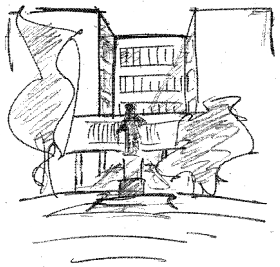
Развој софтвера 1



Саша Малков
Универзитет у Београду
Математички факултет
2023/2024

Развој софтвера је...

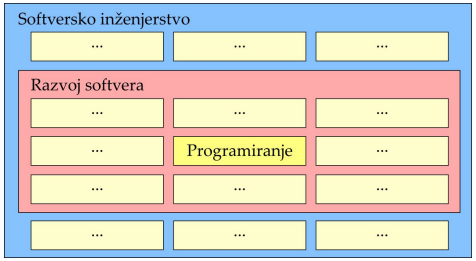
- "... писање рачунарских програма"
- "... развој софтверских производа"
- "... процес израде софтвера"
- ???

Развој софтвера је...

- ... сложен процес израде софтвера,
  који обухвата више различитих активности,
  међу којима су најважније:
  - анализа система
  - препознавање и обликовање захтева
  - анализа захтева
  - детаљна спецификација захтева
  - пројектовање софтвера
  - имплементација софтвера
  - тестирање
  - одржавање
  - планирање, праћење и вођење развојног процеса

Развој софтвера и софтверско инжењерство

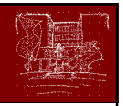
- "***Развој софтвера***" и "***Софтверско инжењерство***"
  - имају много тога заједничког али нису исто
  - ипак, често можемо да их користимо као синониме



Развој софтвера и софтверско инжењерство

- "***Развој софтвера***" и "***Софтверско инжењерство***"
  - имају много тога заједничког али нису исто
  - ипак, често можемо да их користимо као синониме
- У оквиру овог курса сматраћемо да је најважнија разлика то што СИ, поред активности на РС, обухвата бројне техничке послове на вођењу пројеката, као што су:
  - процењивање вредности посла и ризика;
  - организовање тимова
  - планирање ресурса;
  - прављење распореда и заказивање рокова;
  - и друго.
- Овде ћемо се бавити примарно аспектима РС
  - специфичне аспекте СИ ћемо сусретати и дотицати
  - посвећиваћемо им се у мери у којој је то неопходно

План курса



- **Развој софтвера**
  - Проблеми у развоју и њихово решавање
  - Развој ОО методологија, *UML*
  - Важне технике и алати
  - Софтверске метрике
- …

План курса

- Развој софтвера
- **Елементи програмског језика *C++***
  - Класе, наслеђивање, апстрактне класе и методи
  - Динамичке структуре података на језику *C++*
  - Шаблони
  - Функционално програмирање на п.ј. C++
- …

План курса

- Развој софтвера
- Елементи програмског језика *C++*
- **Дизајн и архитектура софтвера**
  - Увод у пројектовање софтвера
  - Принципи пројектовања софтвера
  - Обрасци за пројектовање
  - Рефакторисање
  - Развој вођен догађајима
- …

План курса

- Развој софтвера
- Елементи програмског језика C++
- Дизајн и архитектура софтвера
- **Изабране савремене методе и технике развоја**
  - Развој вођен тестовима
  - Агилни развој софтвера
  - Конкурентно програмирање

Обавезе студената

- Групни пројекат
  - 40 поена (обавезан)
  - део поена у току семестра
  - већина после завршне одбране
  - детаљи следе ускоро
- Завршни испит – писмено-усмени
  - Услов за излазак на испит:
    - завршен и одбрањен пројекат (најмање 16 поена)
  - Писмени део – 40 поена (20 задаци и 20 теорија)
    - > 50% поена на писменом делу испита
    - > 50% поена на задацима
  - Усмени део – 20 поена
- Могуће су измене плана и усклађивање са околностима

Обавезе студената

- Присуство на часовима је веома пожељно и препоручљиво, али НИЈЕ обавезно
  - часови служе онима који желе да нешто науче
  - повремено пописивање присутних
  - нема негативних поена за недолажење
- Пристојно понашање на часовима ЈЕСТЕ обавезно
  - коме се не ради, не сме да омета друге
  - ИМА негативних поена за
    - разговарање
    - доручковање
    - читање новина
    - употребу рачунара и мобилних телефона
    - и све друге видове ометања рада

Литература за курс

- Саша Малков, **Развој софтвера**, Математички факултет, 2023
- E.Gamma, R.Helm, R.Johnson, J.Vlissides, **Design Patterns: Elements of Reusable Object Oriented Software**, Addison-Wesley, 1995.
- Martin Fowler, **Refactoring: Improving the Design of Existing Code**, Addison-Wesley, 1999.
- Stanley B. Lippman, Josee Layoie, Barbara Moo, **C++ Primer, 4thed.**, Addison-Wesley, 2005.
- Robert C. Martin, **Agile Software Development: Principles, Patterns and Practices**, Prentice Hall, 2003.
- Саша Малков, **ООП-C++ кроз примере**, Математички факултет, 2007: www.matf.bg.ac.rs/~smalkov/razno#knjige
- Веб локација наставника: www.matf.bg.ac.rs/~smalkov
- Веб локације асистената
- Друга литература по препоруци наставника и асистената

ВАЖНО !!!

- Презентације коришћене на предавањима **нису** литература за учење
- Оне могу да служе само као подсетник или сажети увод

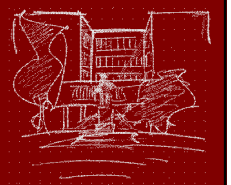
Не брините, није страшно...

...али МОРА озбиљно да се ради

[P290]

Развој софтвера

Саша Малков



Тема 1

Проблеми при развоју и њихово решавање

Проблеми при развоју софтвера

- Проблем је све оно што негативно утиче на процес и резултат развоја ИС
- Основно мерило неуспеха је изгубљена материјална вредност – плаћена цена неуспеха
  - уложена средства
  - изгубљено време
  - последице по читав пословни систем

Примери неуспеха (1)

- Највећи британски снабдевач храном, *J.Sainsbury PLC*
  - октобра 2004 је отписао инвестицију од $526.000.000 у аутоматизацију ланца снабдевања
  - поред тога је морао да запосли 3000 додатних радника за мануелно обављање посла
  - (*Charette, 2005*)

Примери неуспеха (2)

- Годишња процена трошкова услед неуспеха у САД је $81.000.000.000
  - то је око 1/3 свих улагања у ИС
  - слично је било у ВБ, где је процена укупног улагања £33.600.000.000
  - (*Standish Group 1995*)

Врсте неуспеха

- Прекорачење трошкова
- Прекорачење временских рокова
  - трошкови због продуженог развоја
  - трошкови због кашњења пуштања у рад
- Резултат није употребљив (у планираним размерама)
  - Систем је имплементиран у складу са захтевима, али не одговара стварним потребама
  - Неиспуњеност нефункционалних захтева
- Одустајање од пројекта
  - услед неког од претходних разлога (или више њих)
- Катастрофалне грешке (багови)

Примери неуспеха (3)

- Федерална управа летења у САД
  - 1981. започела пројекат унапређења система за контролу лета.
  - 1994 се одустало од пројекта.
  - Предвиђени трошкови су до тада утростручени.
  - Уложено је више од $2.600.000.000.
  - Укупни губици уз неостварену планирану добит се процењују на више од $50.000.000.000.
  - (*Charette, 2005*)

Неупотребљив резултат

- Чак и када постоји планирање и када се пројекат имплементира по плану, резултат може да буде неупотребљив или да представља неуспех

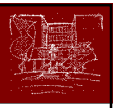
Неупотребљив резултат (2)

- Аспекти “неупотребљивости”
  - Неупотребљив кориснички интерфејс
  - Процедура коришћења није остварива / добра у реалним условима

Неупотребљив резултат (3)

- Аспекти “неупотребљивости”
  - Неупотребљив кориснички интерфејс
    - лоше решени ергономски аспекти
    - непостојање физичког (хардверског) одзива
    - неинтуитиван изглед корисничког интерфејса
    - проблематичне контроле интерфејса
    - спор одзив
    - непоузданост
    - …
  - Процедура коришћења није остварива / добра у реалним условима

Неупотребљив резултат (4)



- Аспекти “неупотребљивости”
  - Неупотребљив кориснички интерфејс
  - Процедура коришћења није остварива или добра у реалним условима
    - Уз купљену књигу се добија бесплатна свеска, али то мора да се заведе као продаја. Ако систем не омогућава мануелну промену цене, то неће бити могуће. Ако систем не допушта да се цена мануелно постави на 0, такође неће бити могуће.
    - Претраживање књига (или сличног каталога) које подразумева уношење тачног наслова.
    - Ако ИС омогућава управи да тачно види и критикује магационере због нерационалног заузећа простора, а не омогућава аутоматску подршку у виду савета, магационери почињу да троше несразмерно много времена на планирање простора. Иако резултат јесте боља искоришћеност простора, губи се на већем утрошку радног времена.

