

Funkcijsko ispitivanje

Ispitivanje performansi

Ispitivanje prihvatljivosti

Ispitivanje instalacije

Zimski 14/15:

Teorija je bila skoro sva iz onog PDF-a na materijalima, bilo je OCSF- okruzenje, objektno orijentirana paradigma, šta će se ispisat, nadopisat kod za implementacijom sučelja.. uglavnom najosnovnije stvari.. dijagrami su isto bili dosta lagani po meni.

+

Ukupno 30(ili 29, ne sjećam se točno) zadataka, 10 na zaokruživanje(1 bod), 4 sa dijagramima(5 bodova). Bio je jedan zadatak za crtanje flowcharta nekog programa i još neki računi vezani uz njega koji se isto morao pisati na košuljicu/papir(3 boda), a ostali zadaci su bili onako kao na završnom, pitanje i odgovori(2 boda). Od dijagrama su došli use-case, komunikacijski, dij. stanja i dij. razreda.

Zimski 11/12:

što se događa sa klijentima kad se pokrene metoda stoplistening na serveru?

posluzitelj prestane osluskivati za nove klijente, a sa onima sa kojima komunicira nastavlja ''razgovor''

.NET - na binarnoj razini? na razini izvornog koda*?*

-na binarnoj razini

Navedi barem dvije razlike između programske komponente u komponentno temeljenoj arhitekturi i objekta u objektno usmjerenoj arhitekturi programske potpore.
-Objekt predstavlja pojedini element u sustavu, dok komponenta može biti i skup objekata, program, radni okvir...

(Prezentacija 09_AR_2, str 50) Sastavljanje komponenti u jednostavan produkt može izvesti vješt korisnik kod oblikovanja zasnovanog na komponentama, dok to kod objektno usmjerenog oblikovanja zahtjeva znanje objektnog programiranja, ali zato oblikovanje komponenata zahtjeva vođenje računa o mnogo korisnika kod oblikovanja zasnovanog na komponentama.
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 moze doci (iliti iz kojih postoji prijelaz) u stanje u kojem vrijedi p.

Pitanje je bilo " Upisati DA i NE ovisno o standardizaciji na binarnoj razini i standardizaciji na razini izvornog koda."

-za CORBA i .NET. Za CORBA je NE binarno, DA izvorno, ali za .NET obrnuto. (gradivo: arhitektura posluzitelj klijent)

"U ispitivanju programske potpore koristi se termin „testni slučaj“.Što je to?".

- Jedan par: ulaz i očekivani izlaz (pri testiranju).

OCSF - Općenito trebaš znati stvaranje veze, prekidanje veze, početak slušanja servera implementirati (par redaka), znati koje metode se gdje moraju implementirati, a koje su opcionalne te što se događa s klijentima nakon prestanka slušanja servera bla bla. Također treba znati koliko se dretvi za što stvara i mislim da je to to. Naravno da ne morate znati sve metode svih klasa niti onaj chat program ni tako slično. Samo svrhu tog paketa i neke važnije stvari. (imho)

-početak slušanja poslužitelja:

Metoda *listen()* na serveru.
ServerSocket serverSocket = new ServerSocket(port);
Socket clientSocket = serverSocket.accept();

-otvaranje veze klijenta s poslužiteljem:

Socket clientSocket = new Socket(host, port);
Za otvaranje veze klijent pokrece metodu *openConnection* koja je definirana u *AbstractClient* razredu

-zatvaranje veze:

Metoda *closeConnection*

-prestanak slusanja:

Metoda *stopListening()* u <<control>>, signalizira run metodi da prestane slušati (nema novih klijenata), ali klijenti i dalje komuniciraju. Alternativa je metoda *close* u <<control>> koja prestane slušati kao i *stopListening*, ali i odspaja sve klijente

-metode koje se MORAJU implementirati

Na klijentskoj strani metoda *handleMessageFromServer()*, a na serverskoj strani metoda *handleMessageFromClient()*

