UNIVERZITET
METROPOLITAN
BEOGRAD

SE311 - PROJEKTOVANJE I ARHITEKTURA SOFTVERA

Stilovi softverskih modula

Lekcija 03

PRIRUČNIK ZA STUDENTE

SE311 - PROJEKTOVANJE I ARHITEKTURA SOFTVERA

Lekcija 03

STILOVI SOFTVERSKIH MODULA

UVOD

Različiti načini dokumentovanja strukture modula određenog softverskog sistema omogućavaju prikaz implementacionih jedinica u vidu modula.

Softverska dokumentacija opisuje glavne jedinice za implementaciju (module sistema) zajedno sa vezama između navedenih jedinica. Pogledi na softversku arhitekturu omogućavaju izvršavanje nekih od mogućih ciljeva kao što su: obrazovanje (budućih članova razvojnog tima, korisnika, naručioca softvera), komunikacija (između različitih korisnika sistema), osnova za izgradnju i analizu. Način na koji se vrši podela softvera na jedinice predstavlja jedan od važnih oblika formiranja sistemske strukture. Moguće je podeliti sistem na jedinice zatim izvršiti opis svake jedinice, uloge te jedinice u sistemu kao i mogućnosti koje mogu biti unapređene i poboljšane. Takođe, omogućen je i prikaz kompletnog sistema i svih modula koji čine sistem.
Struktura modula određuju kako promene jednog modula mogu uticati na ostale delove sistema a samim tim i na mogućnost modifikacije, prenosivost i ponovnu upotrebu sistema.

Dokumentacija softverske arhitekture bez bar jednog pogleda modula ne smatra se kompletnom dokumentacijom i kao takva je često neupotrebljiva.

Ova lekcija vam obezbeđuje sledeće ishode učenja:

- Razumevanje i primena stilova softverskih modula, pregled elemenata, relacija i svojstva pogleda modula. Kombinovanje pogleda modula sa drugim pogledima na softversku arhitekturu, notacije unutar pogleda modula.
- Razumevanje i primena stila razlaganja, pregled elemenata, relacija i svojstva stila razlaganja. Upotreba stila razlaganja na konkretnom primeru i kombinovanje sa drugim stilovima softverske arhitekture.
- Razumevanje i primena stila upotrebe, stila generalizacije, stila slojeva i stila aspekta. Kombinovanje navedenih stilova softverske arhitekture sa drugim stilovima softverske arhitekture, razumevanje notacija prikazivanja različitih stilova i pregled elemenata, relacija i svojstava.
- Predstavljanje softverske arhitekture kroz model podataka poštujući definisanu notaciju, elemente i relacije modela.

UVODNI VIDEO LEKCIJE

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 1

Stilovi modula softverske arhitekture

ELEMENTI, RELACIJE I SVOJSTVA STILOVA MODULA

Različiti stilovi modula omogućavaju formiranje različitih skupova modula i njihovih međusobnih odnosa.

Projektanti softvera koriste izraz *modul* za objašnjavanje softverske strukture uključujući jedinice programskog jezika kao što su: C programi, Java ili C# klase, PL/SQL procedure ili opšte grupacije programskog koda kao što su Java paketi. Modulima, kao što je već navedeno, moguće je dodeliti svojstva, definisati odgovornost u softveru i analizirati karakteristike. *Moduli se mogu dodatno razložiti na nove module ili spojiti u jedan modul.* Slojeviti stil identifikuje module i povezuje ih na osnovu relacije koja dozvoljava takav način korišćenja (allowed-to-use-relation) dok stil generalizacije identifikuje module na osnovu zajedničkih osobina.

Relacije (veze)

Stilovi modula imaju sledeće vrste relacija:

- **je deo od ("**is part of**"):** definiše deo ili celu relaciju između podmodula-određenog dela-modula agregacije-celog modula. Najopštiji primer ove relacije prikazuje agregaciju sa različitim opisom.
- **zavisi od ("**depends on**"):** A zavisi od B, definiše relaciju u kojoj A zavisi od B. Različiti tipovi veze "zavisi od" mogu biti korišćeni u pogledu modula. Koriste se: koristi ("uses") dozvoljava korišćenje ("allowed to use"), unakrsni rez ("crosscuts") i relacije entiteta podataka.
- **je ("**is a**"):** relacija koja predstavlja generalizaciju između više specifičnih modula, deteta modula ("child") i roditelja modula ("parent"). Korišćenjem ove relacije, modul dete se može koristiti u kontekstu u kome se koristi modul roditelj. Objektno-orijentisano nasleđivanje i realizacija interfejsa predstavljaju posebne slučajeva relacije je ("is a").

SVOJSTVA STILOVA MODULA

Svojstva modula pomažu prilikom procesa implementacije ili analize za prikazivanje određenog modula.

Lista svojstava određenog modula može da varira ali uglavnom sadrži sledeće:

- **Ime:** Ime modula predstavlja osnovnu informaciju za svaki modul softvera. Ime može biti povezano sa ulogom modula u softveru, ili pozicijom na kojoj se nalazi unutar strukture (primer: "A.B.C" se odnosi na modul C koji je podmodul modula B a koji je sam podmodul modula A)
- **Odgovornost:** Svojstvo odgovornost modula predstavlja način identifikovanja uloge modula u softveru i uspostavljanje identiteta modula pored njegovog imena. Odgovornost mora biti opisana sa dovoljno detalja kako bi čitaocu bilo jasno šta koji modul unutar softvera radi.
- **Vidljivost interfejsa:** U slučaju da modul sadrži određene podmodule, interfejsi podmodula mogu imati svoje namene, tako da interfejs podmodula može biti nezavisan od interfejsa modula kome pripada. Za interfejse podmodula koji imaju direktnu vezu sa interfejsima modula kome pripadaju ("roditeljskim interfejsom") mogu se koristiti različite strategije kao što je recimo enkapsulacija prikazana na slici 1. U prikazanom slučaju roditeljski modul pruža sopstvene interfejse i mapira sve zahteve prema mogućnostima podmodula. U slučaju da je modul C roditelj modulima A i B, interfejs C će biti podskup interfejsa modula A i B.

• **Informacije o implementaciji:** Moduli predstavljaju jedinice implementacije i potrebno je navesti informacije u vezi njihove implementacije sa stanovišta upravljanja razvojem navedenih jedinica i razvojem softvera koji ih sadrži. Ovakve informacije potrebno je uneti u okviru dokumentacije softverske arhitekture. Informacije o implementacije mogu sadržati:
- Mapiranje jedinica programskog koda , definiše datoteke koje služe za implementaciju modula. Primer može biti: modul Account, ukoliko se implementira u Javi može imati nekoliko datoteka potrebnih za implementaciju: IAccount.java (interfejs), AccountImpl.java (implementacija Account funkcionalnosti) AccountBean.java (klasa koja beleži stanje računa u memoriji) i recimo klasu za testiranje AccountTest.java.
- Informacije za testiranje . Test plan modula, test slučajevi, podaci o testiranju koje je potrebno skladištiti.
- Informacije o upravljanju . Informacije o predviđenom rasporedu i planiranom budžetu za izradu modula.
- Ograničenja implementacije . U većini slučajeva projektant softvera ima definisanu strategiju implementacione faze u slučaju određenih modula i mogućih ograničenja. Ograničenje vezano za modul je privatno i ne prikazuje se van modula i dokumentacije softverske arhitekture navedenog modula

KOJI STILOVI MODULA SE KORISTE

Stilovi modula svoju primenu imaju u različitim fazama projektovanja softverske arhitekture.

Različiti stilovi modula koriste se u fazama:

- Konstrukcije**.** Stil modula može omogućiti plan programskog koda kao i plan za skladištenje podataka.
- Analize**.** Dve važne tehnike analize su praćenje zahteva i analiza uticaja. Vrši se utvrđivanje na koji način su kroz module ispunjeni korisnički zahtevi. Funkcionalni zahtevi takođe mogu biti ispunjeni nizom poziva između modula. Dokumentovanjem poziva

prikazuje se na koji način sistem ispunjava zahteve. Izvršavanjem analize uticaja kreira se plan modifikacije sistema.

- Komunikacije**.** Stilovi modula mogu se koristiti za objašnjenje funkcionalnosti sistema nekome ko nije upoznat sa sistemom. Takođe, moguće je prikazati strukturu programskog koda novom članu razvojnog tima zaduženom za razvoj softvera ili odrediti način održavanja softvera i zadatke članu razvojnog tima koji je zadužen za održavanje softvera.

Jedan od nedostataka stila modula softvera je nemogućnost identifikovanja ponašanja u toku rada jer se moduli prikazuju kao statični delovi softvera. Shodno tome, stilovi modula ne mogu biti korišćeni za analizu performansi ili pouzdanosti softvera pa je u tom slučaju potrebno kombinovati različite stilove softverske arhitekture.

NOTACIJE ZA STILOVE SOFTVERSKIH MODULA

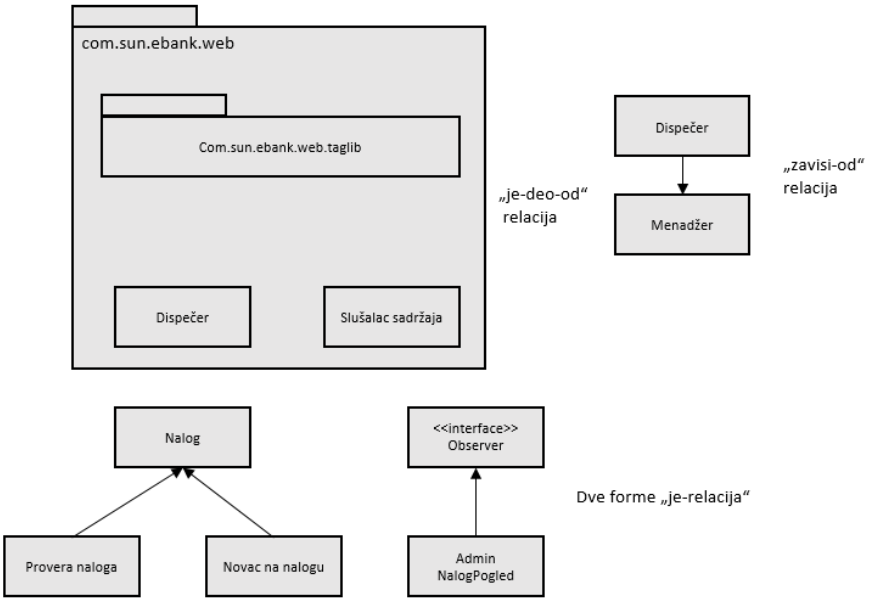
Predstavljanje stilova modula kroz neformalnu notaciju sadrži nekoliko različitih načina.

Neformalne notacije

Jedan od načina predstavljanja je prikaz modula u vidu kutija koje su povezane različitim linijama koje predstavljaju relacije između modula. U upotrebi je i gnezdo ("nest") za prikaz agregacije a strelice prikazuju zavisni odnos. Takođe, može se koristiti i simbol za interfejs koji je prikazan na slici u vidu linije i zatvorenog kruga na kraju linije.

Jedinstveni jezik za modelovanje

Primena UML jezika za predstavljanje modula prikazana je na slici 1.



Slika 1.1 Predstavljanje modula sistema korišćenjem UML jezika [Izvor: Clements [2]]

Drugi način neformalne notacije je tekstualna lista modula sa opisima odgovornosti za svaki modul posebno kao i uz upotrebu ključnih reči za svaki modul.

Korišćenjem UML jezika za prikaz modula omogućena je konstrukcija klasa kroz objektno-orijentisanu specifikaciju modula. Tako konstruisani moduli su generički, nisu specifični za određenu tehnologiju implementacije, platformu ili programski jezik.

Matrica strukture zavisnosti

Matrica strukture zavisnosti predstavlja tabelu koja prikazuje module kao kolonu i redove a zavisnosti kao ćelije tabele. Navedena matrica se kreira sa jednakim brojem kolona i redova i može se razviti kroz određene dostupne alate. Takođe, matricu je moguće kreirati iz programskog koda korišćenjem metoda obrnutog inženjeringa.

Dijagram odnosa entiteta

Dijagram odnosa entiteta koristi se za modelovanje podataka i prikazuje informacije o entitetima koji su bitni za implementaciju u sistemu. Relacije koje se koriste u ovom dijagramu su jedan na jedan, jedan na više i više na više.