Неупотребљив резултат (5)

- Могући узроци
  - Нереални или нејасни циљеви пројекта
  - Непрецизна процена потребних ресурса
  - Лоше дефинисани захтеви
  - Слаба комуникација између клијента, развијаоца и корисника

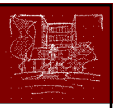
Примери неуспеха (4)

- Планиран информациони систем за лондонску службу хитне помоћи је одбачен убрзо после испоруке 1992. године.
  - Уместо да убрза, испоставило се да је успорио рад и умањио ефикасност
  - После више критичних грешака и непотребних смртних случајева, систем је окривљен за проблеме.
  - Укупна улагања су процењена на £43.000.000
  - (*Bennett, 2002*)

Катастрофални багови

- *Mars Global Surveyor* (*NASA*), 1996.
  - проблем: дошло је до оштећења батерија
  - узрок: грешке у обради земаљских команди и извештавању
    - требало је померити соларне панеле
    - систем је послао обавештења о неким проблемима али је известио да је на крају све урађено како је захтевано
    - ипак, није све урађено по плану и батерије су биле непосредно изложене Сунчевом зрачењу, што је проузроковало њихово отказивање 5 месеци касније
  - цена: губитак уређаја вредног око $154.000.000
  - напомена: и поред тога, мисија је већ трајала 4 пута дуже него што је било предвиђено, па се на крају сматра за веома успешну

Катастрофални багови (2)



- Неуспешно лансирање ракете Аријана 5, 4. јун 1996.
  - проблем: 37 секунди након узлетања ракета је скренула са планиране путање, због чега је дошло до аутоматског самоуништења
  - узрок: грешка у софтверу
    - није на одговарајући начин обрађено прекорачење у целобројним операцијама
  - цена: губитак ракете и модула, у вредности од око $370.000.000
- *Mars Climate Orbiter* (*NASA*), 1999
  - проблем: квар при слетању
  - узрок: неки модули су радили са метричким а неки други са империјалним јединицама за силу
  - цена: око $125.000.000
  - напомена: неки текстови тврде да је у питању људска грешка у процесу управљања

Катастрофални багови (3)

- *Therac-25* уређај за терапију зрачењем, 1985.
  - проблем: масовна појава неодговарајућих емисија зрачења
  - узрок: грешка у софтверу наслеђеном из верзије *Therac-20*
    - грешка се није испољавала у старијем моделу због уграђеног аутоматског хардверског ограничења
    - нико није благовремено проверавао колико је често то ограничење у пракси морало да се активира
  - цена: најмање 5 смртних случајева
- *Multidata Systems International*, 2000.
  - проблем: слично као код *Therac-25*
  - узрок: грешка у софтверу за прорачунавање потребне дозе зрачења
  - цена: најмање 8 смртних случајева и још 20 критичних оштећења

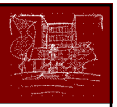
Катастрофални багови (4)

- Процесор *Intel Pentium* (1994)
  - проблем: грешка при дељењу
  - узрок: у процесор је уграђена неисправна таблица оптимизација
  - цена: око $475.000.000
- Скуп чипова за матичне плоче за процесор *Intel Sandy Bridge* (2011)
  - проблем: грешке у комуникацији између процесора и уређаја и дискова
  - узрок: ???
  - цена: процењује се између $300.000.000 и $700.000.000

Узрочници проблема

- Проблеми на страни клијента
- Проблеми на страни развијалаца
- Вишестрани проблеми

Узрочници проблема (2)



- **Проблеми на страни клијента**
  - Нереални или нејасни циљеви пројекта
  - Неусклађеност циљева и стратегије
  - Политика улагача
  - Комерцијални притисак
  - Отпор корисника према примени новог софтвера
- Проблеми на страни развијалаца
- Вишестрани проблеми

Узрочници проблема (3)

- Проблеми на страни клијента
- **Проблеми на страни развијаоца**
  - Слабо вођење пројекта
  - Непрецизна процена потребних ресурса
  - Слабо извештавање о стању пројекта
  - Неуправљани ризици
  - Употреба “незрелих” технологија
  - Неспособност да се изнесе сложеност пројекта
  - Ток практичног развоја без чврстих принципа и правила
- Вишестрани проблеми

Узрочници проблема (4)

- Проблеми на страни клијента
- Проблеми на страни развијаоца
- **Вишестрани проблеми**
  - Слаба комуникација између клијента, развијаоца и корисника
  - Неповерење између клијента и развијаоца
  - Лоше дефинисани захтеви

Узрочници проблема (5)

- Наравно, обично иза неуспеха стоји више разлога
- Сваки од проблема на страни развојног тима има корене међу руководиоцима развоја
- И остали проблеми се често могу на време препознати и отклонити од стране руководиоца развоја.

Лоше планирање

- Лоше планирање је један од најчешћих узрока неуспеха
- Лоше планирање има два основна облика:
  - недовољно добро планирање
  - претерано планирање

Недовољно планирање

- Недовољно планирање се обично огледа кроз:
  - недовољну анализу проблема
  - лоше и/или непрецизно дефинисане захтеве
- Најчешће последице су:
  - несразмерно велики број накнадних корекција захтева
  - слаба употребљивост решења
  - пробијање рокова

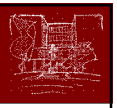
Примери неуспеха (5)

- *Sydney Water Corp*, највећи снабдевач водом у Аустралији, уводио је систем за информисање и наплату 2002.
  - развој је напуштен након уложених 61.000.000 АУД
  - касније анализа је показала да су узрок неуспеха лоше планирање, бројне измене захтева и непрецизне спецификације
  - (*Charette, 2005*)

Претерано планирање

- Можда изгледа парадоксално, али веома чест узрок проблема је вид претераног планирања
- Претерано планирање се обично огледа кроз:
  - прешироко и неконцентрисано улажење у пројекат
    - сувише дубока и обимна анализа са раним детаљним пројектовањем
  - прешироко улажење у имплементацију
  - преамбициозност, несразмерно реалним могућностима
- Најчешће последице су:
  - касно уочавање пропуста
  - отежана транзиција
  - пробијање рокова

Примери неуспеха (6)



- Аустралијска царина је развијала интегрисани систем за управљање товаром
  - Написана је пројектна документација на 19.000 страна:
    - 800 екранских форми
    - 16.000 пословних правила
    - 70 сложених пословних порука
    - база података са 850 табела
    - 3700 извршних модула
    - 1800 типова трансакција
    - 55 пакетних обрада
    - 90 извештаја
    - (*http://www.acs.org.au/iage/200908/*)
  - пројекат је процењен на $33.000.000
  - утрошено је $240.000.000
  - да би се установило да није могуће направити транзицију са старог на нови систем нити их користити паралелно

Управљање ризицима

- “Управљање ризицима је процес препознавања, процењивања и контролисања свега онога што би у пројекту могло да *крене наопако* пре него што постане претња успешном довршавању пројекта или имлементације информационог система.”
  - (*Whitten, 2001*)
- У употреби су и термини
  - “анализа ризика”
  - “инжењеринг ризика”

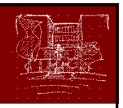
Управљање ризицима и РС

- Управљање ризицима није везано само за развој развој софтвера већ и за сваки други сложен развојни / градитељски процес
- У случају развоја софтвера
  - Највећи значај има добро управљање ризицима у почетним фазама
  - Отежано је у случају дугачких развојних циклуса

Извори ризика

- 10 ризичних чинилаца и технике управљања (*Boehm,1991*):
  - мањак особља
  - нереални рокови и буџет
  - развој погрешних функција
  - развој погрешног интерфејса
  - претеривање (“позлаћивање”)
  - непрекидни низ измена у захтевима
  - слабости екстерно реализованих послова
  - слабости у екстерно набављеним компонентама
  - слабе перформансе у реалном раду
  - рад на границама рачунарских наука

Прилагођавање развојних методологија



- Неки од узрока проблема се могу отклонити измењеним приступом проблему развоја
- Савремене развојне методологије почивају на концептима који омогућавају смањивање ризика

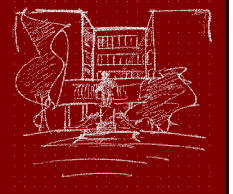
Литература за тему

- *Robert N. Charette*, ***Why Software Fails***, *IEEE Spectrum, September 2005*
- *Barry W. Boehm*, ***Software Risk Management: Principles and Practices***, *IEEE Software 8(1), 1991*
- *Simon Bennett, Steve McRobb, Ray Farmer*, ***Object Oriented Systems Analysis and Design (Using UML)***, *McGraw Hill, 2002*

[P290]

Развој софтвера

Саша Малков



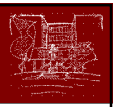
Тема 2

Развој објектно оријентисаних методологија

Савремени концепти развоја

- Неки од најважнијих савремених коцепата развоја софтвера који значајно смањују ризике су:
  - Инкрементални развој
  - Одређивање корака према роковима
  - Појачана комуникација међу субјектима
  - У жижи су објекти а не процеси
  - Прављење прототипова