(prezentacija 8, Arhitekturni obrasci, str 39-76)

Zadatak s testiranjem programske potpore - Bio je zadan nekakav potprogram za sortiranje, trebalo je nacrtat one čvorove i lukove, izračunat CV(G), i za zadano polje, [2,7,5] izračunati postotak pokrivenosti testiranja stanja (ili tako nešto)

CV(G) = Lukovi – Cvorovi + 2*P

Lukovi = broj lukova u grafu

Čvorovi = broj čvorova u grafu

P = Broj povezanih komponenti (potprograma, prekidnih rutina i sl.) – često se zada da je 0

Da se nađe... teorijska pitanja su bila nešto detaljnija, nekoliko pitanja od prošlih godina je bilo spojeno 2 u 1, trazio se neki manji dijagram stanja za nacrtat (dok se garažna vrata otvaraju/zatvaraju svijetli žuto svijetlo, kod otključavanja/zaključavanja zvuk), objasnit općenito dijagram stanja, dijagram za testiranje nekakav, koji testovi su neophodni, use case dijagrami najbolje opisuju koji tip zahtjeva, pitanje sa "povećaj koheziju" i koje su prednosti, onaj dijagram sa geom. tijelima, kakvo je povezivanje korišteno za Rectangle i getBoundingRect() metodu, što je dinamičko povezivanje i što kako utječe na performanse, 4 osnovna tipa veza u use case, značajke dobre dokumentacije, kako izgleda nešto prije faze implementacije i za kog je namijenjeno, osim tražene funkcionalnosti i performansi 4 osnovne značajke dobrog softvera, dalje se ne sjećam.
Use case dijagram je bio ok, sekvencijski lakši nego na 1. mi, class dijagram dosta opširan.

Jesenski 14/15

- pitanja sa 13/14 (bold i u 14/15)

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

Programsko inženjerstvo – disciplina koja se bavi metodama i alatima za profesionalno oblikovanje i produkciju programske potpore uzimajući u obzir cijenu

Razlika – Računalna znanost se bavi temeljima koji su nužni da bi se suštinski razumijeli problemi s kojima se programeri susreću, a programsko inženjerstvo je orjentirano na praksu i rješavanje problema klijenata.

2. Što su CASE sustavi?

Computer-Aided Software Engineering – programski produkti namijenjeni automatiziranoj podršci aktivnostima u procesu programskog inženjerstva

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

Vrste projekata: evolucijski, korektivni, adaptivni, unaprijeđujući, re-inženjerstvo programskog proizvoda, integrativni projekti, hibridni

Legacy SOFTWARE (ne znam jel system točno prepisano) su stariji programski proizvodi ostavljeni novim inženjerima u nasljeđe. Povezano sa evolucijskim projektima.

4. Nabroji generičke aktivnosti inženjerstva zahtjeva

Studija izvedivosti, izlučivanje zahtjeva i analiza zahtjeva, specifikacija zahtjeva, validacija zahtjeva

5. Cilj validacije zahtjeva i zašto se provodi

Cilj validacije je provjera da li zahtjevi koje su dobiveni od klijenata zaista definiraju sustav koji korisnik želi. Provodi se zato što je naknadno ispravljanje pogreške u zahtjevima često mnogo puta skuplje od jednostavnog ispravljanja pogreške u implementaciji (za ovu drugu rečenicu nisam siguran, ali i ova prva govori zašto se provodi. U krajnjoj liniji, provodi se da bi se ostvario njen cilj?)

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

Temeljna karakteristika je ponovna uporabljivost programskih komponenata.

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

Ekstremno programiranje je metoda agilnog pristupa razvoja programske potpore koja se zasniva na razvoju, oblikovanju i isporuci funkcionalnih djelova. Prednosti su brza isporuka konačnog proizvoda i manji gubitak vremena na stvaranju dokumentacije i općenito na birokraciji, a glavni nedostatci su nestabilnost zahtjeva i nerazumljivost sustava zbog nedokumentiranja, što smanjuje mogućnost ponovne uporabe rješenja.