ARCHITECTURAL STYLES (VIDEO)

Trajanje: 1:26 minuta.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Stil razlaganja softverske arhitekture

PREGLED STILA RAZLAGANJA (DEKOMPOZICIJA)

Uzimanjem elemenata i svojstava kroz pogled modula uz fokus na relaciju "je deo od" dobijamo stil razlaganja (dekompoziciju).

Stil razlaganja opisuje strukturu programskog koda u vidu modula i podmodula i podelu sistemskih odgovornosti na njih. Kriterijum za upotrebu stila razlaganja za podelu modula na manje module (podmodule) uključuje:

- **Dostizanje određenih atributa kvaliteta**. Da bi bila omogućena izmenljivost softvera potrebno je izvršiti razlaganje na manje module kako bi proces izmene bio jednostavniji i uključivao što manje delova sistema.
- **Odluke o razvijanju umesto kupovine**. Određeni moduli mogu se kupiti, preuzeti sa interneta kao softver otvorenog koda ili preuzeti iz nekog prethodno završenog projekta. Tako preuzeti moduli imaju definisane odgovornosti u sistemu iz koga su preuzeti pa je potrebno izvršiti dodatno razlaganje modula i prilagođavanje sistemu u kome će biti korišćen.
- **Implementacija softverske proizvodne linije**. U svrhu podržavanja efikasne implementacije softverskog proizvoda i familije softverskih proizvoda (koji se razvijaju u okviru jednog razvojnog tima) bitno je napraviti razliku između zajedničkih modula koji se koriste u svakom ili delimično svakom proizvodu tog razvojnog tima i varijabilnih modula koji se modifikuju shodno primeni u različitim proizvodima.
- **Dodeljivanje tima**. Paralaleno korišćenje određenog modula za sprovođenje različitih uloga u okviru softverskog proizvoda zahteva razdvajanje modula i dodeljivanje različitim razvojnim timovima. Na taj način, modul će biti korišćen u istom proizvodu ali za različite operacije i različiti timovi će voditi proces implementacije i kasnijeg održavanja.

ELEMENTI, RELACIJE I SVOJSTVA STILA RAZLAGANJA

Elementi stila razlaganja (dekompozicije) su moduli.

Određeni moduli mogu sadržati druge pod module i nazivaju se podsistemi. Osnovna relacija ovog stila je "je deo od" relacija koja prikazuje da element može biti deo drugog elementa. Razlaganje modula može omogućiti vidljivost pod modula samou okviru modula u kome se nalaze ("roditelj" modula) ili vidljivost svim ostalim modulima sistema. Za potrebe grafičkog prikaza vidljivosti pod modula mogu se koristiti određene grafičke prezentacije stila kao što je prethodno prikazano.

Pregled	**Stil razlaganja se koristi za podelu sistema na jedinice implementacije. Pregled razlaganja opisuje organizaciju programskog koda u vidu modlua i podmodula i prikazuje kako su odgovornosti i funkcionalnosti podeljene na njih.**
Elementi	**Moduli**
Relacije	**„je deo od“ („is part of“)**
Ograničenja	**U prikazu stila razlaganja nije dozvoljena upotreba petlji. Modul može imati samo jedan „roditelj“ modul na grafičkom prikazu.**
Upotreba	**Podela sistema na module omogućava strukturu softvera u vidu sitnih komada koji međusobno komuniciraju. Omogućava ulaz za davanje radnih zadataka članovima razvojnog tima. Vrši lokalizaciju izmena samo na određeni modul softvera.**

Slika 2.1 Prikaz stila razlaganja [Izvor: Clements [2]]

UPOTREBA STILA RAZLAGANJA

Pregled stila razlaganja je pogodan za proces učenja o sistemu za nove članove razvojnog tima.

Novi članovi se mogu fokusirati samo na određeni deo sistema od koga žele da krenu i zatim pratiti veze sa ostalim identifikovanim modulima sistema. Ne moraju znati kompletan sistem ili sve funkcionalnosti sistema da bi mogli da rade na implementaciji ili modifikaciji određenog modula. Takođe, iz ugla projektanta softvera pogodan je za proces projektovanja i navigacije kroz kompletan projekat. Projektant softvera može izvršiti podelu radnih zadataka članovima tima po modulima dobijenim u stilu razlaganja.

NOTACIJA STILA RAZLAGANJA

Razlaganje može biti prikazano tako što se navede naziv modula a zatim prikaže njegov deo.

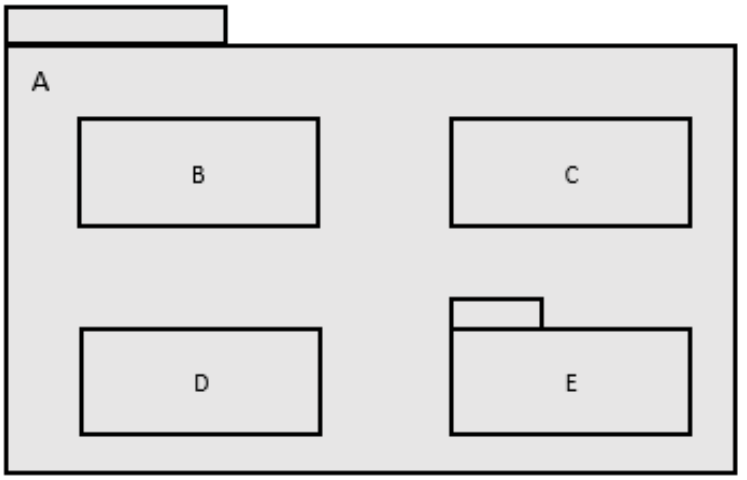
Moduli identifikovani kroz stil razlaganja prikazuju se kao kutije koje sadrže druge kutije. Ukoliko je potrebno prikazati da moduli "dece" nisu vidljiva iznad modula "roditelja" koristi se vizuelna granica u prikazu modula koja to označava. Takođe, pored grafičkog prikaza potrebno je i detaljno tekstualnim opisom navesti na koji način je izvršeno razlaganje.

UML

Prikaz stila razlaganja kroz UML moguć je upotrebom paketa za predstavljanje modula. Paket može sadržati klase, druge pakete (podmodule).

1. Moduli mogu biti ugnježdeni kao na slici 2.

2. Moduli mogu biti prikazani kroz dva dijagrama gde je na prvom dijagramu prikazan modul a na drugom sadržaj modula. Ovakva grafička prezentacija stila razlaganja prikazana je u pokaznom primeru koji sledi.



Slika 2.2 Prikaz ugnježdavanja modula kroz UML [Izvor: Clements [2]]

Druga svojstva modula prikazuju se tekstualno i omogućavaju dodatne informacije o tipu modula. Veza sa drugim stilovima Poželjno je kombinovati stil razlaganja sa pogledom komponenta i konektor. Modul može implementirati sve ili određeni deo komponenti i konektora. Jedna komponenta može zahtevi više različitih modula u procesu implementacije. Projektant softvera ima prikaz kako se struktura implementacije softvera mapiraju na strukture izvršavanja.

POKAZNI PRIMER: STIL RAZLAGANJA (DEKOMPOZICIJE)

Prikazana je primena stila razlaganja na primeru sistema ATIA-M.

Sistem ATIA-M (Army Training Information Architecture-Migrated) predstavlja veb aplikaciju u pisanu u Java EE koja služi za obuke američke vojske. Desktop aplikacija razvijena je korišćenjem .NET (C#) koji komunicira preko veb servisa sa komponentama serverske aplikacije (pisane u Java EE). Na slici 4 prikazan je najviši nivo stila razlaganja ATIA sistema. Programski kod razložen je u tri velika modula:

- Windows apps - koji sadrži tri podmodula :
  -Training and Doctrine Development Tool (TDDT)
  - Unit Training Management Configuration (UTMC)
  - zaseban podmodul koji koriste različite Windows aplikacije

Moduli TDDT i UTMC predstavljaju su dve odvojene Windows aplikacije koje su planirane u procesu projektovanja softvera, ukoliko se javi potreba za dodatnim aplikacija ima moguće ih je lako povezati sa sistemom. Takođe, tu su:

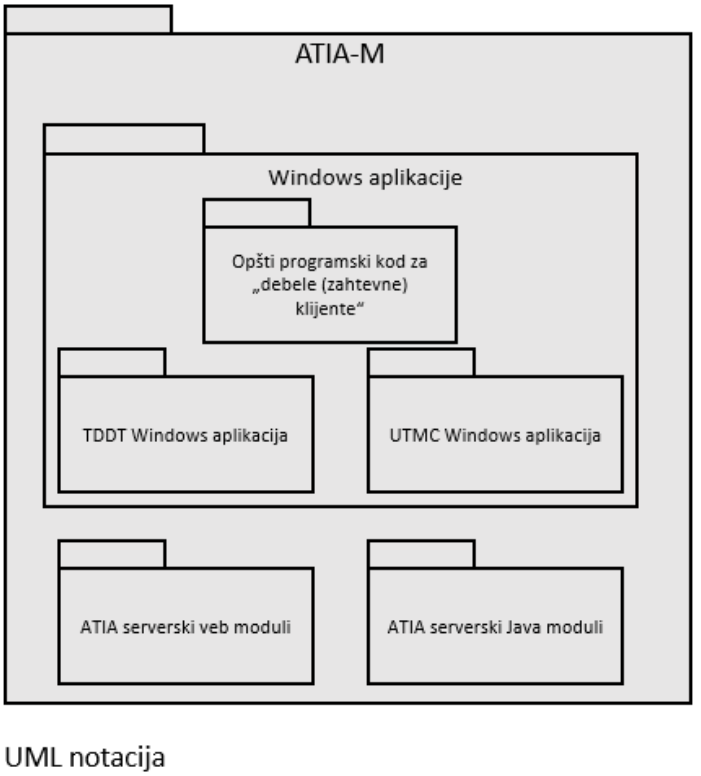
- ATIA veb modul na serverskoj strani - sadrže sve ne-Java module koji su raspoređeni na serverskoj mašini. Veb moduli uključuju datoteke JavaServer Pages (JSP), JavaScript kao i HTML strane.

- ATIA Java moduli na serverskoj strani - sadrži kompletan Java programski kod koji se pokreće na serveru. Ovaj modul ne uključuje JSP, JavaScript ili HTML.

POKAZNI PRIMER: STIL RAZLAGANJA (DEKOMPOZICIJA) - DIJAGRAMI ARHITEKTURE - NAJVIŠI NIVO STILA RAZLAGANJA

Prikazana je upotreba najvišeg nivoa stila razlaganja na primeru sistema ATIA-M sistema.

Primena stila razlaganja na modul "Windows apps" prikazana je na slici 3.



Slika 2.3 Najviši nivo stila razlaganja ATIA sistema [Izvor: Clements [2]]

Izvršena je dekompozicija modula najvišeg nivoa za kompletan ATIA-M sistem. Programski kod je podeljen u tri velika modula:

Windows aplikacija koja sadrži programski kod "debelih klijenata". Tri podmodula odgovaraju trening razvojnom alatu (TDDT), jedinicom za upravljanje obuke (UTMC) i posebnom modulu sa zajedničkim programskim kodom koji koriste različite Windows aplikacije (na slici "Opšti programski kod".

Trening razvojni alat i jedinica za upravljanje obuke predstavljaju dve Windows aplikacije koje su planirane ali je moguće dodati i nove aplikacije.

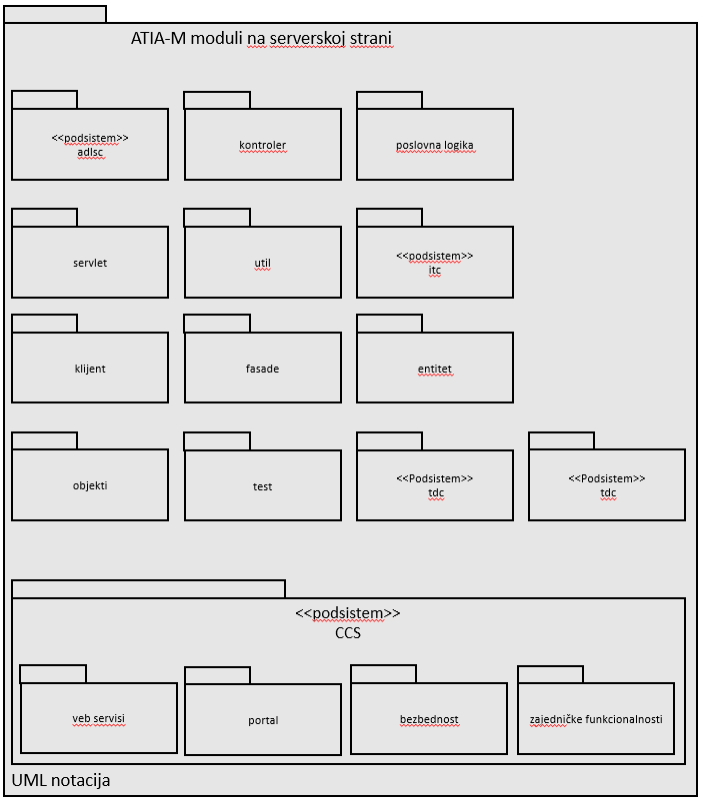
Veb moduli predstavljeni na ATIA serverskoj strani sadrže sve module koje nisu Java i koji trebaju da se nalaze na serverskoj strani. To su: JavaServerPages fajlove, JavaScript i HTML kod kao i dodatne aplete.