Инкрементални развој



- Принцип:
  - Уместо да се развоју приступа као великом целовитом послу, са сложеним пројектом и имплементацијом, посао се дели у низ инкременталних фаза посвећених мањим целинама посла
- Основна добит:
  - Свака од фаза је значајно мања и једноставнија него читав посао, чиме се поједностављују планирање и имплементација
  - Већа је тачност планирања трошкова и рокова и тачније извештавање о току развоја
- Основни ризик:
  - Ако се у првим корацима потпуно занемаре предстојећи, постоји ризик да ће у нередним корацима бити потребне веће измене
  - Ако се у првим корацима узму у обзир сви предстојећи, постоји ризик да се њихова сложеност приближи сложености читавог система (“надувавање корака”), чиме овакав приступ губи смисао
  - Често није могуће сагледати унапред број, цену и укупно трајање свих корака

Одређивање корака према роковима

- Принцип:
  - За сваки инкрементални корак се прво одреде буџет и рокови, а тек после тога послови који ће кораком бити обухваћени
- Основна добит:
  - Драстично смањивање ризика од прекорачивања буџета или рокова
  - Успостављање ритма редовног испоручивања нових верзија система
    - Редовнија контрола квалитета и виши степен поверења између субјеката
    - Постепено прилагођавање корисника новим елементима система
- Основни ризик:
  - Успостављање тесних оквира инкременталног корака може отежати поједине кораке и подићи укупну цену и трајање развоја
    - Неки елементи система се не могу природно поделити у различите кораке
    - Сваки корак захтева трошкове испоручивања што у збиру може да постане велика ставка у случају великог броја малих корака
  - У првим корацима се обично бирају послови који доносе већу добит. При крају развоја постоји ризик да се не имплементирају послови "чија је цена већа од добити" иако су значајни за систем као целину

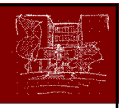
Појачана комуникација међу субјектима

- Принцип:
  - У планирању (и сваког појединачног корака) се укључују у разматрање све врсте субјеката у што већем броју
- Основна добит:
  - Добија се тачнија слика о потребним циљевима из различитих углова
  - Смањује се ризик развоја неупотребљивог решења
  - Субјекти се кроз процес развоја припремају за употребу
- Основни ризик:
  - Превише информација може довести до претераног планирања, а тиме и до "надувавања" појединачних корака
  - Превеликим разматрањем мишљења субјеката који су навикли на постојеће процесе и не сагледавају планиране измене може се смањити обим суштинских функционалних измена у оквиру корака , а тиме и непотребно повећати број корака да се дође до "коначног" решења

У жижи су објекти а не процеси

- Принцип:
  - У средиште пажње се при развоју стављају објекти а не процеси
- Основна добит:
  - Објекти су обично стабилнији од процеса (тј. измене у начину пословања се мање одражавају на објекте него на процесе)
  - Такав приступ је прилагођенији инкременталном развоју
  - Смањује се ризик од погрешних одлука у раним фазама развоја
    - у раним корацима се више пажње посвећује објектима
    - у каснијим корацима се више пажње посвећује процесима
- Основни ризик:
  - Потпуно занемаривање процеса у раним фазама развоја може водити погрешној архитектури система, што се касније веома тешко (скупо) мења
  - Сасвим детаљно разматрање процеа у раним корацима прети да доведе до "надувавања" првих корака

Прављење прототипова



- Принцип:
  - У оквиру анализе проблема се прави прототип који одражава начин фунцкионисања и употребе софтвера
- Основна добит:
  - Олакшава се не-техничким субјектима да у раним фазама развоја уоче одређене недостатке
  - Смањује се ризик од погрешних одлука у раним фазама развоја
- Основни ризик:
  - Прототипови обично одражавају функционалне аспекте и елементе корисничког интерфејса, али не и унутрашњу структуру софтвера. Последица је да су само делимично одговарајући ОО методологијама
  - Недовољно широко направљен прототип и недовољно широко разматрање прототипа могу да прикрију недостатке у другим аспектима ИС
  - Превише пажње посвећене прототипу прети да "надува" фазу његове израде

Околности развоја ООМ

- Постојање методологија које структурирано приступају анализи и описивању процеса
- Потреба за структурираним описивањем података
- Потреба за описивањем ентитета који мењају стања
- Подизање нивоа апстрактности посматрања елемената система
- Програмирање управљано догађајима
- Визуелни кориснички интерфејси
- Повећана модуларност софтвера
- Скраћивање развојног циклуса
- Транзиција модела
- Вишеструка употребљивост софтвера
- и друго

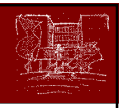
Објектно оријентисане методологије

- За разлику од структурних, које су у средиште пажње стављале уређивање процеса и алгоритама, ОО методологије у средиште пажње стављају уређивање објеката којима се описује систем
- Развој објектно оријентисаних методологија је почео у време када су слабости претходних методологија биле углавном познате
- У њих су уграђени неки од представљених савремених концепата развоја

ООМ и ООП

- ОО методологије и ОО програмирање *нису исто*
- ОО методологије користе ОО приступ разматрању проблема, анализирању и моделирању пројеката
- ОО програмирање користи ОО приступ при имплементирању програма на ОО програмским језицима
- У оквиру ООМ могу да се примењују и ПЈ који нису ОО
- ООП може да се користи и ако се не користи ООМ

Основни концепти ООМ



- Основни концепти Објектно оријентисаних методологија
  - У жижу стављају објекте, а не процесе
  - Све ООМ се одликују скраћивањем трајања развојних циклуса
    - РУП (и друге) прописују инкрементални развој
    - Агилне методологије прописују одређивање корака према роковима и трошковима
  - Појачана комуникација међу субјектима у свим фазама развоја

Појам објекта

- Шта је објекат?
- Објекат је апстракција нечега у домену проблема, која описује способност система да о томе чува информацију, интерагује са тиме или обоје.
  - (*Coad, Yourdon*, 1990)
- Објекат је концепт, апстракција или нешто са јасним границама и смислом у односу на конкретан проблем. Објекат има две сврхе: да помогне размевању стварног света и пружи практичну основу за рачунарску имплементацију.
  - (*Rumbaugh at al.*, 1991)

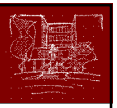
Појам објекта (2)

- Шта је објекат?
- Објекти се описују одговорима које могу да дају на питања:
  - Ко сам ја?
  - Шта могу да урадим?
  - Шта знам?
    - (*Wirf-Brock*, 1990)
- Објекти имају стање, понашање и идентитет
  - (*Booch*, 1994)

Појам објекта (3)

- Резиме:
  - Објекат је апстракција нечег конкретног у домену проблема
  - Објекат мора да има стање, понашање или знање
    - (не обавезно све)
  - Објекат има животни циклус
    - настајање
    - постојање
    - нестајање

Појам класе



- Класа је апстракција скупа објеката који имају “довољно сличности”
  - Као критеријум сличности се превасходно узима у обзир понашање
  - Стање и знање објеката не утиче значајно на њихову класификацију
  - Објекат представља конкретан примерак класе

Дефиниција ОМГ

- “Класа је опис скупа објеката који деле исте атрибуте, операције, методе, односе и семантику. Сврха класе је да декларише колекцију метода, операција и атрибута која у потпуности описује структуру и понашање тих објеката.”
- “Објекат је примерак који потиче из класе, структуриран је и понаша се у складу са својом класом.”

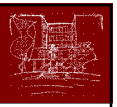
Атрибути и методи

- Атрибути су средство за описивање
  - стања
  - знања
- Методи су средство за описивање
  - понашања

Објекти и класе данас

- Класе се формално дефинишу као скупови свих објеката који:
  - Имају прописане атрибуте
    - одговорајућег имена и типа
  - Могу да примене прописане методе
    - одговарајућег имена и типа
  - Задовољавају одговарајуће формалне услове
    - семантика атрибута и метода

У програмским језицима



- У већини савремених програмских језика се распознавање да ли неки објекат припада некој класи одвија прагматично, без примене ад-хок правила
- Припадност објекта класи
  - по правилу се одређује у тренутку прављења објекта
  - не може се мењати током живота објекта
- Напомена:
  - Постоје п.ј. код којих се припадност класи установљава динамички, проверавањем семантичких правила

У програмским језицима (2)

- У већини савремених програмских језика *класа* представља *тип*
- Објекти *припадају* класи ако *имају* тип класе
- Напомена:
  - Постоје п.ј. код којих класе нису типови, већ скупови типова који задовољавају дата семантичка правила, на пример *Haskell*, или *концепти* у *C++*-у
  - Код неких језика та правила се динамички проверавају, па објекти могу да мењају класу тоом живота, на пример *D* (*Date, Darween*)

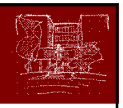
Основни концепти ООМ

- Енкапсулација
- Интерфејс
- Полиморфизам
- Наслеђивање
  - Специјализација и генерализација
  - Хијерархије класа