8. 3 vrste verifikacije PP. U koju spada testiranje(ispitivanje)

-Statička, dinamička i formalna. Testiranje spada u dinamičku verifikaciju.

9. Cilj, čemu služi i na koji se konkretan način koristi konceptualna razina arh. Sustava

Nisam siguran jer nisam našao konkretan odgovor al trebalo bi biti nešto tipa da pomaže u daljnjem planiranju razrade projekta jer omogućava uvid u to kako će u konačnici arhitektura projekta.
10. Povećanje kohezije je jedan od temeljnih principa razvoja PP i kriterija. Objasni funkcijsku kojeziju!

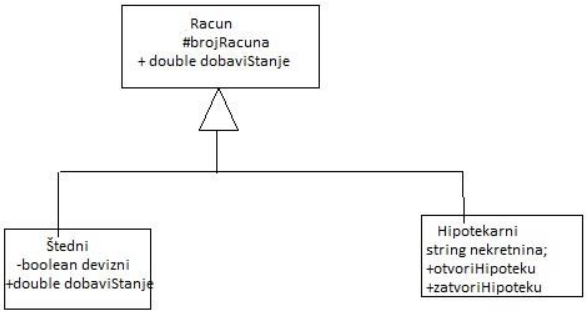
(Generalno kohezija: podsustav ili modul ima veliku koheziju ako grupira međusobno povezane elemente, a sve ostalo stavlja izvan grupe)

Funkcijska kohezija: kod koji obavlja pojedinu operaciju je grupiran, sve ostalo izvan

11. Kod OO arhitekture, objasni metodu razreda

Metoda je funkcija koja opisuje neko ponašanje razreda..?

12. Objasni nasljeđivanje i njegova svojstva u OO programiranju. (slika dole). I bilo je pitanje nešto tipa hoće li se izvesti statičko ili dinamičko povezivanje za dobaviStanje u razredu Štedni



Dinamičko povezivanje se događa kod poziva metode nadrazreda, tj kad postoji hijerarhija. Ako se povezivanje za dobaviStanje u razredu Štedni povezuje preko razreda Štedni, onda će povezivanje biti statičko, ali ako se poziva preko razreda Račun onda će biti dinamičko.

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

Server ima dretvu za osluškivanje novih klijenata i jednu za upravljanje klijentima, a prilikom dolaska svakog novog klijenta stvara novu dretvu koja se bavi pojedinim klijentom. Klijent ima dvije dretve, jedna koja služi za prikaz podataka korisniku i druga za slušanje komunikacije, odnosno prati dostigle poruke od servera.

14. Razlika tradicionalnog pristupa ispitivanja programa temeljenog na pronalaženju pogreške i modernog pristupa ispitivanju, posebno u svijetu moderne tehnologije razvoja PP

Tradicionalni način je ispitivanje, a moderan formalna verifikacija (najbolji odgovor koji sam našao, skripta Procesi programskog inženjerstva na ferwebu, str 4)

15. Potpuno ispitivanje? kako se provodi?

Potpuno ispitivanje provodi se ispitivanjem svih mogućih vrijednosti varijabli, kombinacija ulaza, sekvenci izvođenja programa, SW/HW konfiguracija, svih mogućih načina uporabe programa.

16. Od čega se sastoji ispitni slučaj

Ispitni slučaj se sastoji od para ulaza i očekivanog izlaza (I,O)

17. Navedi koje bi vrijednosti trebalo isprobati kako bi se provelo uspješni funkcijsko ispitivanje metode koja na ulazu treba primiti cijeli broj u rasponu [-100, 100]

(Općenito funkcijsko ispitivanje: engl. Black box testing, ispitivanje nema znanja o provedbi (programskom kodu, komponentama, komunikaciji unutar sustava itd.), nego se koncentrira samo na ispitne slučajeve, odnosno na odnos ulaza i izlaza)

Za gornji primjer: odaberu se vrijednosti blizu graničnih slučajeva (za -100 i 100) te jedan reprezentativan slučaj, npr u sredini (broj 0). Dakle, ispitivati će se za vrijednosti -101, -100, 0, 100 i 101.