ATIA serverski Java moduli sadrže sav programski kod koji je potrebno pokrenuti na aplikacijskim serverima. U ovom modulu se ne nalaze JSP, JavaScript, HTML kod ili kod "debelog" klijenta.

POKAZNI PRIMER: STIL RAZLAGANJA (DEKOMPOZICIJA) - DIJAGRAMI ARHITEKTURE - RAZLAGANJE JAVA MODULA NA SERVERSKOJ STRANI

Prikazana je upotreba razlaganja Java modula na serverskoj strani ATIA-M sistema.

Razlaganje ATIA Java modula na serverskoj strani prikazana je na slici 4.



Slika 2.4 Prikaz razlaganja Java modula na serverskoj strani [Izvor: Clements [2]]

Na slici 4 prikazan je proces razlaganja Java modula ATIA-M sistema na serverskoj strani. Modul se sastoji iz podsistema CCS u kome se nalaze veb servisi potrebni za rad veb portala, bezbednosna komponenta koja služi za zaštitu podataka korisnika na portalu kao i dodatne zajedničke funkcionalnosti koje se koriste u okviru podsistema.

Ostatak identifikovanog modula sastoji se iz četiri podsistema koja su potrebna za funkcionisanje kompletnog modula i različitih komponenata. Komponenta kontroler, klijent,

objekti, test i servlet služe kao podrška navedenim podsistemima dok se u komponenti poslovna logika nalazi kompletan programski kod neophodan za neometan rad modula. Modul, kao što je prethodno objašnjeno, sadrži kompletan programski kod sa koji se pokreće na aplikacijskim serverima ATIA-M sistema.

Poglavlje 3

Stil upotrebe softverske arhitekture

PREGLED STILA UPOTREBE

Stil upotrebe koristi relaciju "zavisi od" koja je specijalizovana za upotrebu

Modul unutar stila upotrebe "koristi" drugi modul ako njegova ispravnost zavisi od ispravnosti drugog modula. Prethodno opisani stil razlaganja pokazuje samo strukturu implementacionih jedinica u smislu modula i podmodula, stil upotrebe ide korak dalje i otkriva koji modul može koristiti drugi modul unutar sistema. Kroz ovaj stil moguće je prikazati programerima koji vrše implementaciju softvera koji drug modul mora postojati u sistemu kako bi modul na kome rade pravilno funkcionisao. Stil upotrebe omogućava inkrementalni razvoj i postavljanje podskupova celih sistema.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ELEMENTI, RELACIJE I SVOJSTVA STILA UPOTREBE

Elementi stila upotrebe su moduli.

Elementi ovog stila su moduli, relacija koja se koristi je "zavisi od" pri čemu jedan modul zahteva tačno izvršavanje drugog modula za njegovo pravilno funkcionisanje.

Pregled	Stil upotrebe prikazuje na koji način moduli zavise jedan od drugog, pomaže u planiranju i definisanju podskupova sistema koji se razvija
Elementi	Moduli
Relacije	„zavisi od“ („depends on“)
Ograničenja	Stil upotrebe nema topoloških ograničenja ali je potrebno pravilno koristiti petlje ili zavisnosti kako bi se održao inkrementalni prikaz.
Upotreba	Planiranje inkrementalnog procesa razvoja softvera Debagovanje i testiranje Utvrđivanje efekata promene

Slika 3.1 Prikaz stila upotrebe [Izvor: Clements [2]]

UPOTREBA STILA UPOTREBE

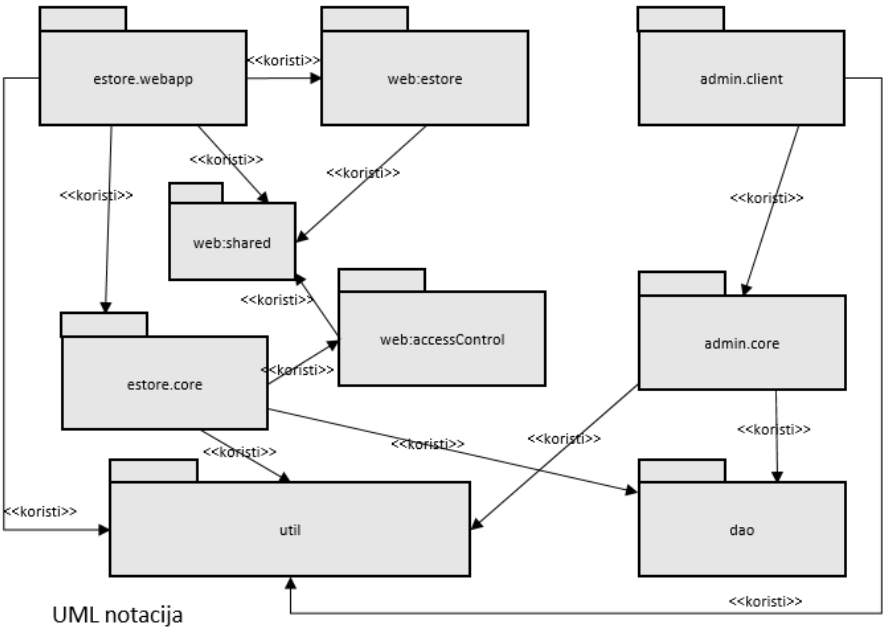
Stil upotrebe je koristan za planiranje inkrementalnog razvoja, proširenje sistema, debagovanje i testiranje.

Na slici 2 nalazi se osnovni prikaz stila upotrebe kao i način na koji pomaže u inkrementalnom razvoju softvera. Sistem koji je predstavljen obuhvata veb prodavnicu koja ima dva tipa korisnika, kupac i administrator. Kupac može kupovati artikle pristupom prodavnici preko veba a administrator može izvršavati određene aktivnosti upotrebom drugih modula koji su predstavljeni na dijagramu. *Definisanje inkrementalnih podskupova podrazumeva da moduli moraju biti pravog nivoa granularnosti.*

Dijagram stila upotrebe na slici 2 prikazuje inkrementalni razvojni plan za modul admin.client za sledeću verziju softvera. Pretpostavka zasnovana na relaciji "koristi" na dijagramu je identifikovano da su moduli koje je dodatno potrebno predstaviti i objasniti:

- admin.core
- dao
- util

Navedeni moduli su neophodni za modul "admin.cilent" i njegovo neometano funkcionisanje. Modul "admin.client" koristi navedene module u predstavljenom primeru iz sistema.



Slika 3.2 Prikaz pogleda stila upotrebe za modul admin.client [Izvor: Clements [2]]

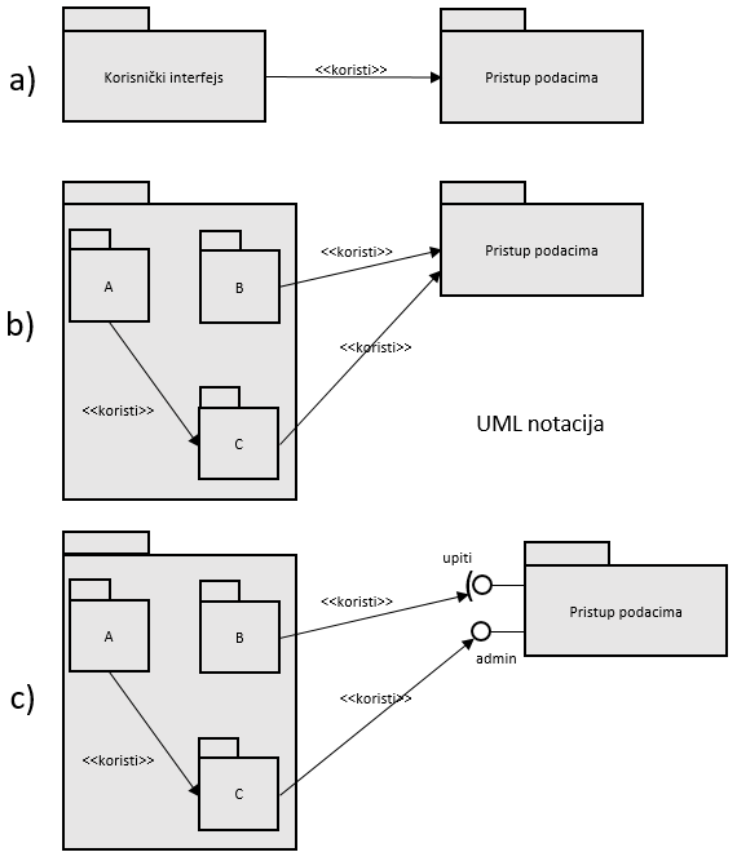
NOTACIJA STILA UPOTREBE

Neformalne grafičke notacije mogu se koristiti ali je često jednostavnije tabelom prikazati povezanost modula unutar sistema.

Notacije za stil upotrebe može se dokumentovati kao tabela sa dve kolone gde jedna kolona sa leve strane predstavlja elemente a druga kolona sa desne strane predstavlja elemente koji se koriste (od elemenata koji su predstavljeni u levoj koloni).

Poluformalne notacije - UML: Primenom UML-a za prikaz stila upotreba zahteva prikaz modula kao UML paketa. Na slici 3 prikazani su različiti primeri predstavljanja stila upotrebe:
a) Modul "Korisnički interfejs" koji "koristi" modul "Pristup podacima"
b) Modul "Korisnički interfejs" razložen u module A, B, C. Najmanje jedan od modula mora koristiti modul "Pristup podacima"
c) Modul "Pristup podacima" ima dva interfejsa koji se koriste od strane modula B i C.

Veza sa drugim stilovima: Stil upotrebe može se koristiti sa slojevitim stilom i pomaže programerima u procesu implementacije. Nakon izbora implementacije vrši se prikaz upotrebe i upravljanje inkrementalnim podskupovima. Ukoliko modul sadrži podmodule razlaganje zahteva prikaz svih relacija ka identifikovanim podmodulima. Na slici 3 modul "Korisnički interfejs" je razložen na tri modula i ukoliko nijedan podmodul ne koristi modul "Pristup podacima" razlaganje nije održivo.

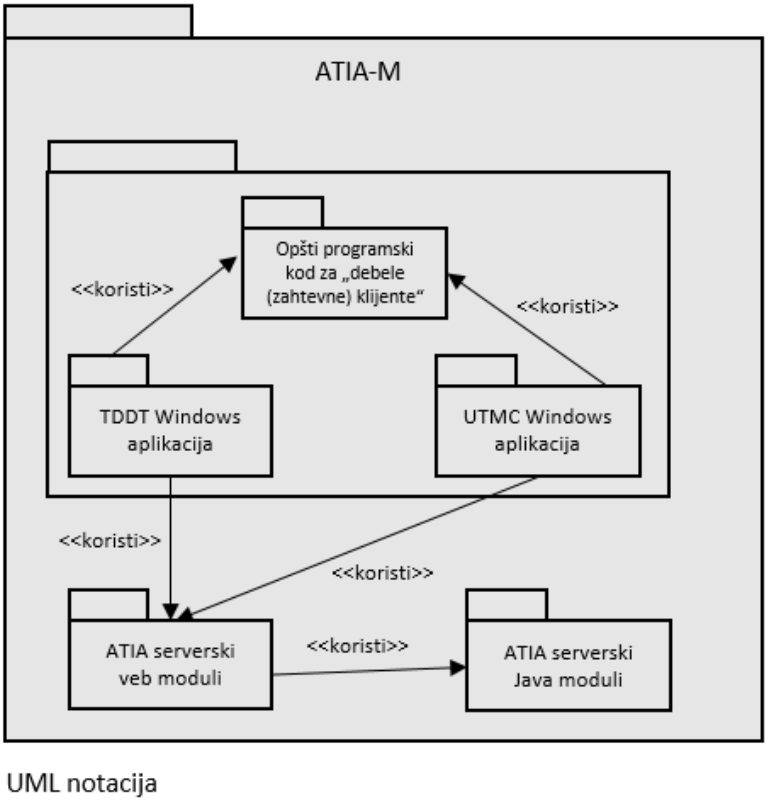


Slika 3.3 Različiti prikazi stila upotrebe [Izvor: Clements [2]]

POKAZNI PRIMER: STIL UPOTREBE

Prikazana je primena stila upotrebe na ATIA-M sistemu.