Објектни и класни језици

- Већина тзв. ОО програмских језика би заправо могли боље да се називу *класни језици*
- Зато што постоје ООПЈ који немају концепт класе, па им боље пристаје назив *објектни језик*

Енкапсулација



- Структура објеката је њихова интерна ствар
  - не сме се излагати спољашњем свету
  - атрибутима се сме приступати само посредно
- Постоје случајеви када неки атрибути представљају основу понашања објеката или класа
  - тада им се може омогућити јавни приступ
  - и тада је ипак препоручљиво енкапсулирање
- Сврха
  - Апстраховање структуре методима
  - Виши ниво међусобне независности класе од модула у којима се употребљава

Интерфејс

- Објекат (класа) пружа спољашњим корисницима само скуп метода путем којих могу комуницирати са њим
- Такав скуп метода се назива *јавни интерфејс* објекта (класе)
- Интерфејс се обликује тако да омогући обављање једног целовитог посла

- Сврха
  - Суштина објеката је у њиховом понашању. Интерфејс омогућава то понашање.
  - Сужавањем интерфејса на суштину функционисања објекта прикрива се сва сложеност имплементације и пружа виши ниво независности објекта од окружења

- Напомена
  - Ако објекат има више интерфејса, значи да има више функција, па је потребно размотрити његово разлагање на више објеката

Полиморфизам

- Полиморфизам подразумева да се једном написан код може употребљавати за различите врсте објеката
- Постоје три основне врсте полиморфизма: хијерархијски, параметарски и имплицитни
  - Сви ОО програмски језици омогућавају хијерархијски полиморфизам
  - Неки савремени ООПЈ омогућавају параметарски полиморфизам
  - Неки језици подржавају имплицитни полиморфизам

- Сврха
  - Полиморфизам омогућава писање апстрактнијег кода, који има високу употребљивост

Наслеђивање

- Наслеђивање класе је еквивалентно увођењу једносмерне парцијално уређене релације “јесте” између класа
  - Класа А “јесте” класа Б акко сваки објекат класе А има све особине које имају и објекти класе Б
- Ако класа А “јесте” класа Б каже се и да је
  - А “изведена” класа из Б или А “је потомак” класе Б
  - Б “основна” класа за А или Б “је предак” класе А
- Релација “јесте” је парцијална релација поретка (рефлексивна и транзитивна)
  - Представља основу за грађење хијерархија класа

- Сврха:
  - Наслеђивање се користи за експлицитно означавање сличности међу класама (објектима)
  - Представља основу за хијерархијски полиморфизам: ако се нешто може урадити са објектом класе Б, онда се то може урадити и са сваким објектом класе А која је изведена из Б

Специјализација и генерализација

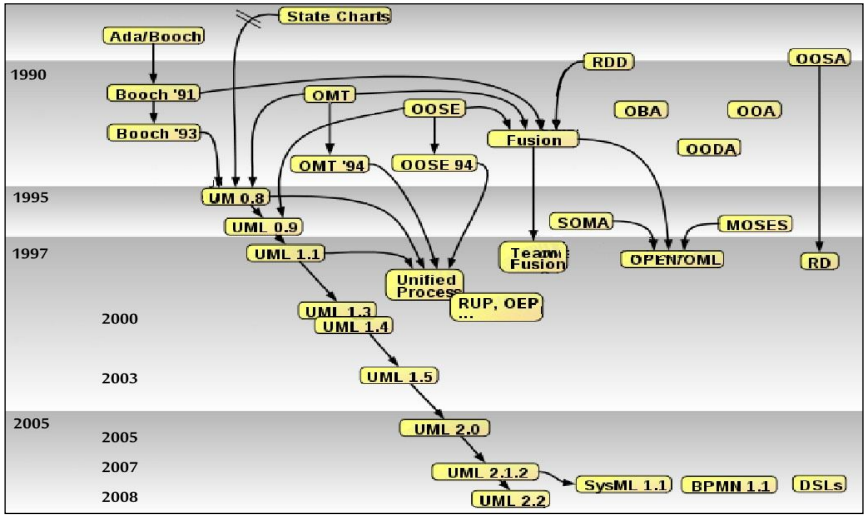
- Наслеђивање се посматра у два смера, као
  - специјализација или
  - генерализација

- Ако класа А “јесте” Б, а класа Б “није” А онда:
  - класа А је посебан (специјалан) случај класе Б
  - класа Б је општији (генералан) случај класе А

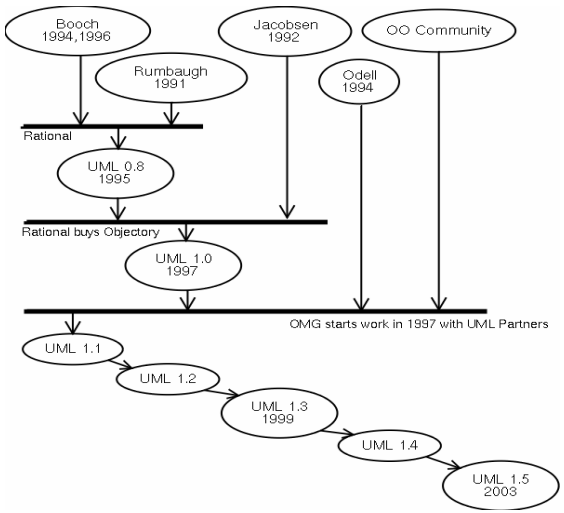
Преглед ОО методологија

- Почетни кораци (-1997)
- Обликовање *UML*-а (1995-2005)
- Пост-*UML* кораци (2000-)

Преглед ОО методологија



Историја УМЛ-а



Преглед ОО методологија (1)

- **Почетни кораци (-1997)**
  - Велики број различитих нотација и методологија
  - Ниједна није довољно широка и комплетна
    - (бар гледано из данашњег угла)
  - *Booch, 1991.*
  - *Coad, Yourdon, 1991.*
  - *Martin, Odell, 1992.*
- Обликовање *УМЛ*-а (1995-2005)
- Пост-*УМЛ* кораци (2000-)

Преглед ОО методологија (2)

- Почетни кораци (-1997)
- **Обликовање *УМЛ*-а (1995-2005)**
  - Неколико дубљих методологија концентрисаних на различите фазе развоја
  - Покушаји уједначавања нотације
  - Акценат на нотацији
- Пост-*УМЛ* кораци (2000-)

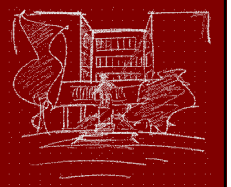
Преглед ОО методологија (3)

- Почетни кораци (-1997)
- Обликовање *УМЛ*-а (1995-2005)
- **Пост-*УМЛ* кораци (2000-)**
  - Уједначена нотација
  - Широко схватање процеса развоја
  - Потпуно посвећивање методологијама
  - Агилне методологије
  - РУП и друге савремене методологије

[P290]

Развој софтвера

Саша Малков



Тема 3

УМЛ

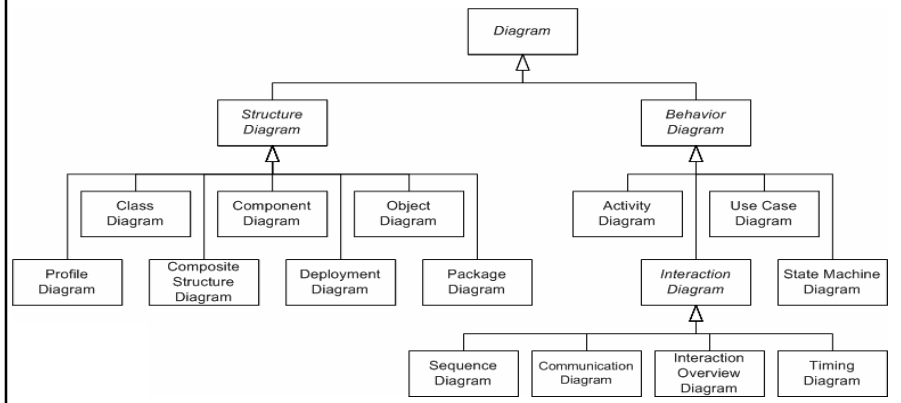
УМЛ

- Обједињени језик за моделирање (*Unified Modeling Language - UML*)
- Сам језик не представља методологију, али је обликован према паралелно развијаној методологији
  - "Обједињени приступ" (*Unified Approach*)
  - данас се зове РУП (*Rational Unified Approach*)
- Сам језик је развијен, публикован и шире прихваћен пре одговарајуће методологије

УМЛ – Врсте дијаграма

- Дијаграми се деле у три групе:
  - дијаграми понашања
  - дијаграми интеракције
  - структурни дијаграми

