Dr Milunka Damnjanović, red.prof,
OBJEKTNO ORIJENTISANE TEHNIKE PROJEKTOVANJA SISTEMA

5 Analiza zahteva

Šta mora da ostvari MODEL ZAHTEVA?

Treba postići maksimalno razumevanje između korisnika i onih koji će projektovati i realizovati sistem. To se radi pravljenjem modela koji moraju da ispune nekoliko ciljeva:

- **Moraju da sadrže kompletan opis onoga što softver treba da radi.**
- **Moraju da predstave ljude, fizičke stvari i koncepte koji su važni za analitičara da bi razumeo šta se dešava u domenu aplikacije.**

Šta mora da ostvari MODEL ZAHTEVA?

- **Moraju da prikažu veze i interakcije između ljudi, stvari i koncepata.**
- **Moraju da prikažu poslovnu situaciju dovoljno detaljno za procenu mogućih projekata.**
- **Idealno, trebalo bi da su organizovani na takav način da kasnije mogu da budu korisni pri projektovanju softvera.**

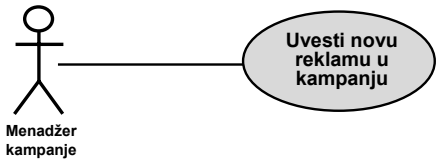
Kako modelovati zahteve?

- **Razvijeni softver mora da odražava situaciju zbog koje je razvijen.**
- **Opis domena aplikacije mora da sadrži i konceptualnu strukturu.**
- **Konceptualna osnova OO pristupa.**
- **Dijagram klasa u UML-u konstruisan je da predstavi analizu zahteva na takav način da struktura modela može kasnije da se direktno prevede u softverske komponente.**

Realizacija korisničkih slučajeva:

Da bi se od inicijalnog korisničkog slučaja došlo do implementacije softvera koja adekvatno ispunjava zahteve identifikovane korisničkim slučajem, potrebna je najmanje jedna iteracija kroz sve razvojne aktivnosti - od modelovanja zahteva do implementacije. U odnosu na jedan korisnički slučaj, ova aktivnost je poznata kao *realizacija korisničkog slučaja*.

Primer:

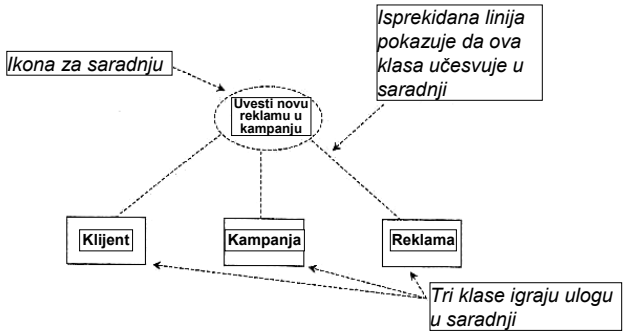


Saradnja:

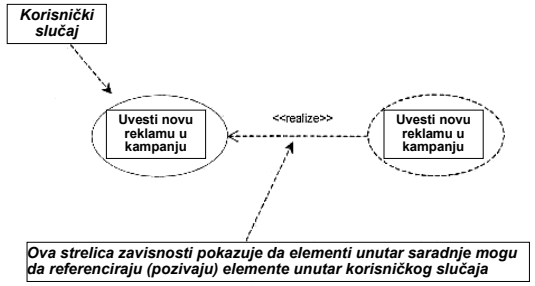
Realizacija korisničkog slučaja obuhvata identifikaciju mogućeg skupa klasa, zajedno sa interakcijom koja obezbeđuje funkcionalnost korisničkog slučaja.

Skup klasa poznat je kao *saradnja* .