Slika 4 prikazuje prikazuje dijagram glavnog nivoa pogleda upotrebe za ATIA-M sistem. Modul TDDT i UTMC "koriste" "ATIA serverski veb modul". Takođe, "ATIA serverski veb moduli" koriste "ATIA serverske Java module".



Slika 3.4 Pogled stila upotrebe na ATIA-M sistem (Izvor: Clements [2])

Sistem predstavljen na slici 4, kao što je prethodno objašnjeno, podeljen je na module. Prvobitna podela izvršena je na module koji se nalaze na serverskoj strani aplikacije (veb module) i na module koji se nalaze na klijentskoj strani. Veze koje su predstavljene na dijagramu prikazuju module koji koriste druge module za svoj rad.

- **TDDT (Trening razvojni alat) Windows aplikacija** i **UTMC (Jedinica za upravljanje obuke) Windows aplikacija** koristi modul u kome se nalazi **opšti programski kod** kao i **ATIA serverske veb module**
- **ATIA serverski veb moduli** koriste **ATIA serverski Java module**

Na ovaj način prikazana je povezanost identifikovanih modula upotrebom relacije **<<koristi>>.**

Poglavlje 4

Stil generalizacije softverske arhitekture

PREGLED STILA GENERALIZACIJE

Stil je koristan kada projektant softvera želi da omogući proširenje i evoluciju arhitekture ili individualnih elemenata arhitekture.

Stil generalizacije koristi "je" relaciju. Kada su moduli predstavljeni kroz stil generalizacije to znači da "roditelj" modul predstavlja generalnu verziju modula "dete". Unapređenja modula mogu biti izvršena dodavanjem, brisanjem ili izmenom modula "deteta" dok se unapređenje roditelja automatski primenjuju na modul "dete".

ELEMENTI, RELACIJE I SVOJSTVA STILA GENERALIZACIJE

Elementi stila generalizacije su moduli.

Element generalizacije je modul. Relacija koja se koristi je generalizacija, konkretno "je" relacija. Modul može biti apstraktan tako da ne sadrži kompletnu implementaciju već od drugog modula dobija informacije o implementaciji.

Pregled	Stil generalizacije koristi „je“ relaciju da podrži proširenje i evoluciju arhitekture ili njenih individualnih elemenata. Moduli su predstavljeni tako da mogu da preuzmu izmene ili varijacije drugih modula.
Elementi	Moduli. Modul može imati apstraktno svojstvo koje znači da modul ne sadrži kompletnu implementaciju.
Relacije	„je“ („is a“)
Ograničenja	Modul može imati više „roditelj“ modula iako takav pristup često može predstavljati opasan i nepreporučljiv pristup projektovanja softverske arhitekture. Ciklusi u stilu generalizacije nisu dozvoljeni. To znači da modul „deteta“ ne može biti generalizacija jednog ili više modula unutar pogleda.
Upotreba	Prikazivanje nasleđivanja u objektno-orijentisanom projektovanju Opis evolucije i proširenja softvera Podrška ponovnom korišćenju modula

Slika 4.1 Prikaz stila generalizacije [Izvor: Clements [2]]

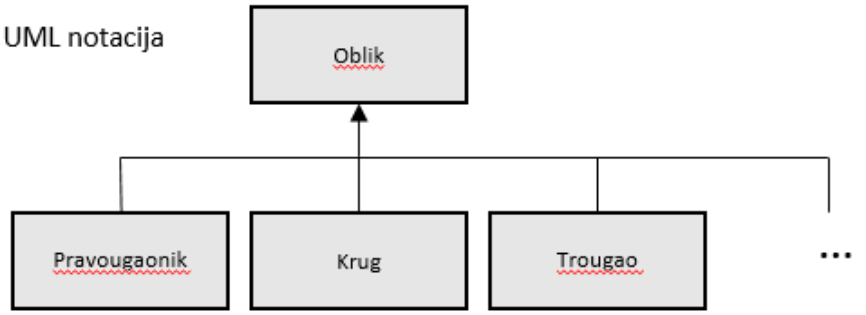
UPOTREBA STILA GENERALIZACIJE

Stil generalizacije može biti primenjen na različite tipove softverske arhitekture.

Generalizacija može biti korišćena za:

- **Objektno-orijentisano projektovanje**. Stil generalizacije omogućava izražavanje nasleđivanja u objektno-orijentisanom projektovanju sistema.
- **Produžavanje**. Često je lakše razumeti modul koji je drugačiji od drugog poznatog modula nego razumeti nov modul od nule. Generalizacija predstavlja mehanizam za proizvodnju inkrementalnih opisa u cilju formiranja potpunog opisa modula.
- **Lokalna promena ili varijacija**. Svrha arhitekture je da obezbedi stabilnu strukturu koja odgovara mogućim promenama ili varijacijama u procesu projektovanja softvera.
- **Ponovna upotreba**. Odgovarajuće apstrakcije mogu se ponovo koristiti samo na nivou interfejsa ili kroz proces implementacije. Definisanje apstraktnih modula omogućava njihovu ponovnu upotrebu.

Izražavanje generalizacije moguće je kroz UML. Moduli se prikazuju kao klase ili interfejsi. Na slici 2 prikazuje se osnovna notacija koja je dostupna u UML-u za klasu ili interfejs.



Slika 4.2 Predstavljanje modula stila generalizacije kroz UML [Izvor: Clements [2]]

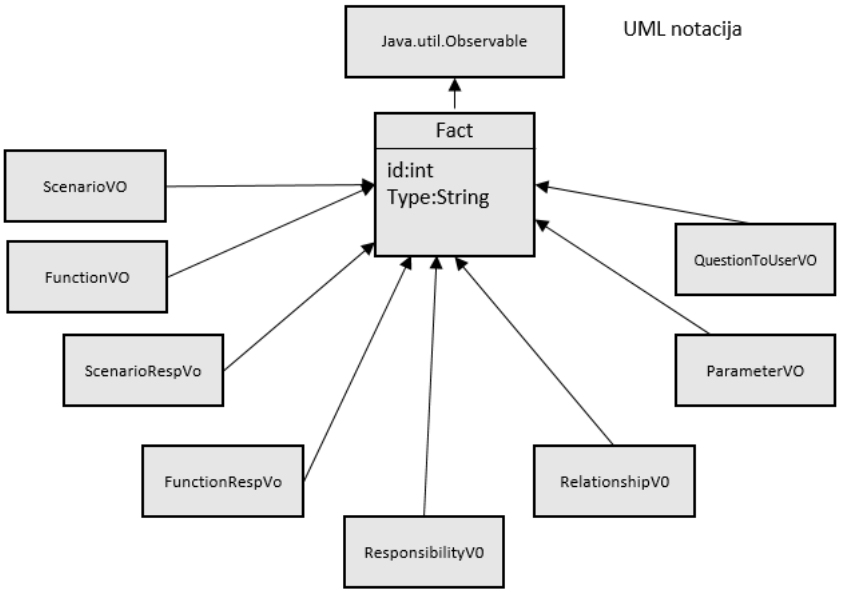
Klase ili interfejsi predstavljeni su pravougaonikom sa punom linijom. "Oblik" modul predstavlja "roditelj" modul podmodulima "Pravougaonik", "Krug" i "Trougao". Modul "Oblik" je opšti a podmoduli su specijalizovane verzije. Strelica je okrenuta ka modulu "Oblik" od njegovih podmodula.

Veza sa drugim stilovima: Realizacija nasleđivanja i interfejsa dopunjuje ostale module i često se koristi u okviru prikaza stila generalizacije uz obavezno razlaganje (dekompoziciju).

POKAZNI PRIMER: STIL GENERALIZACIJE

Prikazan je pokazni primer upotrebe stila generalizacije na alatu ArchE.

Na slici 3 prikazan je deo pogleda generalizacije iz ArchE alata (SEI Architecture Expert). Alat omogućava projektovanje softverske arhitekture shodno atributima kvaliteta, korisničkim zahtevima i definisanim pravilima projektovanja. Pogled omogućava prikaz operacija koje se izvršavaju. Takođe, klase koje su prikazane na ovom dijagramu koriste se i u drugim dijagramima gde im se detaljno opisuju atributi, radne operacije iz svake klase kao i odnosi između ovih i drugih modula sistema.



Slika 4.3 Predstavljanje modula stila generalizacije kroz UML [Izvor: Clements [2]]

Generalizacija prikazana slici 4 prikazuje PetStore aplikaciju. Razvijena je kao veb aplikacija koja se sastoji iz nekoliko slojeva i služi kao internet prodavnica kućnih ljubimaca. Hijerarhije klasa predstavljaju događaje u sistemu i realizacije interfejsa. Paket, predstavljen na desnoj strani slike je deo okvira veb aplikacije koji služi za upravljanje događajima. Za aplikaciju su definisani specifični događaji i oni se koriste za interakciju između modula.

Slika 4.4 Predstavljanje modula stila generalizacije kroz UML [Izvor: Clements [2]]

GENERALIZATIONS (VIDEO)

Trajanje: 1:26 minuta.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 5

Stil slojeva softverske arhitekture

PREGLED STILA SLOJEVA

Slojeviti stil kao i svi stilovi modula vrši podelu softvera na jedinice.

Jedinice u ovom stilu predstavljaju slojeve. Svaki sloj predstavlja grupu modula koji su povezani i omogućavaju povezani skup usluga. Odnosi između slojeva moraju biti jednosmerni. Slojeviti prikaz arhitekture softvera jedan je od često korišćenih prikazana u dokumentaciji. Korišćena relacija ovog sloja je dozvoljeno da koristi (" allowed to use"). Takođe, često se dešava da projektanti softvera pogrešno koriste prikaz softverske arhitekture prikazujuću slojeve sistema čak i ako oni nisu slojeviti. Softverska arhitektura se kroz slojeviti stil prikazuje uglavnom odozgo nadole. U softverskoj arhitekturi često može biti prisutno pravilo da moduli na veoma visokom nivou zahtevaju korišćenje modula na veoma niskom nivou. Dijagram slojevitog stila mora prikazati ovako projektovanu softversku arhitekturu.
Slojevi predstavljaju logične grupe koje omogućavaju pomoć u kreiranju i komuniciranju arhitekture ali često nisu eksplicitno ograničene kroz programski kod. Izvorni kod može otkriti gde se koji modul koristi i šta određeni modul koristiti unutar sistema. Kriterijumi za definisanje slojeva sistema mogu biti očekivanja projektanta softvera da će određeni sloj biti nezavisno razvijan od drugih slojeva ili zaduženje određenog člana tima da razvija određeni sloj sistema. Takođe, podela slojeva u razvojnom timu može biti i na osnovu specifičnog znanja nekog od članova razvojnog tima kome će biti dodeljen određeni sloj.

ELEMENTI, RELACIJE I SVOJSTVA STILA SLOJEVA

Slojevi omogućavaju postavljanje atributa kvaliteta za modifikaciju i prenosivost softverskog sistema.

Elementi slojevitog stila su slojevi. Sloj predstavlja kolekciju modula a moduli mogu biti bilo šta kao na primer veb servisi ili deljivi podaci. U slojevitom stilu dozvoljena je komunikacija između slojeva relacijom "dozvoljeno da koristi". Za dva sloja koja koriste navedenu relaciju svakom modulu koji je u prvom sloju je dozvoljeno da koristi drugi modul drugog sloja. Slojevi mogu imati određena svojstva koja moraju biti dokumentovana kroz katalog elemenata unutar grafičkog prikaza:

- Sadržaj. Opisuje sloj i sadrži uputstva kako moduli funkcionišu u sloju i na koji način se implementiraju. Svaki modul treba dodeliti tačno jednom sloju. Slojevi mogu imati oznake koje su opisne ili nejasne kao na primer "mrežni komunikacioni sloj" ili "sloj poslovnih pravila" pa je potreban dodatni opis koji identifikuje kompletan sadržaj svakog sloja.

- Softveru je dozvoljeno korišćenje sloja. Ovaj deo dokumentacije objašnjava izuzetke, ukoliko ih ima kao i pravila korišćenja određenog sloja sistema.