УМЛ – Врсте дијаграма



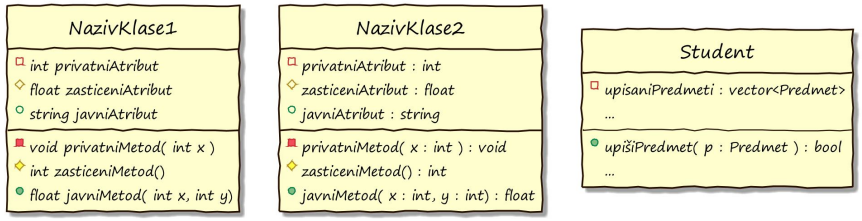
УМЛ – Структурни дијаграми

- Дијаграм класа (*class diagram*)
- Дијаграм компоненти (*component diagram*)
- Дијаграм објеката (*object diagram*)
- Дијаграм профила (*profile diagram*)
- Дијаграм сложене структуре (*composite structure diagram*)
- Дијаграм испоручивања (*deployment diagram*)
- Дијаграм пакета (*package diagram*)

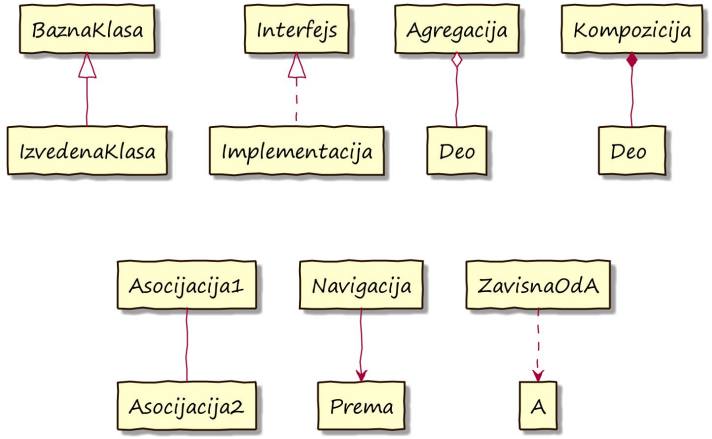
УМЛ – Дијаграм класа

- Илуструје елементе статичног модела, као што су класе, њихов садржај и међусобне односе
- Садржи
  - називе класа
  - атрибуте класа
  - методе класа
  - специјализацију и генерализацију
  - односе
    - асоцијацију
    - агрегацију
    - композицију

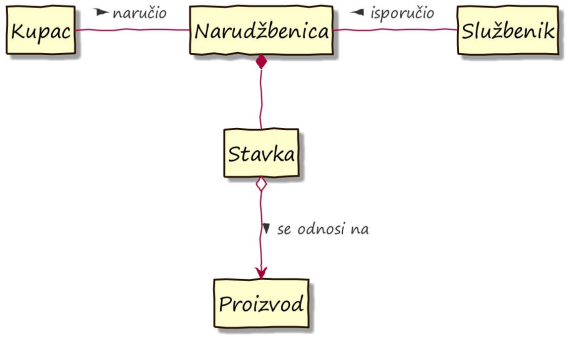
УМЛ – Дијаграм класа – пример 1



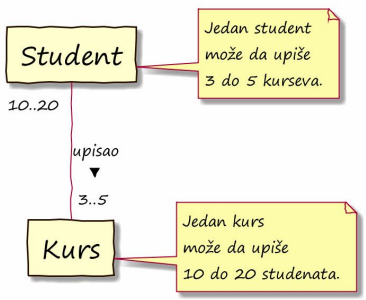
УМЛ – Дијаграм класа – пример 2 – Односи међу класама



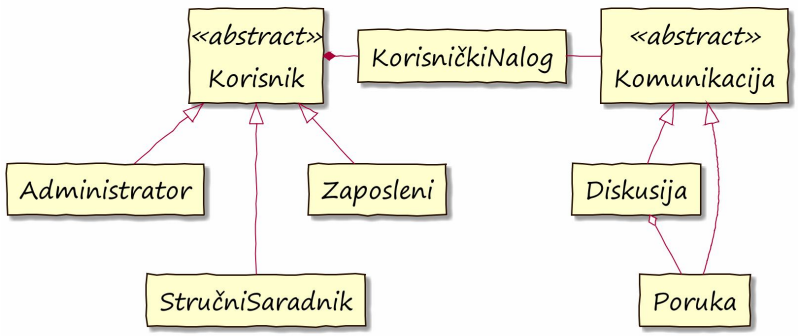
УМЛ – Дијаграм класа – пример 3 – Именовање односа



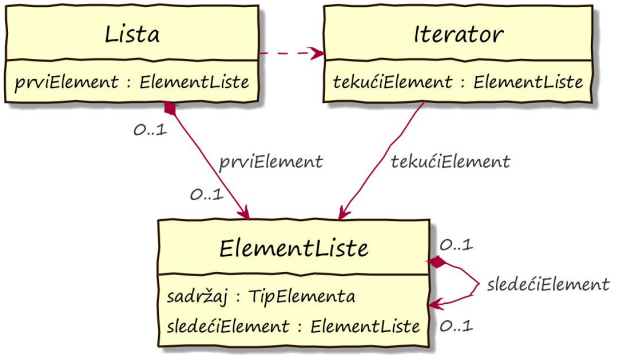
УМЛ – Дијаграм класа – пример 4 – Кардиналност односа



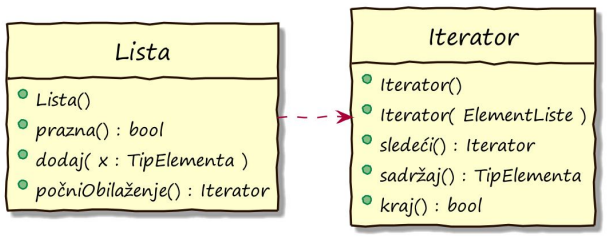
УМЛ – Дијаграм класа – пример 5 – Стереотипови



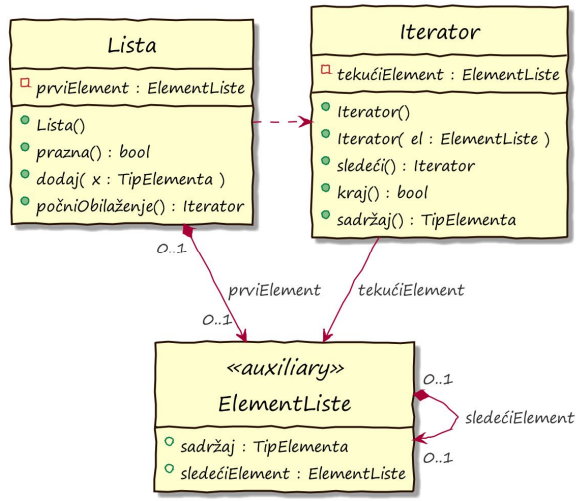
УМЛ – Дијаграм класа – пример 6 – Дијаграм модела домена



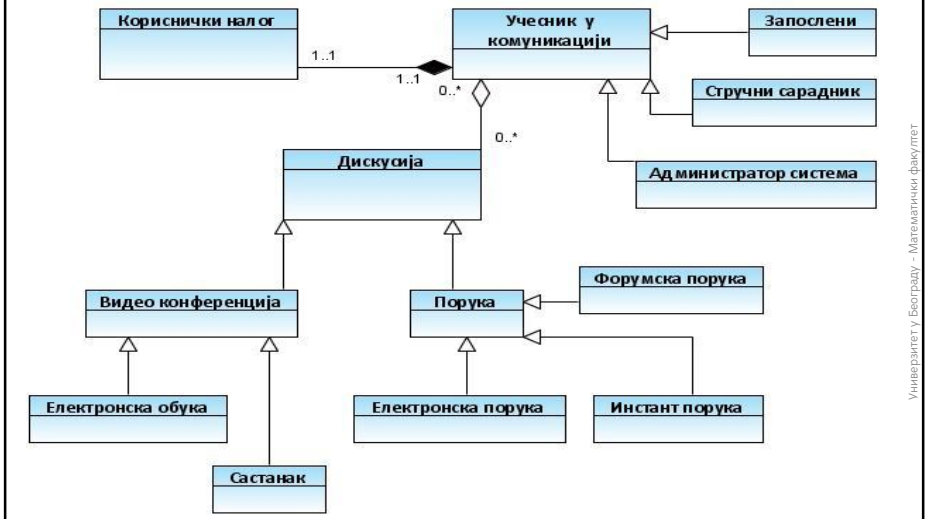
УМЛ – Дијаграм класа – пример 7 – Дијаграм интерфејса



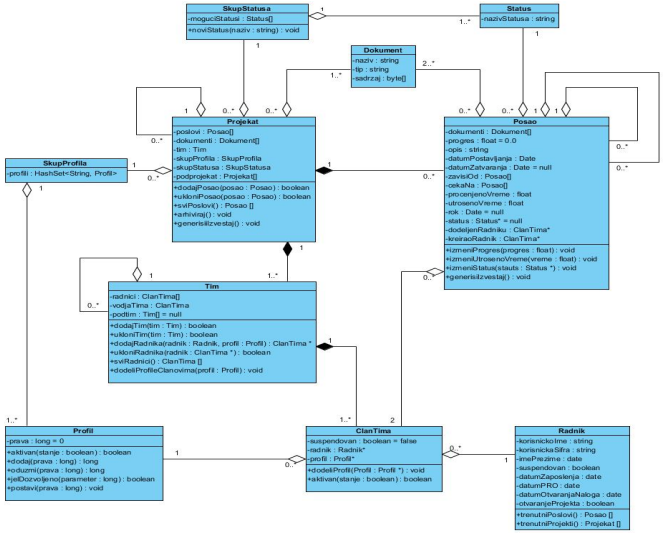
УМЛ – Дијаграм класа – пример 8 – Дијаграм имплементације