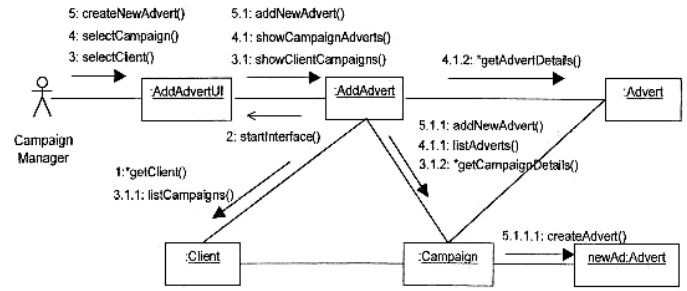
Primer saradnje:



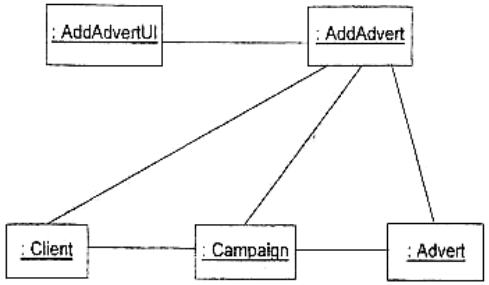
Saradnja gledana sa spoljašnje strane:



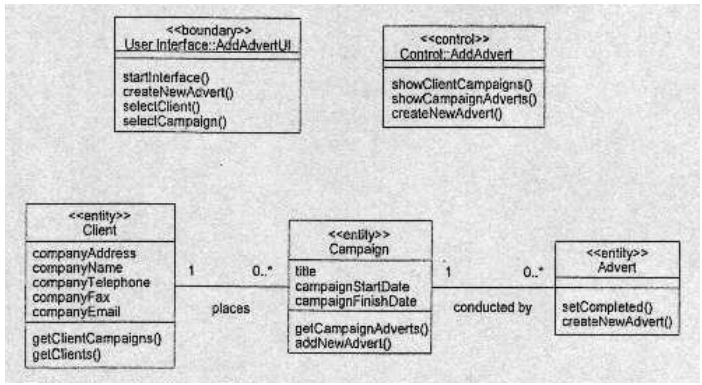
Dijagram saradnje:



Saradnja predstavljena kao dijagram objekata (uzoraka):



Saradnja predstavljena kao dijagram klasa:



Stereotipi klasa analize:

Novi tip elementa koji se modeluje i koji proširuje semantiku metamodela. Stereotip mora biti zasnovan na postojećim tipovima i klasama u metamodelu. Stereotipi mogu da prošire semantiku ali ne strukturu postojećih klasa. Neki stereotipi definisani su u UML-u, a ostale može definisati projektant.

U stvari, to znači da uzorci stereotipa klase imaju zajednički pogled na zadatak, što ih na značajni način razlikuje od ostalih stereotipova.

Tri stereotipa klasa analize:

- granične klase,
- kontrolne klase i
- klase entiteta (jezgra)

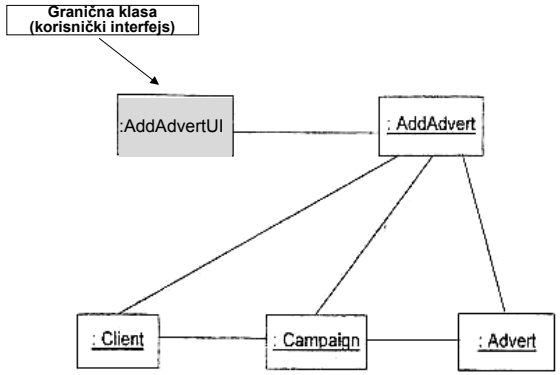
13

Granične klase:

- *Granične klase* modeluju interakciju između sistema i aktera. S obzirom da su deo modela zahteva, one su relativno apstraktne.
- Formiranjem stereotipa granične klase naglašeno je da je njihov glavni zadatak upravljanje transferom podataka preko granice sistema.