Sloj predstavlja primenu principa prikrivanja informacija. Promena na donjem sloju softverske arhitekture može biti sakrivena iza njegovog interfejsa i neće uticati na slojeve iznad u hijerarhiji. Promene u okviru određenog sloja mogu uticati na pretpostavku u performansama softvera (može se desiti problem u izmeni sloja koji će koristiti više resursa nego što mu je potrebno).
Pogrešna pretpostavka je da slojevi uvode dodatne troškove izvršenja, jer u složenim sistemima često se dešava da se troškovi razvoja i implementacije softvera primenom slojevitog stila na softversku arhitekturu dodatno smanjuju.
Slojevi omogućavaju projektantu softvera da kroz organizovanje modula u slojeve sa interfejsima upravlja složenošću strukture softvera i programerima koji rade na izvršavanju zadataka.

NOTACIJA STILA SLOJEVA

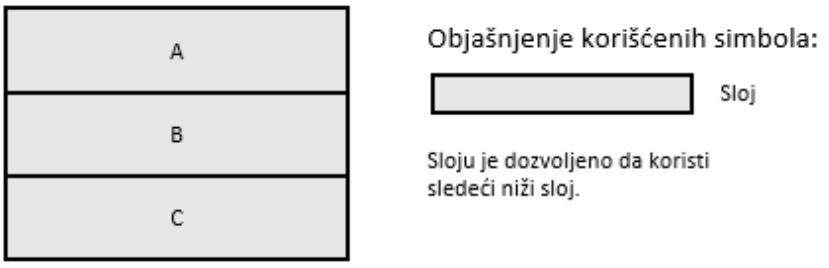
Slojevi se najčešće crtaju kao naslagane kutije.

Relacija "dozvoljeno da koristi" se crta od ozgo a dole. Na slici 1 prikazan je sloj kome je dozvoljeno korišćenje bilo kog sloja na nižem nivou. UML

Unutar UML-a ne postoji oznaka za sloj. Slojevi mogu biti predstavljeni kao paketi, kao što je prikazano na slici 2. Paket je mehanizam opšte namene za grupisanje elemenata i odgovara pojmu sloja. Relacija koja se koristi u tom slučaju može biti stereotipna zavisnost između paketa slojeva. Ako paket A može pristupiti paketu B, paket B može pristupiti paketu C ali to ne znači automatski da paket A može pristupiti paketu C.

Veza sa drugim stilovima: Pored dijagrama slojevitog stila često se koriste drugi dijagrami:

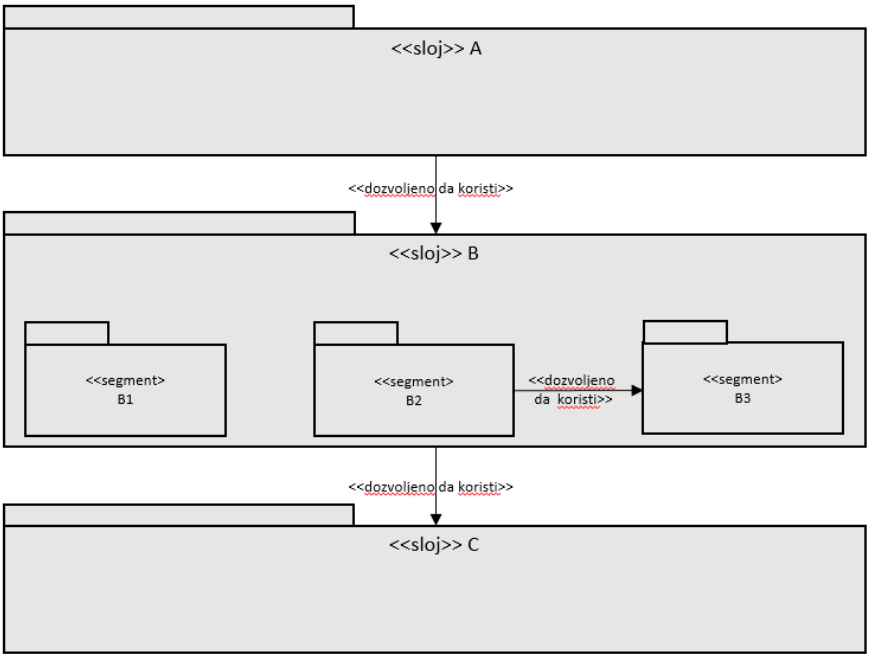
1. **Modul razlaganja**. Slojevi u slojevitom pogledu su uvek povezani a sloj može da prikaže više od jednog modula. Dva podmodula mogu biti deo različitih slojeva.



Slika 5.1 Prikaz slojevitog stila modula [Izvor: Clements [2]]

U slučaju da se modul prikazuje na više slojeva potrebno je to naglasiti korišćenjem različite boje na grafičkom prikazu sloja.
2. **Položaj.** Slojevi se često prikazuju sa nivoima što može zbuniti čitaoca dokumentacije.

Višeslojni stil je stil komponente i konektora jer se skupljaju i prikazuju komponente izvršavanja.


<<sloj>> A
<<dozvoljeno da koristi>>
<<sloj>> B
<>
B1
<>
B2
<<dozvoljeno
da koristi>>
<>
B3
<<dozvoljeno da koristi>>
<<sloj>> C

UML notacija

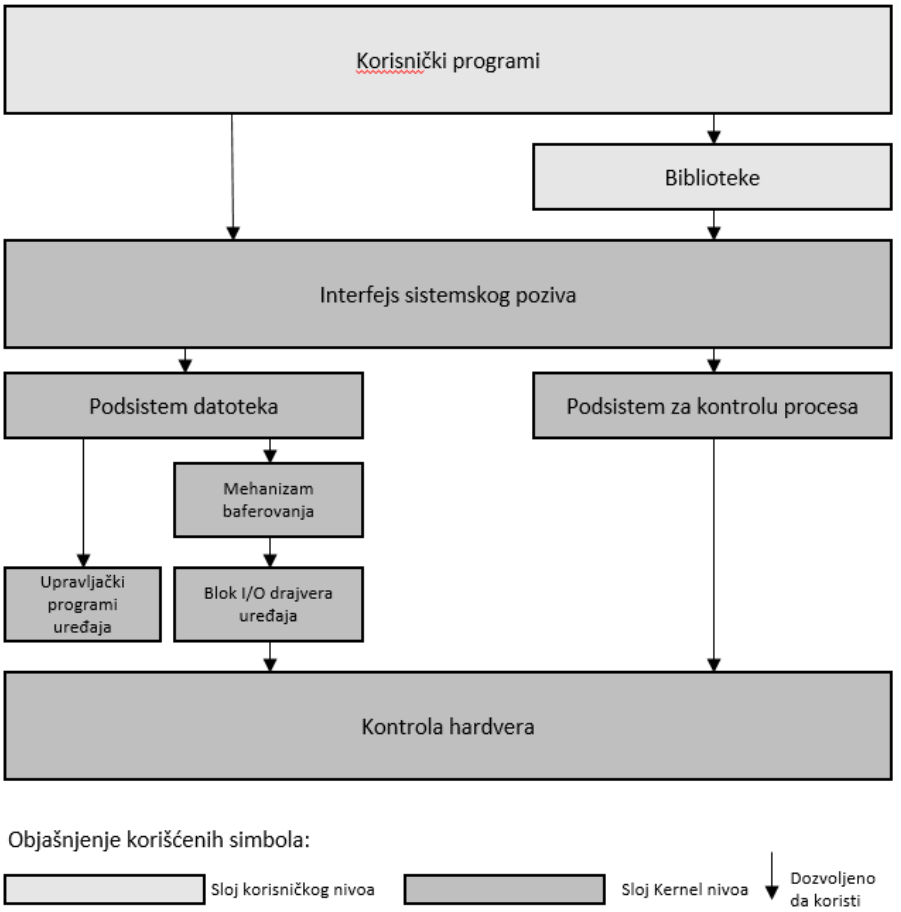
Slika 5.2 Prikaz slojevitog stila modula kroz UML [Izvor: Clements [2]]

POKAZNI PRIMER: STIL SLOJEVA

Prikazana je primena stila slojeva na sistemu Unix System V.

Na slici 3 prikazana je primena slojevitog projektovanja operativnog sistema UNIX System V. Donji slojevi čine sistemsko jezgro dok gornji slojevi predstavljaju programere koji koriste sistemsko jezgro ili biblioteke koje pristupaju jezgru kroz sistemske pozive. Podsistem datoteka služi za upravljanje datotekama (uređaja koji se tretiraju kao datoteke) i koji administriraju slobodan prostor, kontrolišu pristup i čitaju ili pišu podatke.
Podsistem za kontrolu procesa odgovoran je za planiranje procesa, sinhronizaciju procesa, upravljanje memorijom. Hardverski kontrolni sloj služi za upravljanje prekidima i komunikaciju sa mašinom.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.



Slika 5.3 Glavni prikaz slojevitog stila sistema Unix System V [Izvor: Clements [2]]

Poglavlje 6

Stil aspekta softverske arhitekture

PREGLED STILA ASPEKTA

Stil aspekta omogućava izolaciju određenog modula u softverskoj arhitekturi.

Kada se vrši implementacija softverskih modula, logika je da se moduli izdvoje tako da svaki modul bude odgovoran za određenu funkcionalnost. Tako, na primer, ukoliko je potrebno napraviti bankarski sistem, moduli koji će činiti taj sistem su: "Račun", "Korisnik", "ATM". Modul "Račun" u ovom slučaju sadržao bi podatke koji se odnose na poslovnu logiku (otvaranje ili zatvaranje naloga, depozit, transfer). U praksi prilikom kreiranja softvera potrebno je obratiti pažnju na unakrsne funkcionalnosti koje obuhvataju više modula kako bi sistem izvršio određenu operaciju. Stil aspekta definiše da moduli koji su odgovorni za unakrsno kretanje podataka moraju biti postavljeni u jedan ili više aspekata gledišta. Takvi moduli se nazivaju aspekti (nazvani po terminologiji aspektno orijentisanog programiranja AOP). Prikaz aspekata treba da sadrži informacije o svakom modulu i načinu povezivanja sa drugim modulima koji zahtevaju funkciju unakrsnog kretanja. Stil aspekta je primenjiv kroz nasleđivanje klasa i interfejsa, korisničkih biblioteka ili drugih alternativa. Primenom aspekata omogućeno je poboljšanje modifikacije modula koji spadaju u funkcionalnosti poslovnog domena sistema.

ELEMENTI, RELACIJE I SVOJSTVA STILA ASPEKTA

Elementi unutar stila aspekta su aspektni moduli.

Odnos koji se koristi između aspekata naziva se unakrsni rez. Aspekt vrši unakrsni rez modulu ukoliko aspekt sadrži funkcionalnost unakrsnog reza ka samom modulu. Aspekt može sadržati ista svojstva kao i regularan modul. Takođe, može sadržati i svojstva koja opisuju modul koji je cilj aspekta.

Koristi se za modelovanje i implementaciju unakrsnih koncepata. Poboljšava modifikovanje softverskog sistema povećanjem modularnosti i izdavanjem unakrsnih funkcija poslovnog domena.

Pregled	Stil aspekta prikazuje aspektne module koji implementiraju unakrsne koncepte i omogućavaju prikaz veza prema drugim modulima u sistemu.
Elementi	Aspekt koji predstavlja specijalizovani modul koji sadrži implementaciju unakrsnog koncepta.
Relacije	Unakrsni koncepti između modula
Ograničenja	Jedan aspekt može primeniti unakrsne koncepte prema jednom ili više modula. Aspekt koji primenjuje unakrsne koncepte može izazvati beskrajnu rekurziju zavisno od implementacije.
Upotreba	Modelovanje unakrsnog koncepta u objektno-orijentisanom dizajnu Poboljšanje prilagodljivosti

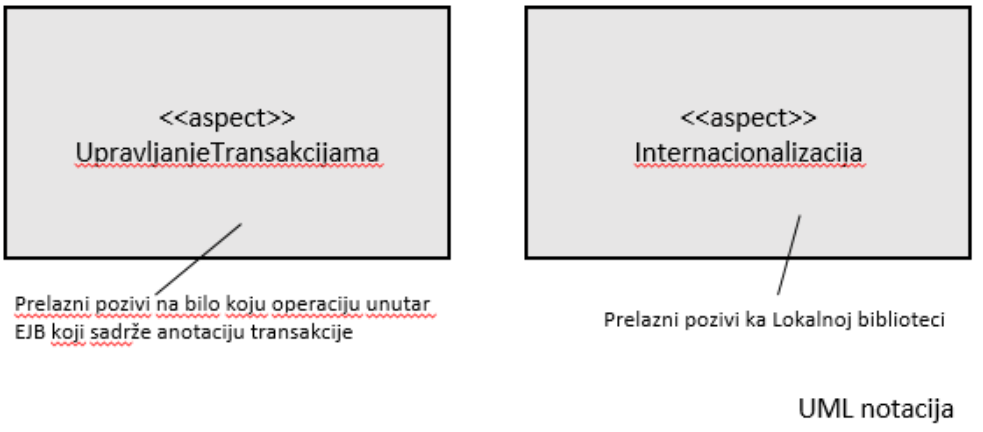
Slika 6.1 Prikaz stila aspekta [Izvor: Clements [2]]

NOTACIJA STILA ASPEKTA

Za prikaz aspekata UML nema definisan simbol tako da se najčešće koriste klase unutar klasnog dijagrama.