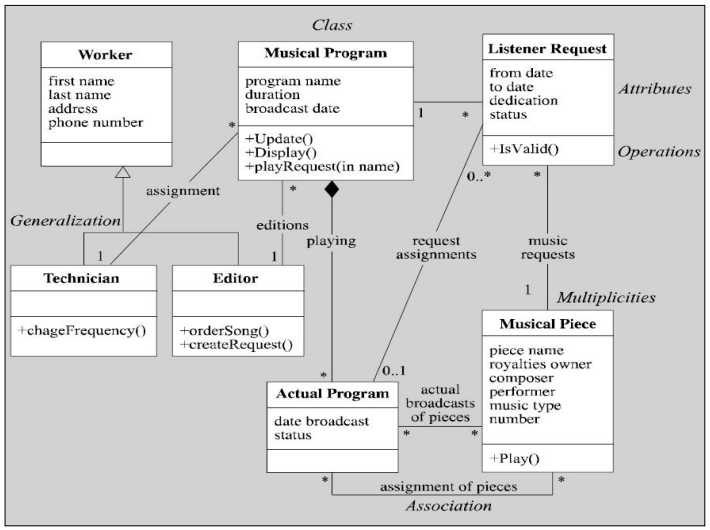
УМЛ – Дијаграм класа – пример 9



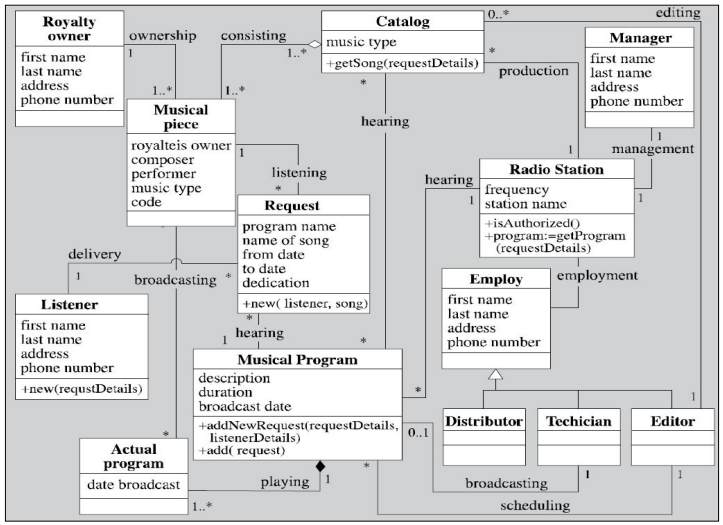
УМЛ – Дијаграм класа – пример 10



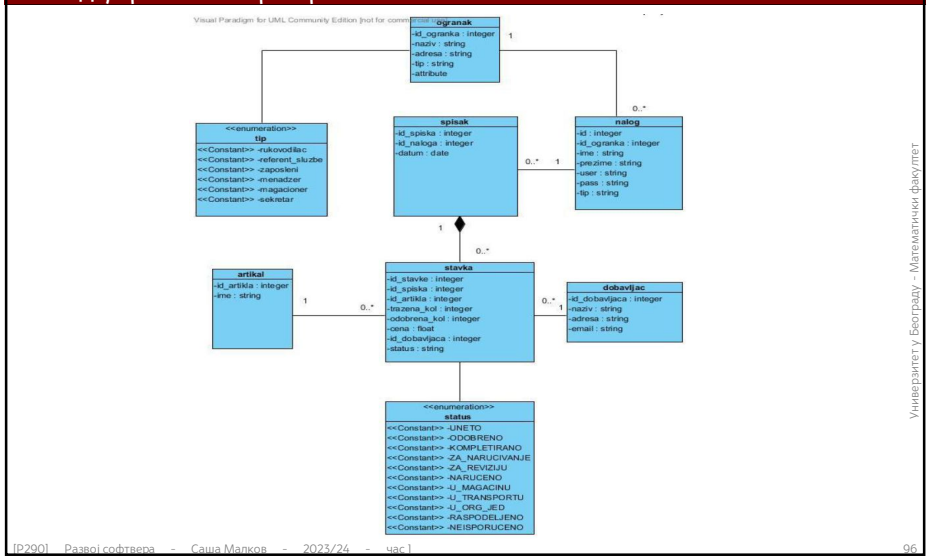
УМЛ – Дијаграм класа – пример 11



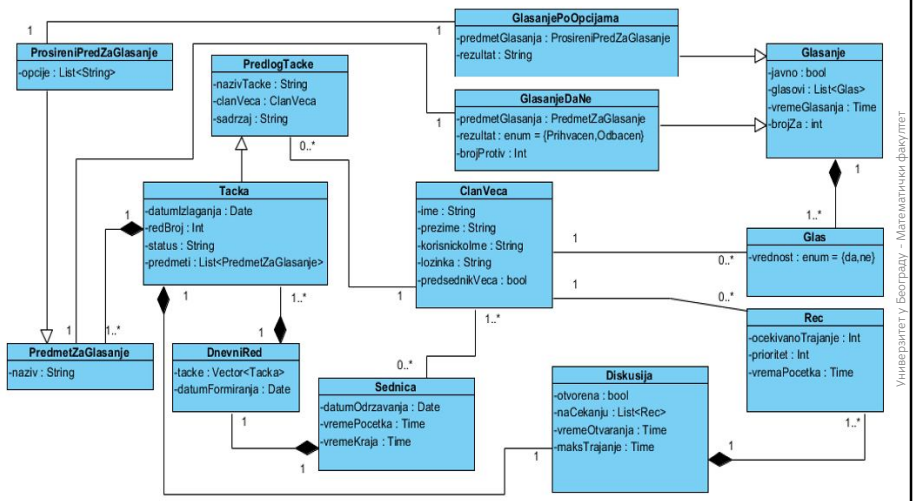
УМЛ – Дијаграм класа – пример 12



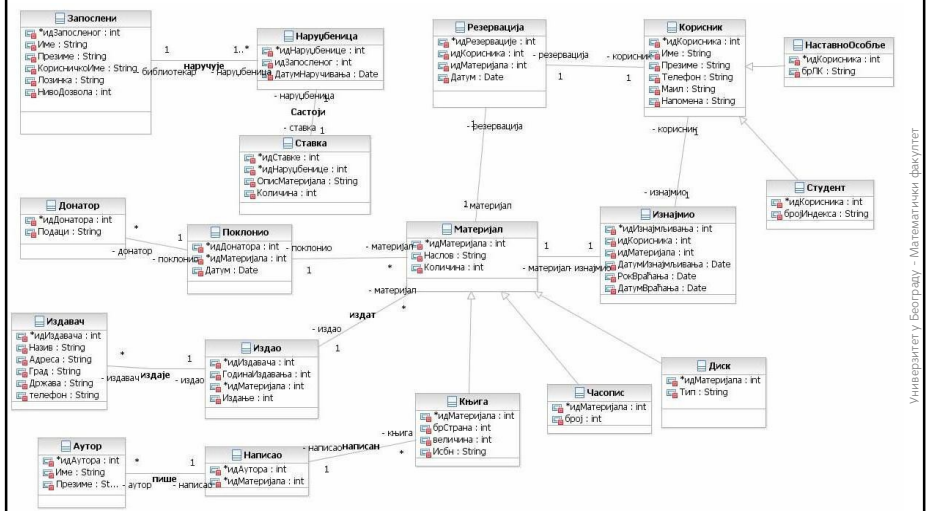
УМЛ – Дијаграм класа – пример 13



УМЛ – Дијаграм класа – пример 14



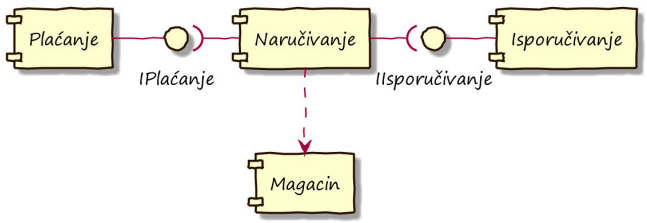
УМЛ – Дијаграм класа – пример 15



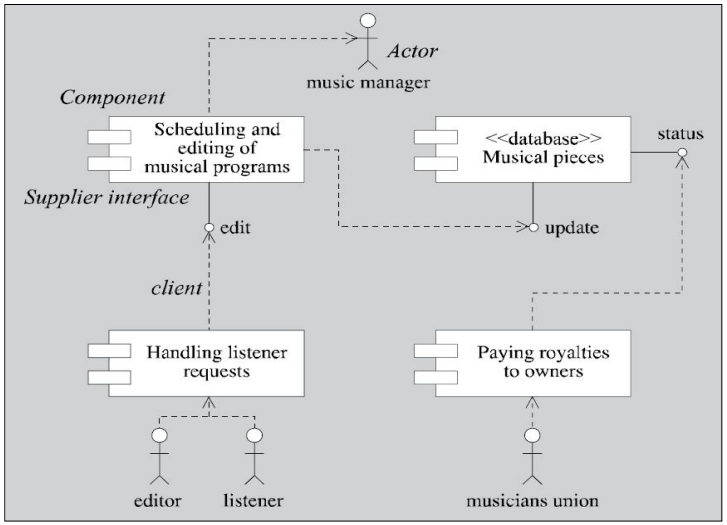
УМЛ – Дијаграм компоненти

- Илуструје компоненте које чине апликацију, систем или организацију.
- Садржи
  - називе компоненти
  - њихове међусобне односе
  - јавне интерфејсе