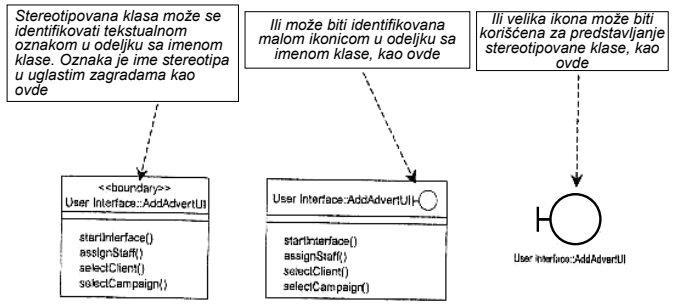
14

Primer:



15

Simboli korišćeni kod graničnih klasa:



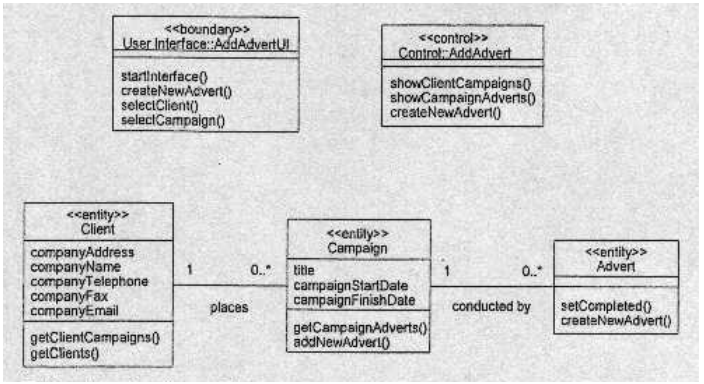
16

Klase entiteta:

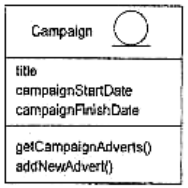
Klase entiteta koriste se za modelovanje informacije i pridruženog ponašanja nekog fenomena ili koncepta kao što je individualni, objekat iz stvarnog života, ili događaj iz stvarnog života.

- Kao opšte pravilo, klase entiteta predstavljaju nešto unutar aplikacionog domena, ali spoljašnje u odnosu na softverski sistem, o čemu sistem mora da pamti neku informaciju.
- Ova stvar može biti potpuno apstraktna, npr. kampanja, ili može biti konkretna, npr. član osoblja.

Primer - tipovi klasa:



Notacija (simboli) kod klasa entiteta:

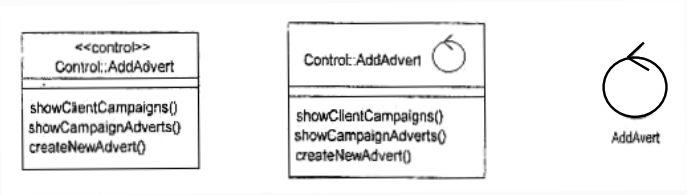


Kontrolne klase:

Kontrolne klase **predstavljaju koordiniranje, uređivanje, transakciju i kontrolu drugih objekata.**

Kontrolna klasa predstavlja aspekte izračunavanja i vremenskog raspoređivanja logike korisničkog slučaja – delove koji nisu specifični za ponašanje klase entiteta, a specifični su za korisnički slučaj.

Notacija (simboli) kod kontrolnih klasa:



21

Dijagrami klasa:

Relativna stabilnost klasa i uzoraka:

- **Klase entiteta obično predstavljaju permanentnije aspekte *aplikacionog domena*, dok granične i kontrolne klase predstavljaju relativno stabilne aspekte *načina kako softver radi.***
- **Suprotno, *uzorci objekata obično se često menjaju*, odražavajući potrebu sistema da održi aktuelnu sliku dinamičke okoline poslovanja.**

22

Relativna stabilnost uzoraka #1:

Uzorci su predmet kod tri glavna tipa promena u toku izvršavanje sistema.

1) Oni se kreiraju (npr. objekat *kampanja*)

2) Oni mogu biti uništeni (Posle kraja kampanje, svi podaci postaju nevažni i uništavaju se)

3) Objekti se mogu obnoviti (promena vrednosti na zapamćenu vrednost jedne ili više karakteristika).

23

Relativna stabilnost uzoraka #2:

Mnogo objekata entiteta su relativno dugovečni u odnosu na granične i kontrolne objekte, a neki se obnavljaju često u toku životnog veka.

Granični i kontrolni objekti mogu se često obnavljati, ali uglavnom ove promene prate tekuće promene stanja u toku izvršavanja softvera bez prostiranja promena na memorisanu informaciju o aplikacionom domenu.