Aspekti su slični klasama, mogu sadržati atribute i operacije a mogu izvršiti i nasleđivanje. Prikaz unakrsnog koncepta može biti predstavljena kao stereotipna zavisnost koja ide sa aspekta na svaki ukršteni modul. Takođe, može biti dodat komentar modulu aspekta kako bi veza između aspekta i drugog modula bila opisana (kroz formalnu sintaksu ili na prirodnom jeziku). Na slici 2 prikazan je primer primene aspektnog stila na softversku arhitekturu.

Veza sa drugim stilovima: Aspekti omogućavaju nasleđivanje i stil aspekta može biti kombinovan sa stilom generalizacije ukoliko je potrebno prikazati hijerarhiju aspekata.



Slika 6.2 Glavni prikaz aspektnog stila na softversku arhitekturu [Izvor: Clements [2]]

POKAZNI PRIMER: STIL ASPEKTA

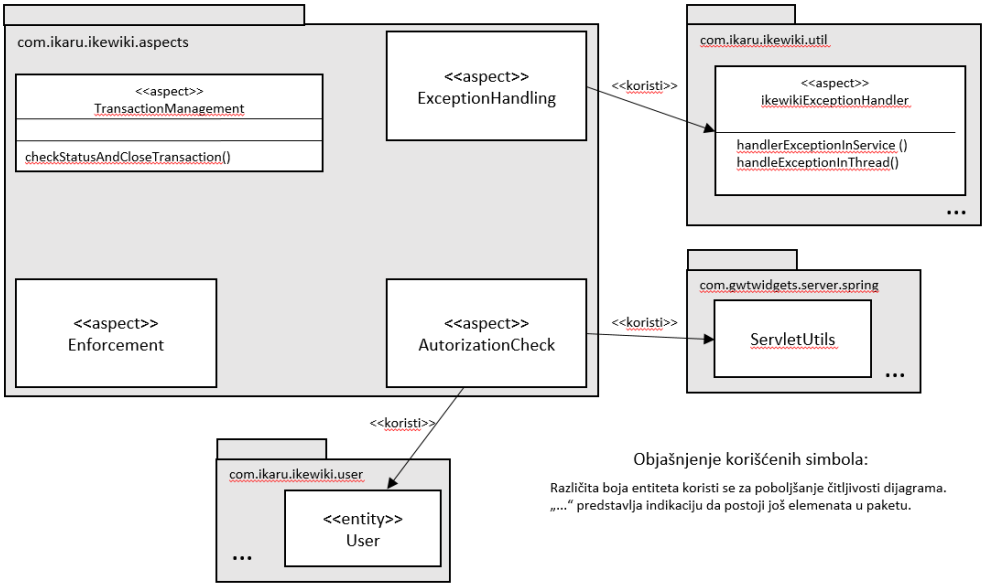
Prikazana je primena stila aspekta na sistemu IkeWiki.

Na slici 1 prikazan je pogled aspekta softverske arhitekture aplikacije "IkeWiki". Projektovanje ove aplikacije zahteva upotrebu aspekata za upravljanje transakcijama, upravljanje izuzetkom, proveru autorizacije, sprovođenje arhitektonskih ograničenja. Linije koje bi predstavljale relacije nisu dobro rešenje pa je projektant softverske arhitekture odlučio da kroz komentare jednostavno navede način prelaska aspekta sa jednog modula na drugi.

Aplikacija je razvijena korišćenjem Java EE koristeći Spring okviru i Google Web Toolkit. Aspekt "TransactionManagement" omogućava da svi zahtevi koje primi server budu zatvoreni i da se resursi baze podataka koriste kako bi se izvršio povratak u slučaju izuzetka. Aspekt "ExceptionHandling" služi za prijavljivanje greške i slanje obaveštenja putem mejla korisniku. Takođe, prikazuje i poruku korisniku zbog čega je došlo do greške unutar sistema. Aspekt "AuthorizationCheck" se koristi da bi proverio da li trenutni korisnik ima dozvolu za izvršavanje određene metode.

POKAZNI PRIMER: STIL ASPEKTA - NASTAVAK

Na slici je prikazan pokazni primer upotrebe stila aspekta



Slika 6.3 Primer upotrebe stila aspekta [Izvor: Clements [2]]

Poglavlje 7

Model podataka softverske arhitekture

PREGLED MODELA PODATAKA

Modeliranje podataka je standardna aktivnost u procesu razvoja informacionih sistema.

Izlaz iz procesa modeliranja predstavlja model podataka koji opisuje statičku strukturu informacija u vidu entiteta podataka i njihovih međusobnih odnosa (relacija). U bankarskom sistemu obično se koriste entitet Račun, Korisnik i Zajam. Modul Račun može imati nekoliko atributa kao što su broj računa, tip, status i trenutno stanje. Veza može prikazati da kupac može imati jedan ili više računa i da jedan račun može biti povezan sa jednim ili dva korisnika. Model podataka se često crta upotrebom ERD dijagrama ili UML klasnog dijagrama. Tipovi modela podataka mogu biti:

- **Konceptualni**. Konceptualni model podataka se fokusira na entitete i njihove veze.
- **Logički.** Logički model podataka predstavlja evoluciju konceptualnog modela podataka kroz tehnologiju upravljanja podacima (kao što su relacione baze podataka).
- **Fizički**. Fizički model podataka vrši implementaciju entiteta podataka. Omogućava optimizaciju, kreiranje identifikacionih ključeva i indeksa i vrši optimizaciju performansi.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ELEMENTI, RELACIJE I SVOJSTVA MODELA PODATAKA

Entiteti u modelu podataka su povezani sa nekim od modula u drugačijim prikazima modula određenog sistema, naročito sa modulima koji sadrže podatke u memoriji.

Elementi u modelu podataka se nazivaju entiteti, dok svaki objekat koji sadrži informaciju koja može biti skladištena ili prezentovana u sistemu može biti entitet.
Najpopularnija ERD (" Entity Relationship Diagram") notacija koristi linije sa posebnim simbolima na svakom kraju kako bi ukazala na kardinalnost.
UML

Model podataka može biti prikazan kao UML klasni dijagram gde klase predstavljaju entitete podataka. Atributi se dodaju u okviru entiteta kao i moguće operacije. UML je prvobitno kreiran za objektno-orijentisano modelovanje a ne za modelovanje podataka. Sada kroz UML moguće je kreirati konceptualni model podataka koji omogućava prikaz entiteta i njihovih svojstava.

U objektno-orijentisanim sistemom koji koriste relacionu bazu podataka za čuvanje podataka, uglavnom se pronalaze klase koje odgovaraju određenim entitetima. Često mapiranje nije jedan na jedan jer relacijska paradigma se suštinski razlikuje od objektno-orijentisane paradigme. Ovaj problem je poznat kao neusklađenost objektno-relacionice impedance (Ambler 2006) i rešava se kroz alat za objektno-relaciono mapiranje (ORM) ili kroz šablone kao što su Hibernate za Javu ili LLBLGen za .NET.

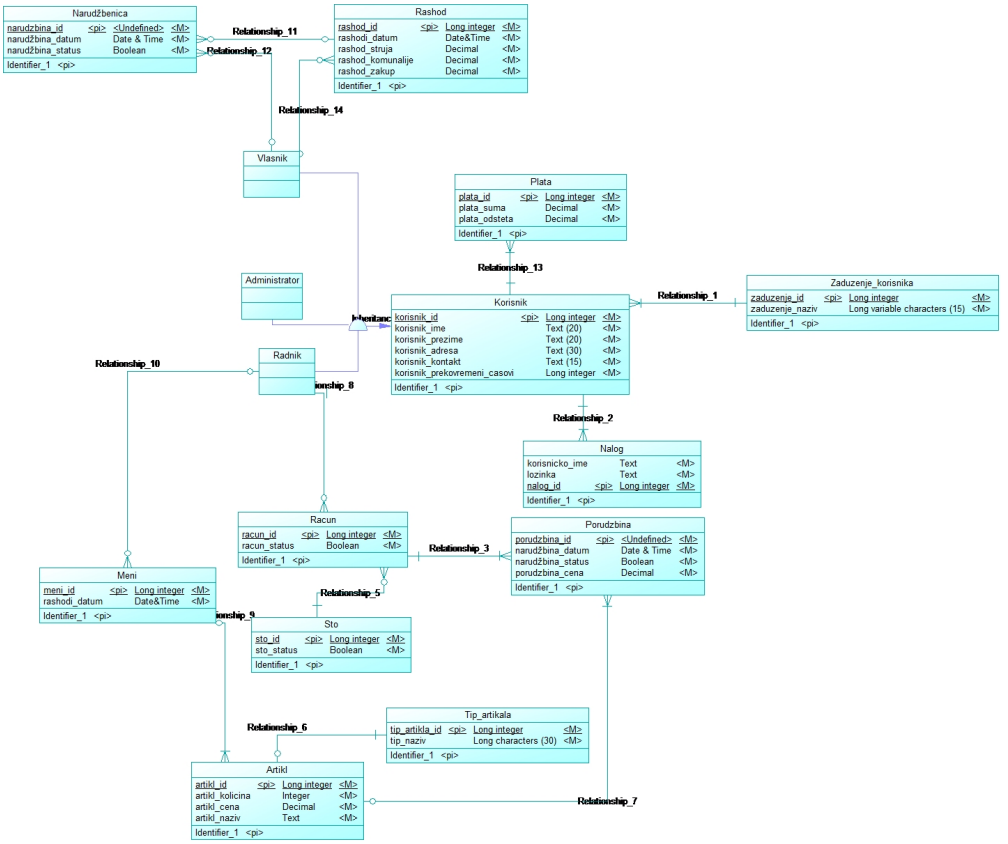
Projektant softvera treba da naznači koji moduli (kroz pogled modula), koje komponente (kroz pogled komponente i konektora) koriste entitete podataka.

Pregled	Model podataka opisuje strukturu entiteta podataka i njihovih relacija.
Elementi	Entitet podataka koji predstavlja objekat koji sadrži informacije koje su potrebne unutar sistema. Svojstva uključuju ime, atribute, primarni ključ kao i pravila za pristup korisnika u određenom entitetu.
Relacije	Jedan na jedan („one to one“), jedan na više („one to many“), više na više („many to many“) Generalizacija/specijalizacija koja omogućava upotrebu „je“ relacije između entiteta Agregacija koja pretvara vezu u entitet agregacije
Ograničenja	Prilikom modelovanja modela podataka treba izbegavati funkcionalne zavisnosti.
Upotreba	Opisivanje strukture podataka korišćenih u sistemu Omogućavanje analize uticaja izmena na model podataka Sprovođenje kvaliteta podataka izbegavanjem redudantnosti i nedoslednosti Vođenje implementacije modula koji pristupaju podacima

Slika 7.1 Prikaz stila modela podataka [Izvor: Clements [2]]

POKAZNI PRIMER: MODEL PODATAKA

Prikazan je primer modela podataka aplikacije za podršku rada restorana.



Slika 7.2 Glavni prikaz modela podataka za aplikaciju za podršku rada restorana[Izvor: Nebojša Gavrlović]

Na slici 2 prikazan je model podataka za aplikaciju za podršku rada restorana.pisanu u Microsoft .NET-u. Sistem za podršku rada restorana olakšaće poslovanje restorana tako što će omogućiti evidenciju svih aktivnosti koje se odvijaju tokom poslovanja objekta. Podaci se nalaze u relacionoj bazi podataka. Većina funkcionalnosti sastoji se od prikupljanja, kreiranja ili ažuriranja elemenata podataka koji su prikazani unutar modela podataka.