18. Preslikaj rečenicu u predikatnu logiku prvog reda tako da najprije definiraš sve potrebne predikate i konstante (*ne identično na 14/15)

"Ako Ana voli Milovana, tada postoji netko tko voli Anu a ne voli Milovana"

Predikati:

Voli(X,Y) – X voli Y

Konstante:

Ana, Milovan

Predikatna logika:

$$\mathrm{Voli}(\mathrm{Ana}, \mathrm{Milovan}) \Rightarrow \exists X(\mathrm{Voli}(X, \mathrm{Ana}) \wedge \neg \mathrm{Voli}(X, \mathrm{Milovan}))$$

19. 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) AF(AG odobreno)

b) AF(AG p => AX p)

c) AG(p => A(q U r))

(za ova 3 nisam siguran, CTL mi nije jača strana. A ni OPP)

20. Objasni što je to, kako funkcionira implicitno pozivanje kod arh zasnovane na događajima

Implicitno pozivanje bazira se na tome da komponenta ne poziva direktno procedure, nego najavi da se dogodio događaj, a zatim sustav pokrene sve procedure koje su prijavljene da se trebaju pokretati prilikom nekog događaja. (izvor: Wikipedia)

+ 14/15: Uz to, bilo je još uloga i svrha arhitekture u procesu prog. inž., nešto sa big bang što u životu nisam čuo, arhitektura protoka podataka (njena 3 obilježja ili što već), spiralni model objasniti detaljnije, što su iteracije u procesu prog. inž. i koji modeli to koriste, od CTL logike je bilo samo ono da su zadane svakakve formule i označi koje su neispravne.
Zadatak sa OCSF-om je bilo broj dretvi na poslužiteljskoj strani ako su spojena 2 klijenta, broj dretvi na klijentskoj strani i napisati onu famoznu metodu koja mora biti implementirana.

-Uloga arhitekture: Strukturira razvojni projekt i samu programsku podršku. Smanjuje cijenu oblikovanja, razvoja i održavanja programske podrške, omogućava ponovnu uporabu rješenja, poboljšava razumljivost, poboljšava kvalitetu proizvoda, razjašnjava zahtjeve, omogućuje donošenje temeljnih inženjerskih odluka, omogućuje analizu i rano uočavanje pogrešaka u oblikovanju.

-Big bang je pristup integracijskom ispitivanju, prilikom čega se integriraju sve komponente bez prethodnog ispitivanja. U slučaju pogrešaka problem predstavlja otkrivanje mjesta pogreške.

-Arhitektura protokola podataka: Komunikacijski protokoli – jedan jezik formiraju poruke koje klijenti šalju poslužitelju, drugi por koje poslužitelj šalje klijentima, a ta dva jezika i pravila konverzacije čine protokol. (nekako mislim da se to nije odnosilo na ovo ali nisam našao nešto bolje.

-Spiralni model razvoja specifikacije programske potpore je trostupanjski ciklus koji se sastoji od ponavljanja specifikacije, validacije i izlučivanja zahtjeva. U ranim fazama procesa, najveći intenzitet aktivnosti je na razumijevanju poslovnih i nefunkcionalnih zahtjeva visoke razine apstrakcije. Zatim se proučavaju korisnički zahtjevi i izrađuju prototipovi sustava, da bi se u vanjskim slojevima spirale konačno prešlo na detaljno izlučivanje, analizu i specifikaciju svih zahtjeva. Na taj način, spiralni model izravno podržava razradu zahtjeva prema razini detalja. Broj iteracija po spirali pritom može varirati. Kod agilnih metoda razvoja programske potpore, umjesto razvoja prototipa proizvoda, može se ići u simultani razvoj specifikacije i implementacije. (skripta Procesi programskog inženjerstva, str 26)