24

Atributi:

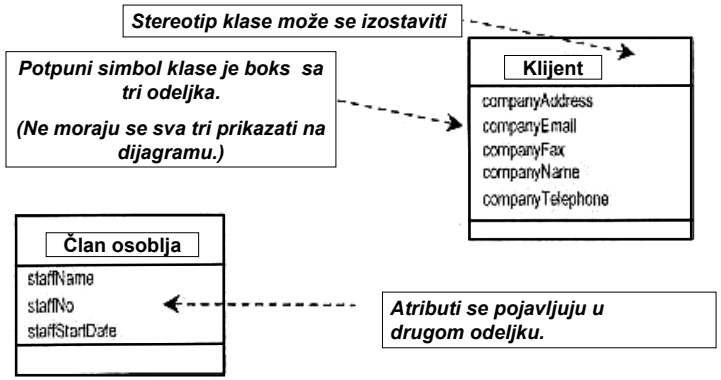
Atributi su deo neophodnog opisa klase.

Oni pripadaju klasi (za razliku od objekata koji su uzorci klase).

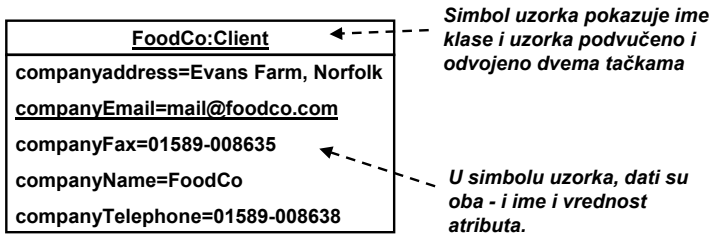
Atributi pojašnjavaju šta klasa može da 'zna'.

Svaki objekat ima sopstvenu, moguće i jedinstvenu, vrednost svakog atributa (ili vrednosti ako je atribut polje).

Atributi u dijagramu klase:



Vrednosti atributa u dijagramu uzorka:



Atributi i stanje:

Tekuće stanje objekta opisano je vrednostima atributa uzorka.

Kada se menja vrednost atributa, objekat može da promeni stanje.

Neke promene su značajne tj. utiču na ponašanje sistema (na način odgovora na događaje, a neke ne.

Značajne promene modeluju se dijagramima stanja.

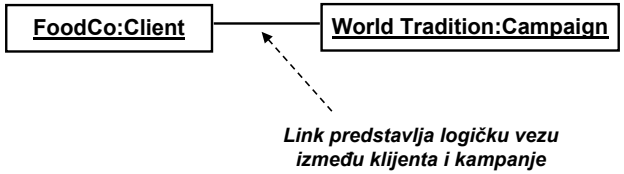
Link:

Link je logička veza između dva ili više objekata. (Npr. :Client i :Campaign, odnosno :StaffMember)

U analizi zahteva, ovo izražava logičku relaciju na *aplikacionom* nivou.

Linkovani uzorci mogu biti iz različitih klasa ili iz iste klase. Mada je neuobičajeno, link može da povezuje uzorak sa samim sobom (Npr. kapiten tima koji je istovremeno i igrač, selektuje sebe za člana ekipe.).

Link:



Asociranje:

Asocijacija povezuje dve klase.

Asocijacija opisuje skup sličnih linkova (kao što klasa opisuje skup sličnih uzoraka). Otuda definicija: linkovi su uzorci asocijacije.

Sličnost: Linkovi koji postoje, svi su između objekata asociranih klasa.

Asocijacija je veza između dve klase koja predstavlja mogućnost da njihovi uzorci učestvuju u linku.