Poglavlje 8

Pokazna vežba-Stilovi softverske arhitekture

PRIMENA STILA RAZLAGANJA NA SOFTVERSKU ARHITEKTURU APLIKACIJE PERSONALNI TRENER

Primer upotrebe stila razlaganja na aplikaciji "Personalni trener".

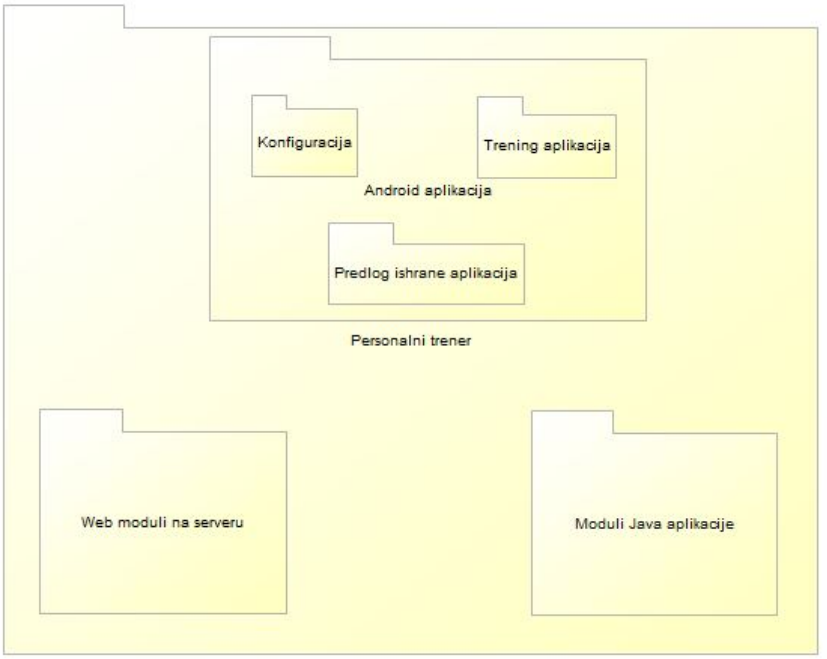
Softverska arhitektura aplikacije "Personalni trener" koja je korišćena na prethodnim vežbama predstavljena je upotrebom stila razlaganja. Aplikacija ima sledeće funkcionalnosti:

- definisanje treninga za korisnika
- predlog ishrane za korisnika
- konfigurisanje korisničkog profila

Shodno opisanim funkcionalnostima, izvršena je podela aplikacije na module tako što su identifikovani moduli: konfiguracija, trening aplikacija i predlog ishrane aplikacija. Navedene aplikacije treba da komuniciraju sa veb modulima na serveru i modulima java aplikacija u cilju razmene podataka o korisniku koji koristi aplikaciju i definisanja treninga i predloga ishrane.

Dva modula "Web moduli na serveru" i "Moduli Java aplikacije" sadrže svoje podmodule koji nisu predstavljeni kroz ovaj pogled stila razlaganja.

Slika 1 predstavlja samo početni korak u primeni stila razlaganja na softversku arhitekturu aplikacije "Personalni trener". (15min)



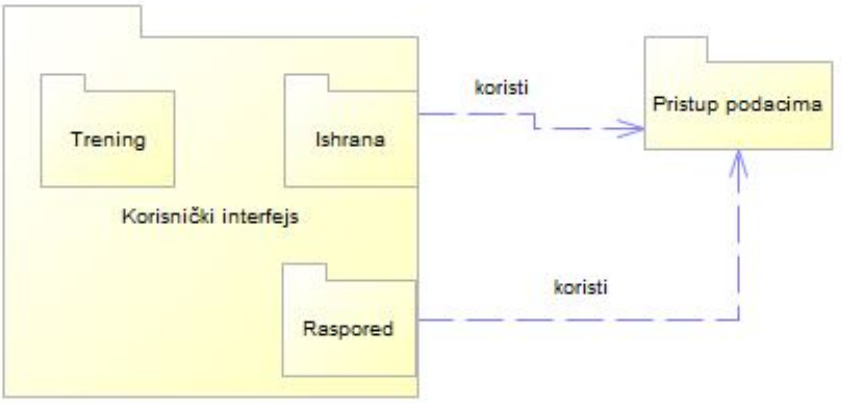
Slika 8.1 Primer primene stila razlaganja na aplikaciju "Personalni trener" [Izvor: Nebojša Gavrilović]

Preporuka je da se za predstavljanje stila razlaganja koristi Component diagram (preporučeni alati PowerDesigner ili draw.io).

PRIMENA STILA UPOTREBE NA SOFTVERSKU ARHITEKTURU APLIKACIJE PERSONALNI TRENER

Primer upotrebe stila upotrebe na aplikaciji "Personalni trener".

Na slici 2 prikazan je stil upotrebe primenjen na softversku arhitekturu aplikacije "Personalni trener". Sastoji se iz dva glavna modula "Korisnički interfejs" i "Pristup podacima". Modul "Korisnički interfejs" sadrži tri podmodula "Trening", "Ishrana" i "Raspored". Modul "Ishrana" koristi modul "Pristup podacima" kako bi izvršio određenu operaciju kao i modul "Raspored" koji takođe koristi modul "Pristup podacima". (10 min)



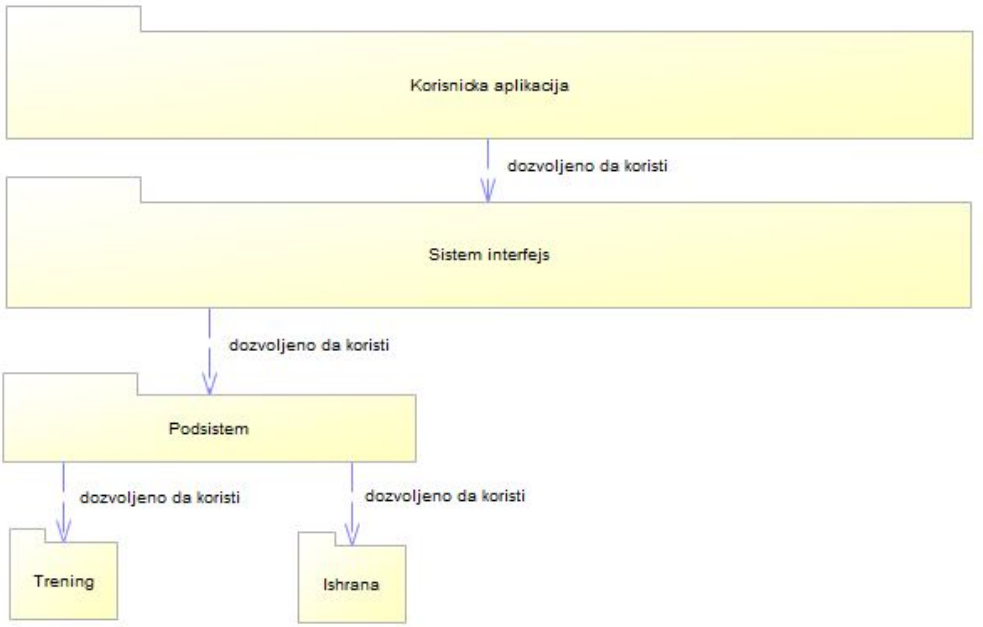
Slika 8.2 Primer primene stila upotrebe na aplikaciji "Personalni trener" [Izvor: Nebojša Gavrilović]

Preporuka je da se za predstavljanje stila upotrebe koristi Component diagram (preporučeni alati PowerDesigner ili draw.io).

PRIMENA STILA SLOJEVA NA SOFTVERSKU ARHITEKTURU APLIKACIJE PERSONALNI TRENER

Primer upotrebe stila slojeva na aplikaciji "Personalni trener".

Slika 3 prikazuje primenu stila slojeva na softverskoj arhitekturi aplikacije "Personalni trener". Prikazani su slojevi "Korisnička aplikacija", "Sistem interfejs", "Podsistem" i "Trening" i "Ishrana". Slojevi komuniciraju između sebe metodom odozgo na dole. (10min)



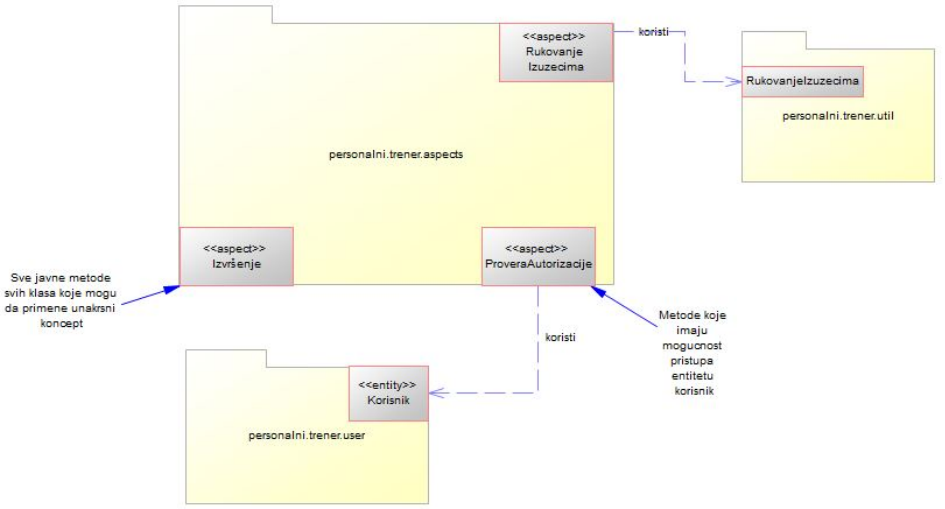
Slika 8.3 Primer primene stila slojeva na aplikaciji "Personalni trener" [Izvor: Nebojša Gavrilović]

Preporuka je da se za predstavljanje stila slojeva koristi Component diagram (preporučeni alati PowerDesigner ili draw.io).

PRIMENA STILA ASPEKTA NA SOFTVERSKU ARHITEKTURU APLIKACIJE PERSONALNI TRENER

Primer upotrebe stila aspekta na aplikaciji "Personalni trener".

Na slici 4 prikazana je softverska arhitektura aplikacije "Personalni trener" na kojoj je primenjen stil aspekta. Definisani su aspekti "Izvršenje", "Rukovanje izuzecima", "ProveraAutorizacije". Pored toga, u prikazu arhitekture nalaze se i entiteti "Korisnik" i "Rukovanje izuzecima" koji služe aspektima za obavljanje određenih operacija unutar sistema. (15min)

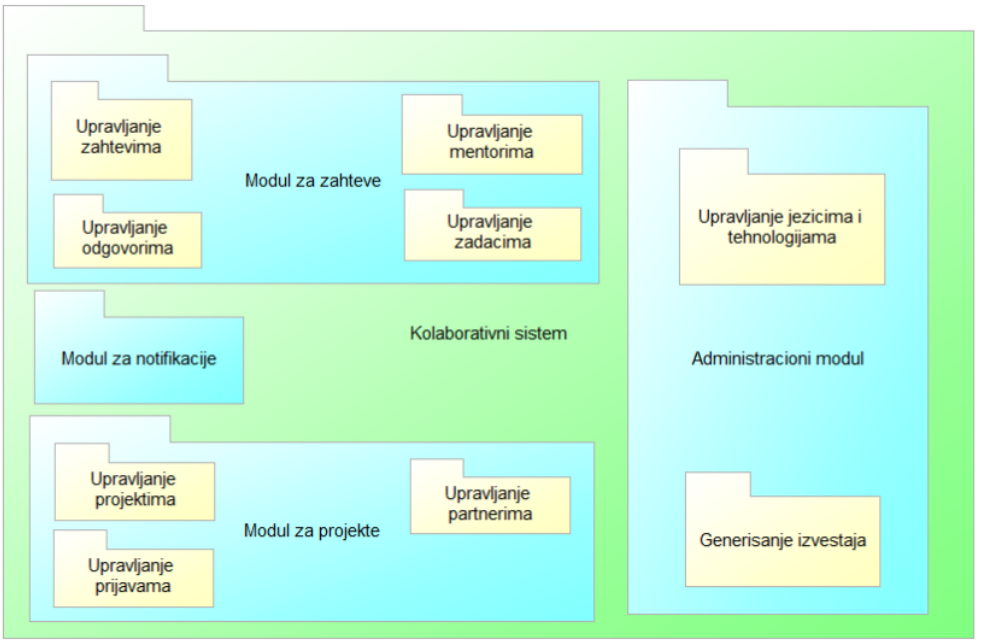


Slika 8.4 Primer primene stila aspekta na aplikaciji "Personalni trener" [Izvor: Nebojša Gavrilović]

Preporuka je da se za predstavljanje stila aspekta koristi Component diagram (preporučeni alati PowerDesigner ili draw.io).