УМЛ – Дијаграм компоненти – пример 1



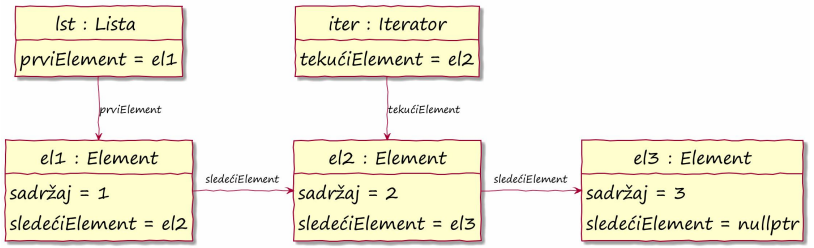
УМЛ – Дијаграм компоненти – пример 2



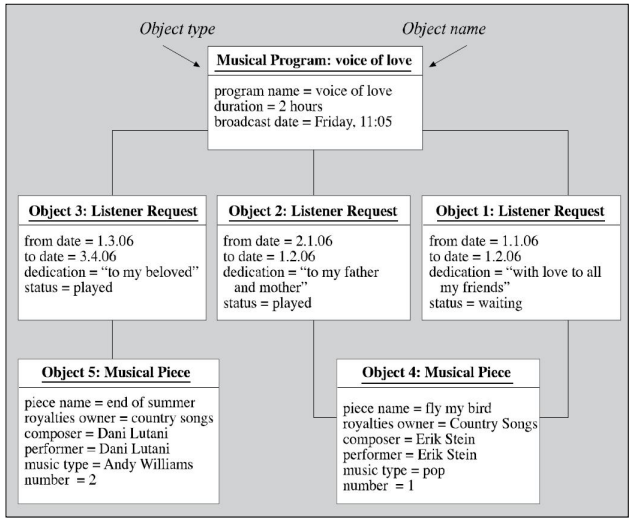
УМЛ – Дијаграм објеката

- Представља објекте и њихове односе у једном тренутку времена. Користи се као допуна дијаграма класа и комуникације за описивање динамичких система.
- Садржи
  - називе класа
  - називе објеката
  - имена и вредности атрибута
  - односе

УМЛ – Дијаграм објеката – пример 1



УМЛ – Дијаграм објеката – пример 2



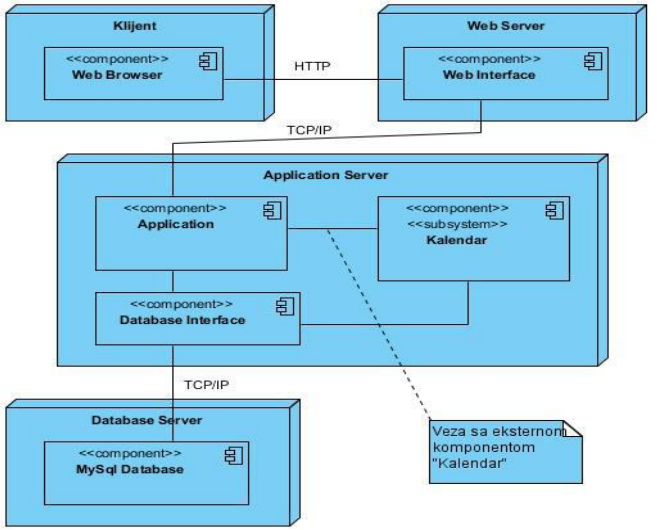
УМЛ – Дијаграм сложене структуре

- Представља интерну структуру класе, објекта, компоненте или случаја употребе.
- Садржи
  - сложене компоненте и њихове елементе
  - тачке интеракције са другим елементима система

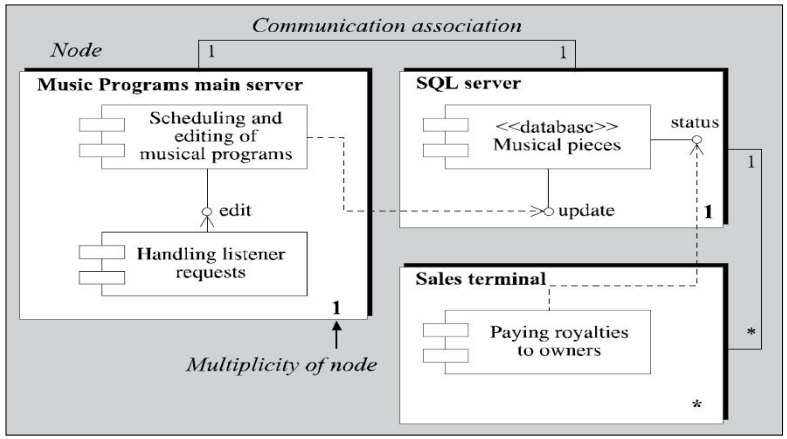
УМЛ – Дијаграм испоручивања

- Представља елементе физичке архитектуре система.
- Садржи
  - чворове (сервере)
  - софтверске или хардверске подсистеме
  - међусобне везе подсистема
  - може да илуструје и заступљеност компоненти у подсистемима

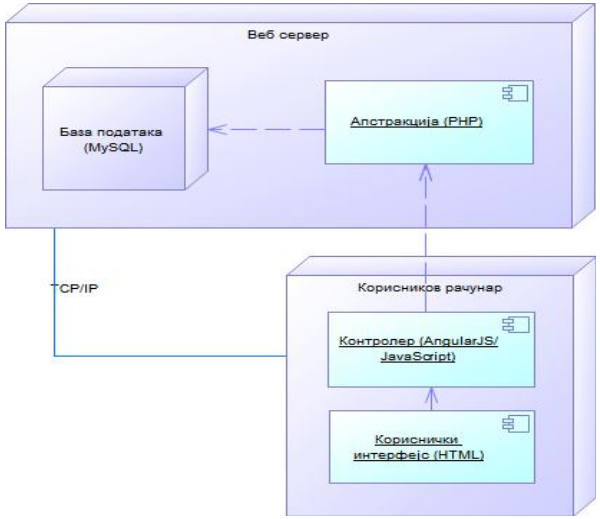
УМЛ – Дијаграм испоручивања – пример 1



УМЛ – Дијаграм испоручивања – пример 2



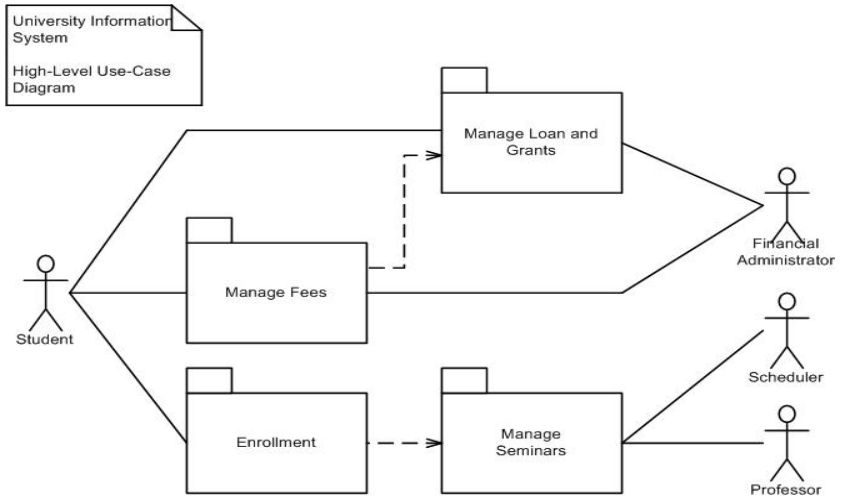
УМЛ – Дијаграм испоручивања – пример 3



УМЛ – Дијаграм пакета

- Илуструје како су елементи логичког модела организовани у пакете, као и међузависности пакета.
- Садржи
  - називе и границе пакета
  - класе у пакетима
  - међусобне односе класа
  - међусобне зависности пакета
  - може да се користи и у домену случајева употребе