Prepoznavanje asocijacija:

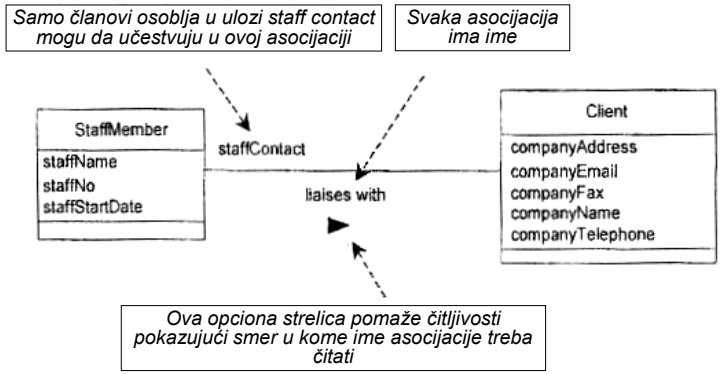
Neke asocijacije prepoznaju se bez uočavanja specificiranja linkova (npr. Client i Campaign).

Ostale asocijacije identifikuju se kroz postojanje linkova koji se modeluju na dijagramima saradnje, kao deo aktivnosi kod realizacije korisničkog slučaja.

Generalno pravilo:

Gdegod postoji link između dva objekta entiteta, postoji i odgovarajuća asocijacija između njihovih klasa.

Primer asocijacije:



Značaj asocijacija:

Dok link između dva objekta predstavlja vezu u stvarnom svetu, asocijacija između dve klase predstavlja *mogućnost* linka.

Saznanje da dve klase imaju asocijaciju ne govori nam koji uzorci su linkovani (ako su uopšte linkovani). Asocijacija je apstraktna i generalna, a ne posebna.

Asocijacije i stanje:

Stanje je definisano *tekućim vrednostima atributa* objekta.

Drugi aspekt stanja objekta definisan je *tekućim skupom linkova* ka ostalim objektima.

Kadgod se kreira ili ukine link ka drugom objektu, objekat menja stanje.

Višestukost (multiplicity):

Asocijacije predstavljaju moguće linkove između objekata.

***Višestrukost (multiplicity)* definiše granicu broja linkova sa objektima drugih specifičnih klasa u kojima može da učestvuje jedan objekat.**

Višestrukost odražava pravila preduzeća (ili posla) koja predstavljaju ograničenja za događanje poslovnih aktivnosti.

Primer višestrukosti #1:

Jedinstveni račun **može imati samo jednog korisnika (vlasnika),**

pridruženi račun **može imati tačno dva korisnika, dok**

poslovni račun **može imati dva ili više korisnika.**

Ova pravila moraju strogo biti ugrađena u sistem da bi sistem pravilno funkcionisao!

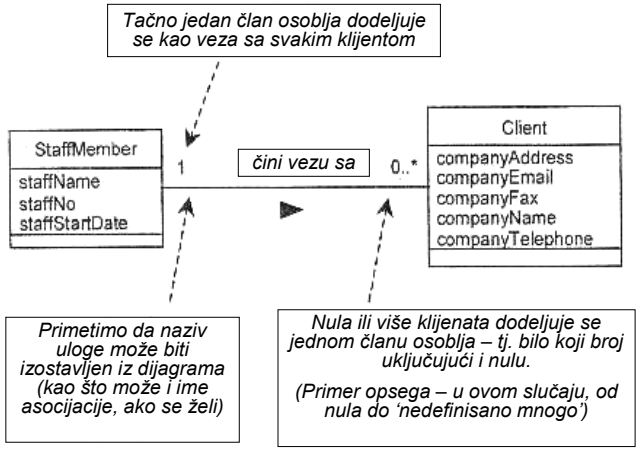
37

Zadavanje višestrukosti:

U modelu zahteva ne mora da stoji ime kontakt-osobe sa svakim klijentom (mogu i da se menjaju), ali model zahteva mora da nam pokaže varijetete načina linkovanja individualnih uzoraka i naročito ograničenja koja treba primeniti.

38

Primer višestrukosti #2:



Operacije #1:

Operacije su elementi zajedničkog ponašanja svih uzoraka u klasi. To su akcije koje se mogu izvršiti pomoću objekta ili na objektu.

Klase modelovane u toku analize zahteva predstavljaju stvari i koncepte iz stvarnog sveta. Njihove operacije predstavljaju aspekte ponašanja tih istih stvari i koncepata.