-Iteracije – pojedini prolaz kroz neku fazu razvoja programske potpore prilikom čega se dodaje novi sadržaj ili otklanjaju pogreške iz dokumentacije ili izvršne datoteke. (iz skripte: Često se može čuti kako određeni dokument ili izvršna datoteka prolazi kroz iteracije. Time se želi reći da neki dokument ima više inačica i da se u svakom koraku dodaje novi sadržaj, ili otklanjaju pogreške. Procesi programskog inženjerstva, str 36). Postoje dva međusobno ovisna pristupa iteracijama: Inkrementalni i spiralni. Iteracije se mogu primijeniti na bilo koji generički model procesa programskog inženjerstva i javljaju se u svakoj fazi procesa programskog inženjerstva.

-OCSF: (princip sa dretvama je c/p od gore): Server ima dretvu za osluškivanje novih klijenata i jednu za upravljanje klijentima, a prilikom dolaska svakog novog klijenta stvara novu dretvu koja se bavi pojedinim klijentom. Klijent ima dvije dretve, jedna koja služi za prikaz podataka korisniku i druga za slušanje komunikacije, odnosno prati dostigle poruke od servera. U ovom slučaju znači da server ima 2 dretve za osnovno funkcioniranje i 2 za spojene klijente, znači 4 sve ukupno, a svaki klijent uvijek ima 2.

Metoda za implementirat na serveru: *handleMessageFromClient()*, a na klijentu *handleMessageFromServer()*

Bilo je nešto manje zadataka s dijagramima (samo 16 bodava - 3 zadatka - obrasci uporabe, dijagram razreda i komponenti), ali pitanja su očekivana (generički procesi i to), bilo je zadataka s formalnom verifikacijom (definicija logičke posljedice i modela), kripke + CTL logika, predikatna

logika (napiši varijable/konstante i napiši "rečenicu" u predikatnoj logici).

Od iznenađenja bio je zadatak s dinamičkim i statičkim povezivanjem (binding), napisati primjer koda za svaki (mislim da to nismo službeno radili na OPP-u), + 2 boda na dijagram stanja koji je bio relativno složen pa sam ga preskočio.

Da, i zadatak s obrascima uporabe je IDENTIČAN onome iz UML priručnika. Dijagram razreda se mogao dosta dobro napisati, bilo je sitnica tipa enumeration (ako sam ja to dobro napravio?), napisati stereotipe na <<interface>> i <<abstract>> i puno klasa koje ovise jedne o drugima (svaki djelatnik ovisi o direktoru, svaki radnik ovisi o voditelju odjela, odjel ovisi o voditelju odjela pa je ispalo dosta neuredno).

Zadnji je bio dijagram komponenti, pisalo je doslovno SVE (samo je trebalo znati kako se crtaju komponente i paketi), ako ste riješili ijedan primjer nije trebao biti problem potpuno točno nacrtati (ja nisam riješio NIJEDAN pa mislim da mi je dobro rješenje). :)

++ naučit stara pitanja (napamet sve podjele(nabrajanja), u ispitu je bilo 8 zadataka ak se ne varam koji kazu navedi, nabroji itd),

++ dijagrame i logiku (ona 2 predavanja, zadatke, dobro je ak ste imali umjetnu inteligenciju jer se tamo jos podrobnije radi logika). Bilo bi suludo ic kroz sva ona predavanja i mislit si sta je najbitnije, to ako se ima vremena, onda malo citas pa se nadas da ti nesto ostane u glavi. I da, nabavit rucni sat za svaki slucaj da vidis koliko se vremena jebes na nekom zadatku, jer ode vrijeme, nema se puno vremena za kemijanje i ono najgore, nema se vremena za provjeravanje jesi li tocno rjesio zadatak.

Ljetni 13/14:

Pitanja rok ljetni
Bilo nas je 5,sva su bila opisna,evo nekih kojih se sjecam

Genericke aktivnosti u procesu programskog inzenjerstva?