PRIMENA STILA RAZLAGANJA NA SOFTVERSKU ARHITEKTURU SISTEMA ZA MENTORISANJE

Stilom razlaganja, sistem je podeljen na implementacione jedinice.

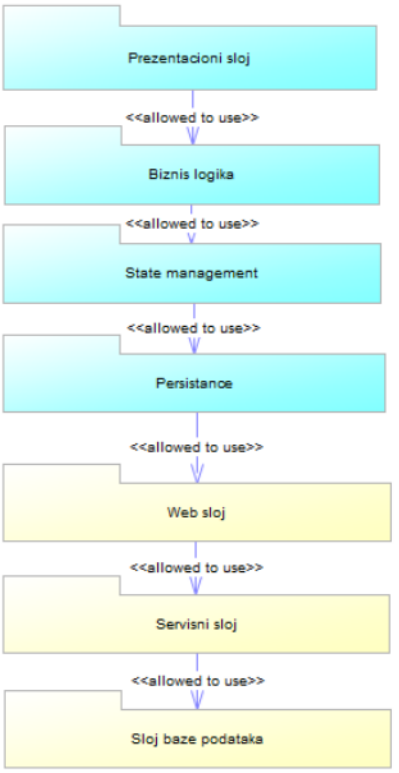


Slika 8.5 Stil razlaganja na softverskoj arhitekturi sistema za mentorisanje [Izvor: Nebojša Gavrilović]

Ovi moduli su nezavisni, ali komuniciraju jedni sa drugima kako bi ispunili funkcionalne zahteve. Ovaj sistem je podeljen u 3 modula: modul za projekte, za zahteve i administracioni modul. Moduli za zahteve i projekte omogućavaju upravljanje odgovarajućim podacima. Administracioni modul se koristi za generisanje izveštaja i administraciju sistema.(15min)

PRIMENA STILA STILA SLOJEVA NA SOFTVERSKU ARHITEKTURU SISTEMA ZA MENTORISANJE

Stilom slojeva, sistem je podeljen na 7 slojeva. Prvih 4 pripadaju frontend arhitekturi, a poslednjih 3 backend arhitekturi.



Slika 8.6 Primena stila slojeva [Izvor: Nebojša Gavrilović]

Frontend je podeljen u 4 sloja:

1. Prezentacioni sloj: DOM manipulacija, formatiranje podataka, događaji
2. Biznis logika: Aplikaciona logika, validacija podataka
3. State management: Upravljanje stanjima aplikacije
4. Persistance: Servisi, HTTP pozivi ka backend-u

Backend je podeljen u 3 sloja:

1. Web sloj: REST endpoints
2. Servisni sloj: Poslovna logika, obrada i validacija podataka
3. Sloj baze podataka: Upis i čitanje iz baze podataka (15min)

Poglavlje 9

Individualna vežba - Zadaci

ZADATAK ZA SAMOSTALAN RAD

Na osnovu pokaznih primera u vežbama potrebno je uraditi zadatke za individualni rad.

Softver je namenjen za zaposlene u medicinskoj ustanovi (medicinska sestra, doktor, direktor klinike).

Svaki korisnik se prijavljuje i na osnovu prijave ima različite opcije korišćenja sistema. Opcije su sledeće:

- Prijavljivanje na sistem klinike
- Otvaranje kartona pacijenta
- Pristup kartonu pacijenta
- Slanje pacijentovih podataka iz kartona lekaru
- Prepisivanje terapije
- Izdavanje recepta za leka
- Davanje uputa za dalje lečenje
- Pretraživanje zaposlenih
- Potpisivanje naloga, zapisnika i pravilnika klinike
- Prikaz pravilnika klinike

Korišćenjem opisanog scenarija potrebno je:

1. Prikazati softversku arhitekturu sistema kroz stil razlaganja (25min)
2. Prikazati softversku arhitekturu sistema kroz stil upotrebe (25min)
3. Prikazati softversku arhitekturu sistema kroz stil slojeva (25min)
4. Prikazati softversku arhitekturu sistema kroz stil aspekta (15min)

Uporediti modelovane softverske arhitekture i dokumentaciju za svaki modelovani stil. Utvrdite i objasnite stil arhitekture koji najviše odgovara opisanom scenariju. (30min)

ZADACI ZA SAMOSTALAN RAD - STIL RAZLAGANJA

Primeniti stil razlaganja na softversku arhitekturu proizvoljnog sistema.

Zadatak 1: Prikazati najviši nivo razlaganja (primenom stila razlaganja) softverske arhitekture na proizvoljnom sistemu. Prilikom modelovanja dijagrama pridržavati se pravilne notacije stila razlaganja. (10min)

Zadatak 2: Nakon prikaza najvišeg nivoa razlaganja proizvoljne softverske arhitekture, izvršiti podelu tako dobijene softverske arhitekture na podmodule. Objasniti tako dobijenu softversku arhitekturu. (10min)

ZADACI ZA SAMOSTALAN RAD - STIL UPOTREBE

Primeniti stil upotrebe na softversku arhitekturu proizvoljnog sistema.

Zadatak 1: Primeniti stil upotrebe na proizvoljnom sistemu. Modelovati odgovarajući dijagram i pridržavati se definisane notacije za predstavljanje stila upotrebe. (30min)
Zadatak 2: Detaljno dokumentovati svaki identifikovani modul u softverskoj arhitekturi i opisati relacije između modula. (10min)

ZADACI ZA SAMOSTALAN RAD - STIL GENERALIZACIJE

Primeniti stil generalizacje na softversku arhitekturu proizvoljnog sistema.

Zadatak 1: Predstaviti primenu stila generalizacije na proizvoljnoj softverskoj arhitekturi. Pridržavati se definisane notacije za predstavljanje stila generalizacije. Identifikovati module softverske arhitekture, prikazati klase i atribute tako dobijenih klasa. (30min)
Zadatak 2: Prikazati i dokumentovati relacije između identifikovanih modula softverske arhitekture i opisati primenu stila generalizacije. (10min)

ZADACI ZA SAMOSTALAN RAD - STIL SLOJEVA

Primeniti stil slojeva na softversku arhitekturu proizvoljnog sistema.

Zadatak 1: Odabrati proizvoljan sistem, primeniti stil slojeva i modelovati dijagram koji prikazuje sve identifikovane slojeve softverske arhitekture. (30min)
Zadatak 2: Izvršiti detaljnu specifikaciju relacija između slojeva u softverskoj arhitekturi. (10min)

ZADACI ZA SAMOSTALAN RAD - STIL ASPEKTA

Primeniti stil aspekta na softversku arhitekturu proizvoljnog sistema.

Zadatak 1: Primeniti aspektni stil na softversku arhtitekturu proizvoljnog sistema. Modelovati klasni dijagram aplikacije i pridržavati se notacije stila aspekta. (30min)
Zadatak 2: Prikazati nasleđivanje u odabranoj softverskoj arhitekturi. (10min)

ZADACI ZA SAMOSTALAN RAD - MODEL PODATAKA

Razviti model podataka proizvoljnog sistema. Izvršiti specifikaciju identifikovanih relacija između entiteta.

Zadatak 1: Kreirati model podataka (konceptualni model i fizički model) proizvoljnog sistema. (30min)
Zadatak 2: Specificirati relacije modela podataka i detaljno opisati entitete modela shodno definisanoj notaciji modela podataka. (10min)

ZADACI ZA DISKUSIJU NA VEŽBI

Otvorite diskusiju po svakom od dole postavljenih pitanja.

1. Primenom stila razlaganja na softverskoj arhitekturi određenog sistema, na koji način projektant softverske dokumentacije predstavlja sistem? Šta je fokus primene ovog stila? (10min)

2. Kroz stil generalizacije omogućeno je objektno-orijentisano projektovanje. Na koji način? (10min)

3. Šta se dešava u situaciji kada je potrebno izvršiti promenu na nekom od donjih slojeva softverske arhitekture? Navesti primer.(10min)

4. Koja je razlika između fizičkog i konceptualnog modela podataka i šta čitalac softverske dokumentacije može da vidi iz navedenih modela podataka. (10min)

5. U kojim situacijama je bolje prikazati tabelom povezanost modula unutar sistema? Na koji način tabela predstavlja pravi izvor informacija za čitaoca softverske dokumentacije? (10min)

Poglavlje 10

Domaći zadatak br.3

PRAVILA ZA IZRADU DOMAĆEG ZADATKA

Student zadatake treba samostalno da uradi i da dostavi asistentu.

Domaći zadatak br.3: Uporediti stil razlaganja i stil slojeva softverske arhitekture i navesti koji stil najviše odgovara odabranom sistemu.
Pri radu, koristite Power Designer.

Domaći zadaci su dobijaju se po definisanim instrukcijama. Domaći zadatak se imenuje: SE311-DZ03-ImePrezime-brIndeksa gde su vrednosti Ime, Prezime i br.Indeksa vaši podaci. Domaći zadatak je potrebno poslati na adresu asistenta:

nebojsa.gavrilovic@metropolitan.ac.rs (tradicionalni studenti iz Beograda i internet studenti)
jovana.jovic@metropolitan.ac.rs (tradicionalni studenti iz Niša)

sa naslovom (subject mail-a) SE311-DZ03. Potrebno je poslati modelovane dijagrame i dokument sa opisom dijagrama ili odgovorima na pitanja.

Napomena: Domaći zadaci treba da budu realizovani u zadatku navedenom razvojnom okruženju i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje rešenja sa interneta ili od drugih studenata strogo je zabranjeno.

Poglavlje 11

Zaključak

ZAKLJUČAK

U okviru ove lekcije prošli smo kroz stilove predstavljanja softverske arhitekture.

- Stil dekompozicije prikazuje podelu odgovornosti kroz module i podmodule
- Stil upotrebe prikazuje kako moduli zavise jedan od drugog. Pogled ovog stila omogućava inkrementalni razvoj i posebno je pogodan za analizu izmena unutar sistema
- Stil generalizacije se koristi u objektno-orijentisanim sistemima uz korišćenje nasleđivanja i povezivanja između modula
- Slojeviti stil omogućava podelu sistema na grupe modula koji imaju određena zaduženja u sistemu. Grupe se nazivaju slojevi, povezani su međusobno i omogućavaju sistemu portabilnost i mogućnost izmene
- Stil aspekta prikazuje specijalne module (aspekte) koji su odgovorni za primenu unakrsnih koncepata. Pogled stila aspekta omogućava implementaciju sistema korišćenjem aspektno-orijentisanog programiranja
- Model podataka opisuje strukturu podataka unutar sistema kao i veze između entiteta. Pomaže u procesu implementacije i omogućava unapređenje performansi i izmena sistema.

LITERATURA

U ovoj lekciji korišćena je kao obavezna literatura, referenca 2, poglavlje 1 i poglavlje 2(2. izdanje).

Obavezna literatura:

1. Onlajn nastavni materijal na predmetu SE311 Projektovanje i arhitektura softvera, , školska 2018/19, Univerzitet Metropolitan
2. P. Clements et ad. , Documenting Software Architectures: Views and Beyond, 2nd ed., Pearson Education, poglavlje 1 (strane od 55 do 64), poglavlje 2 (strane od 65 do 121)

Dopunska literatura:

1. D. Budgen, Software Design, 2nd ed.,Addison-Wesley, 2003.
2. T.C.Lethbrigde, R. Lagariere - Object-Oriented Software Engineering - Practical Software Development using UML and Java - 2005
3. Ian Sommervile, Software Engineering, Tenth Edition, Pearson Education Inc.,2016. ili 9th Edition, 2011
4. P.B. Kruchten, “The 4+1 View Model of Architecture,” IEEE Software, vol. 12, no. 6, 1995,

pp. 42–55.
5. E. Gamma et al., Design Patterns:Elements of Reusable Object-Oriented Software, 1st ed., Addison-Wesley Professional, 1994.
6. J. Nielsen, Usability Engineering, Morgan Kaufmann, 1993.
7. Service-oriened Architecture, Concept, Technology, and Design, autora T. Erl u izdanju Pretince Hall, 2005, ISBN 0-13-185858-0.
8. P. Stevens, Using UML – Software Engineering with Objects and Components, Second Edition, Assison-Wesley, Pearson Eduction, 2006
9. R. Pressman, Software Engineering – A Practioner's Approach, Seventh Edition, McGraw Hill Higher Education, 2010

Veb lokacije :
• http://www.software-engin.com
• http://www.uml.org/