Operacije su aspekti ponašanja objekata neophodni za simuliranje načina rada aplikacionog domena.

Operacije su servisi čije obavljanje mogu od objekta zahtevati drugi objekti.

40

Operacije #2:

Operacija je specifikacija ponašanja klase sa nekog aspekta.

Operacije se definišu za celu klasu i važe za sve uzorke klase.

U toku modelovanja i analize zahteva *nije neophodno razmatrati detaljno kako radi svaka operacija*.

***Neophodno je znati koje operacije treba uključiti u model zahteva* i gde je odgovarajuća lokacija operacije unutar klase.**

41

Notacija za operacije:

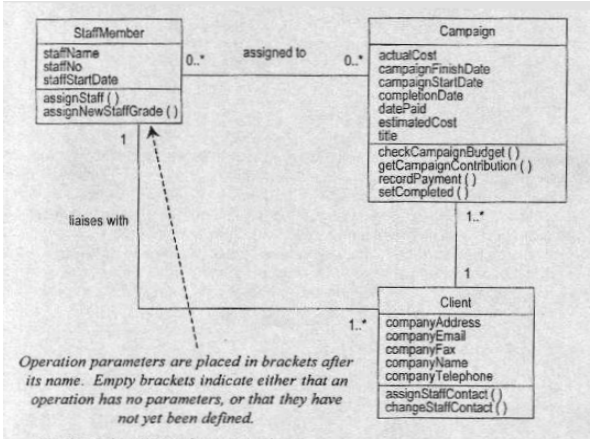
Imena operacija smeštaju se u treći odeljak pravougaonog simbola klase.

Imena operacija pišu se malim slovima.

Ne postoji odvojena notacija za prikazivanje operacija jednog uzorka objekta – operacije imaju potpuno isto značenje za uzorke i za klase (za razliku od atributa i asocijacija).

42

Primer – parcijalni dijagram klasa za sistem Agate sa nekim atributima i operacijama:



43

Operacija i metod:

Operacije se izvršavaju pomoću *metoda,* a šta u stvari metod radi u datim okolnostima može biti ograničeno vrednostima atributa objekta i linkova kada se metod pozove.

Efekti koje može imati jedna operacija mogu da budu promene karakteristika objekta inovirajući vrednosti atributa.

Efekat operacije može biti promena linkova objekta.

44

Operacije i stanje:

Postoji relacija između operacija objekta i njegovih stanja.

Objekat može promeniti svoje stanje samo izvršavanjem neke operacije.

Servisu koji jedan objekat obezbeđuje, može se pristupiti samo kroz interfejs objekta, a on se sastoji od oznake operacije definisane na nivou klase.

Da bi jedan objekat uradio bilo šta - promenio svoje podatke, kreirao ili uništio linkove, odgovorio na jednostavne upite, drugi objekat mora poslati poruku koja sadrži ispravni poziv njegove operacije.

45

Crtanje dijagrama klasa:

U praksi, jedan projekat mnogo se razlikuje od drugog i neki koraci mogu biti izostavljeni ili izvedeni na sasvim drugačijem stepenu životnog ciklusa.

Iskusni analitičari će uvek koristiti sopstveno prosuđivanje kako nastaviti u datoj situaciji.

46

Postupak razvijanja dijagrama klasa:

Polazeći od korisničkog slučaja, preko dijagrama saradnje (collaboration), preko dalje analize, dijagram klasa razvija se za svaki korisnički slučaj i razni dijagrami klasa korisničkiog slučaja obično se zatim spajaju u veće dijagrame klasa analize. Ovo se prvo crta za jedan podsistem, ali dijagrami klasa mogu se crtati na bilo kome nivou koji je pogodan, od jednog uzorka korisničkog slučaja do velikih kompleksnih sistema.

47

Identifikovanje klasa:

Dijagram klasa predstavlja osnovu OO analize.

Kroz sukcesivne iteracije, obezbeđuje i osnovu visokog nivoa za arhitekturu sistema i osnovu niskog nivoa za dodelu podataka i ponašanja individualnim klasama i uzorcima objekata, kao i osnovu za projektovanje koda koji implementira sistem.