-Specifikacija, razvoj i oblikovanje,validacija i verifikacija,evolucija

Koji model procesa programskog se primjenjuje kod malih i srednjih interaktivnih sustava?

-Evolucijski

Koji model ima najmanje novog koda?

-Komponentni

RUP koje aktivnosti ima?

-Zahtjevi,analiza,oblikovanje,implementacija,test

Dva pristupa iteracijama u procesu programskog inzenjerstva?

-Inkrementalni i spiralni

Kod onih 12. Principa oblikovanja i kriterija za odabir arhitekture, zasto je dobro povecat koheziju?(barem dvije prednosti):

-Veca je kohezija ako se grupira medusobno povezane elemente, sve ostalo je izvan. Prednost je ta sta olaksava razumjevanje i promjene u sustavu

Bilo je nesto s izražavanjem zahtjeva, nisam sigurna jesu li bile metode u izlučivanju(ako da to su:Intervjuiranje,scenarij,obrasci uporabe,dijagrami interakcija(sekv.dij)**) ili je bilo kako izrazavamo korisnicke zahtjeve(**strukturiran prirodni jezik,poebni jezici za opis oblikovanja, uml,matematica specifikacija**)**

Različiti dionici imaju različit pogleda na zahtjeve. Koji su moguci pogledi i koje korisnici sustava koriste? Pogledi interakcije,indirektni pogledi,pogledi domene primjene.

Korisnici koriste poglede interakcije

Zahtjevi s obzirom na sadrzaj.

Funkcionalni,nefunkcionalni,domene primjene

Dvije formule, ro i omega, koja je razlika u ekvivalenciji koja u logickoj posljedici

Pretvorit 3 recenice u predikatnu logiku(Ante ima najvise jednu sestru, Ante ima tocno jednu sestru, Ante ima barem dvije seste)

-Nigdje nisam našao konkretne primjere sa brojevima, ali ovo je moja nekakva pretpostavka:

<u>Predikati:</u>

ImaSestra(X,Y) – subjekt X ima Y (y je broj) sestra

VeciOd(X,Y) – X je veci od Y

<u>Konstante:</u>

Ante

<u>Rješenja:</u>

Ante ima najviše jednu sestru

$\exists Y(\mathrm{ImaSestra}(\mathrm{Ante}, Y) \wedge \mathrm{VeciOd}(Y, 0) \wedge \neg \mathrm{VeciOd}(Y, -1))$

(s tim da teoretski možda i ne treba definirat da je y manji od -1, al nikad se ne zna)

Ante ima točno jednu sestru

$\mathrm{ImaSestra}(\mathrm{Ante}, 1)$

Ante ima barem dvije sestre.

$\exists X(\mathrm{ImaSestra}(\mathrm{Ante}, X) \wedge \mathrm{VeciOd}(X, 1))$

Još jednom naglašavam, ovo je čisto nagađanje, nisam vidio primjer da se može ovako staviti konstanta Ante i broj u istu predikatnu rečenicu.

Bilo je ono sa racunanjem kod ispitivanja a da se produ svi putovi tako nekeko:

-zadane grane,čvorovi i P se valjda uvrsti jedan jer je sve povezano pa je formula: lukovi-cvorovi+2P

Bilo je zadano ispitivanja, poredat ih po vremenu:

ispitivanje komponenti,funkcijsko ispitivanje,ispitivanje performansi,ispitivanje prihvatljivosti, ispitivanje instalacije, ne znam jel ovo dobro poredano i bila su jos dva zadana cini mi se

(ovaj primjer je gore rješen)

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

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

Trebalo je nacrtat dijagram razreda knjiga,sadrzaj knjige,indeks pojmova,poglavlje u knjizi

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

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

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

OCSF iz AR1(mislim da se tako zove preznetacija) i pogledaj staticko i dinamicko povezivanje iz prezentacije OO1 i nauci crtati control flow graph... Moja preporuka