Vrlo je važno identifikovati klase korektno. Međutim, s obzirom na iterativnu prirodu OO pristupa, nije obavezno postići sve ovo u prvom pokušaju.

48

Identifikovanje objekata i klasa :

- **Najbolji način za nalaženja objekata je razmatranje korisničkih slučajeva i interakcije.**
- **Ponekad je korisno razviti prvo *model domena* – model klase za analizu koji je nezavisan od bilo koga korisničkog slučaja.**
- **Vredi pregledati dokumentaciju o pozadini sistema koja je prikupljena u toku faze prikupljanja činjenica. Posle drugog čitanja mogu se otkriti nove klase zbog boljeg razumevanja sistema.**

49

Identifikovanje objekata i klasa :

- **Uključiti predstavnike korisnika i oni će identifikovati brojne nove klase.**
- **Sopstvena intuicija je koristan izvor, kao i intuicija kolega. Intuiciju proveriti sa nekim ko je vrlo blizak sa poslom koji se razvija.**
- **Dobra taktika može biti identifikacija šta ne može biti objekat i šta ne može biti klasa.**

50

Identifikovanje objekata i klasa :

Za svaki pojam u listi, postaviti sledeća pitanja kao pomoć u određivanju klasa:

- ***Da li je to izvan delokruga sistema?***

  Možda su uključeni ljudi, stvari i koncepti koji nisu neophodni za opis domena aplikacije. Njih ukloniti sa liste

- ***Da li se to odnosi na ceo sistem?***

Obično nije neophodno za model da sadrži klase koje predstavljaju ceo sistem.

51

Identifikovanje objekata i klasa :

- ***Da li to duplira neku drugu klasu?***

Ovo će postati jasnije kad se napišuopisi za druge klase.

- ***Da li je to suviše nejasno?***

Eliminisati sve potencijalne klase za koje se ne može napisati jasan opis, osim ako se zna da je to zbog trenutnog nedostatka informacija.

- ***Da li je to suviše specifično?***

Sve do modelovanja specifične interakcije, korisno je modelovati klase, a ne uzorke.

52

Identifikovanje objekata i klasa :

•*Da li je to suviše vezano za fizičke ulaze i izlaze?*

Trebalo bi izbeći stvari koje suviše zavise od fizičkog načina rukovanja sistemom.

•*Da li je to u stvari atribut?*

•Atribut je karakteristika klase. Problem se može javiti tako da pojam predstavlja atribut u jednom domenu, a klasu u drugom, u zavisnosti od zahteva. Dakle, neki pojmovi sa potencijalne liste mogli bi da se bolje modeluju kao atributi.

53

Identifikovanje objekata i klasa :

•*Da li je to stvarno operacija?*

Operacija je akcija, odgovornost klase. To može biti zbunjujuće jer neke akcije se mogu bolje modelovati kao klase.

•*Da li je to stvarno asocijacija?*

Asocijacija je vrsta odnosa između dve klase. Zbunjujuća činjenica je da se neki odnosi mogu predstaviti kao klase. Ipak, najvažnije je da su svi značajni odnosi modelovani, ili kao klase ili kao asocijacije.

54

Dodavanje i lociranje atributa:

Mnogi atributi javljaju se već u opisima korisničkih slučajeva. Ostali se pojave u toku detaljnijeg razmišljanja o modelu. Jednostavno pravilo je da atributi moraju da budu smešteni unutar klase koju opisuju.

Ponekad je teže identifikovati korektnu klasu za neki atribut. Atribut ne može da pripadne na pogodan način ni jednoj klasi koja je već identifikovana.

55

Dodavanje asocijacija:

Asocijacije treba naći razmatranjem logičkih odnosa između klasa u modelu. Asocijacije se mogu naći u opisima korisničkih slučajeva i drugim tekstualnim opisima domena aplikacije, kao glagoli koji izražavaju permanentne ili trajne odnose.

Puno razumevanje asocijacija u modelu klasa može se postići kasnije analiziranjem interakcije između različitih kasa.

56

Određivanje višestrukosti:

S obzirom da višestrukost asocijacija predstavlja ograničenje načinom kako korisnici izvode njihove poslovne aktivnosti, važno je ovo izvesti ispravno, a jedini način da se ovo uradi je da se korisnici pitaju o svakoj asocijaciji. Analitičar treba da proveri i šta piše u dokumentima.

57

Nalaženje operacija:

Operacija se može zamisliti kao mali doprinos jedne klase postizanju većeg cilja predstavljenog celim korisničkim slučajem.

One se nekad mogu naći kao glagoli akcije u opisima korisničkih slučajeva, ali ova slika može ostati nekompletna sve dok interakcija između klasa ne bude shvaćena dublje.

58

Preliminarno dodeljivanje operacija:

Pri odlučivanju u koju klasu smestiti koju operaciju, zadovoljavajućće rešenje može se dobiti prateći dve vodilje:

1. **Zamisliti svaku klasu kao nezavisnog glumca odgovornog za izvođenje ili znanje nekih stvari. Na primer, možemo da pitamo 'Šta član osoblja treba da zna ili da može da uradi u ovom sistemu?**
2. **Smestiti svaku operaciju u istu klasu u kojoj podaci to zahtevaju. međutim, ovo je često problematično jer možda nisu još uvek identifikovani svi atributi.**

Generalno, ne treba očekivati da se sve uradi ispravno u prvom pokušaju.

59

CRC (Class Responsibility Collaboration) kartice #1:

***Odgovornost (responsibility)* je opis visokog nivoa onoga što klasa može da radi.**

Ona pokazuje znanje ili informaciju koja je dostupna toj klasi, ili smeštena unutar njenih sopstvenih atributa ili zahtevana preko saradnje sa ostalim klasama, a takođe i servise koje mogu da ponude objektima.

60

CRC (Class Responsibility Collaboration) kartice #2:

Kartice saradnje odgovornosti klase (Class Responsibility Collaboration, CRC) predstavljaju efektivnu tehniku za traženje mogućih puteva dodele odgovornosti klasama i saradnje neophodne za ispunjenje odgovornosti.

CRC kartice mogu se koristiti na nekoliko stepena projekta za različite ciljeve.

61

Format CRC kartice:

Ime klase:	
Odgovornosti	**Saradnje**
U ovom delu listaju se odgovornosti klase.	***Ovde se listaju saradnje sa drugim klasama, zajedno sa kratkim opisom cilja saradnje.***

62

Upotreba CRC kartica #1:

- **Identifikovati koji objekti su uključeni u korisnički slučaj.**
- **Dodeliti svaki objekat članu tima koji će igrati ulogu tog objekta.**
- **Odglumiti korisnički slučaj. To podrazumeva i pogodbe među objektima (odigrane od strane članova tima) da bi se utvrdilo kako se može dodeliti odgovornost i identifikovalo kako objekti mogu da sarađuju jedni sa drugima**
- **Identifikovati i zabeležiti nedostajuće ili preopširne objekte.**

63

Upotreba CRC kartica #2:

Taktika učesnika u igri mora da bude odgovarajuća distribucija odgovornosti među klasama. Jedno pravilo je da se svaki objekat (član tima koji ga igra) ponaša kao maksimalno lenja osoba nalazeći koji objekat bi mogao da prihvati odgovornost umesto njega. Tako se tek ubeđivanjem zasnovanim na racionalnim argumentima prihvata odgovornost.

Cilj je smanjiti broj poruka koje moraju biti prosleđene, smanjiti kompleksnost, a istovremeno napraviti definicije klasa kje su koherentne i dobro fokusirane.

64

Kratak pregled:

- **Cilj: formirati inicijalni dijagram klasa za model zahteva i slediti ga procesom realizacije korisničkih slučajeva.**
- **Važni elementi modela analize na ovom stepenu: klase, sa atributima i operacijama, i asocijacije koje pokazuju odnose između klasa**
- **Model prkazuje glavne funkcionalne zahteve sistema pomoću odgovornosti za obezbeđivanje servisa.**
- **Definisana je logička arhitektura koja je osnova rada na projektu nadalje.**

65