УМЛ – Дијаграм пакета – пример 1



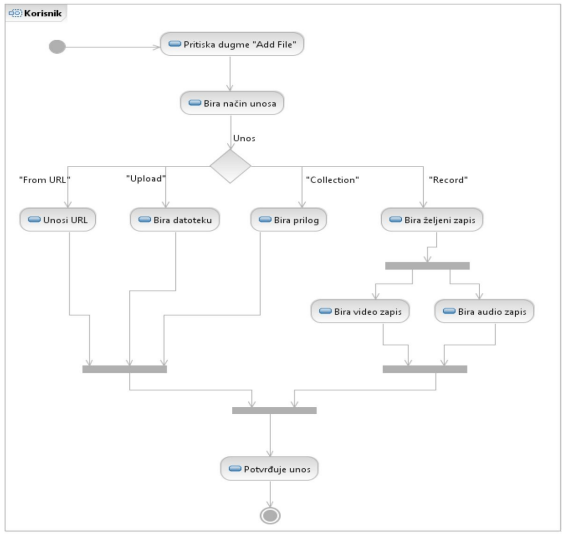
УМЛ – Дијаграми понашања

- Дијаграм активности (*activity diagram*)
- Дијаграм стања (*state machine diagram*)
- Дијаграм случајева употребе (*use case diagram*)
  - У ове дијаграме се могу убројати и дијаграми интеракције

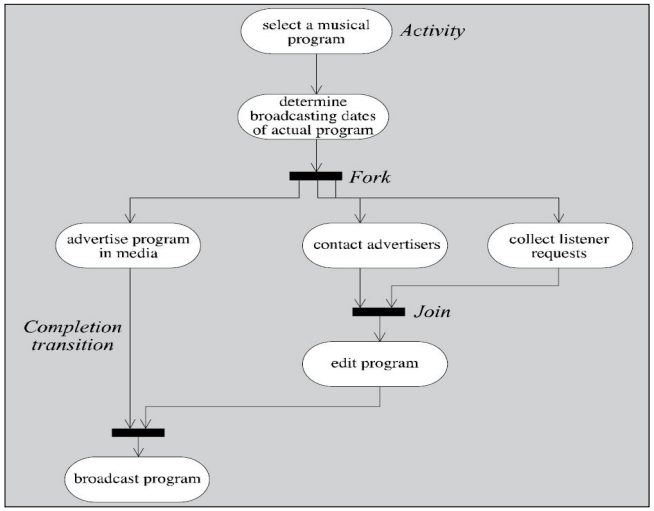
УМЛ – Дијаграм активности

- Представља пословне процесе вишег нивоа, токове података и евентуално сложене логичке елементе система.
- Садржи
  - процесе
  - токове података
  - чворишта и гранања
  - условне тачке
  - почетне и завршне тачке
  - може да садржи и “линије аутора”

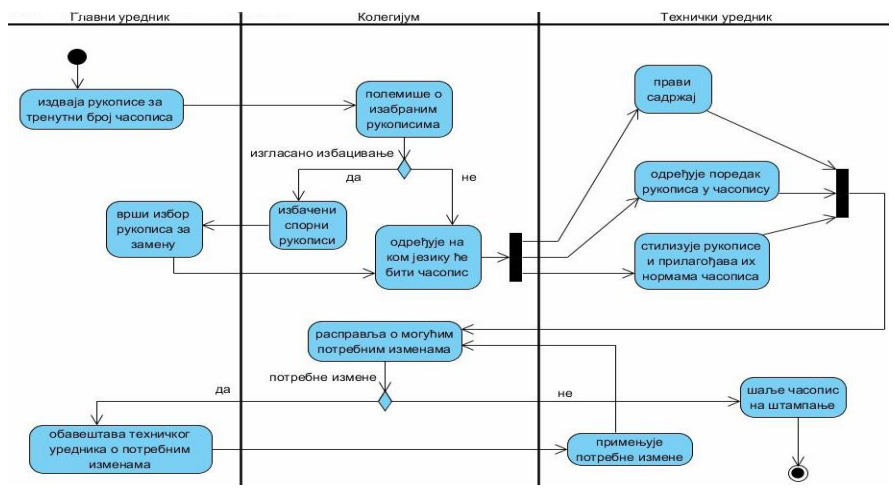
УМЛ – Дијаграм активности – пример 1



УМЛ – Дијаграм активности – пример 2



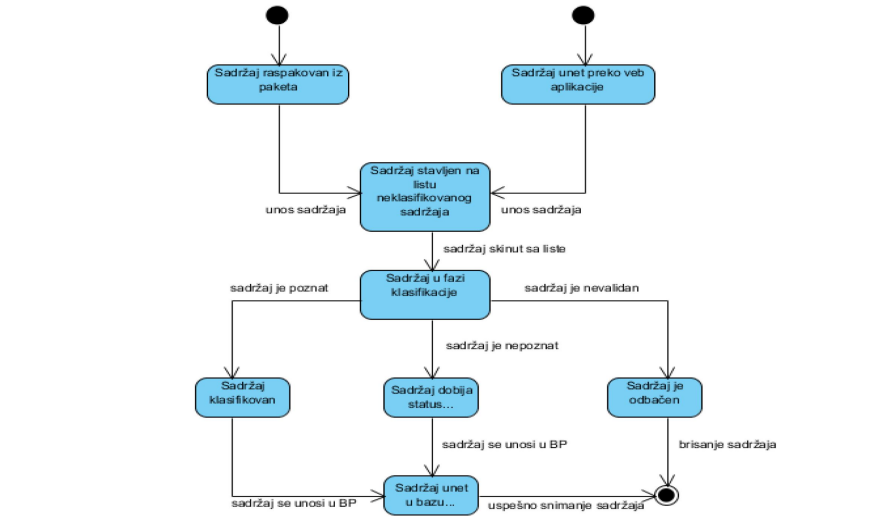
УМЛ – Дијаграм активности – пример 3



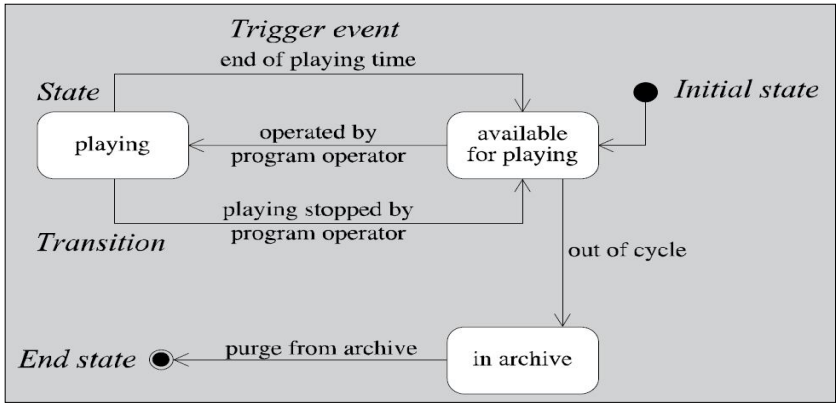
УМЛ – Дијаграм стања

- Описује како се стања објекта мењају у зависности од интеракција у које објекат улази.
- Садржи
  - сва могућа стања објекта
    - посебно означено почетно и завршно стање
  - преласке између стања
    - стрелица од претходног према наредном стању
    - називи догађаја који мењају стање објеката
    - одговарајућа објашњења

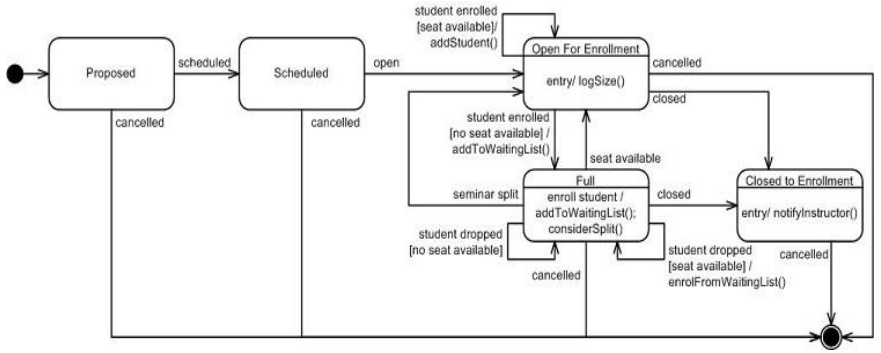
УМЛ – Дијаграм стања – пример 1



УМЛ – Дијаграм стања – пример 2



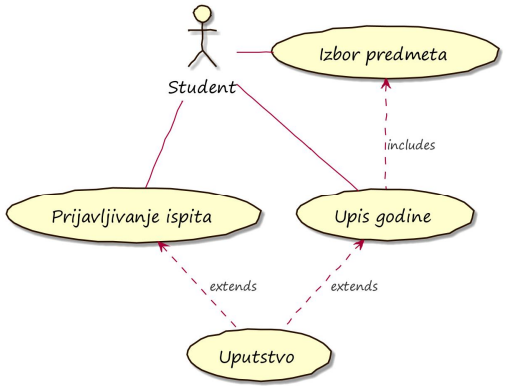
УМЛ – Дијаграм стања – пример 3



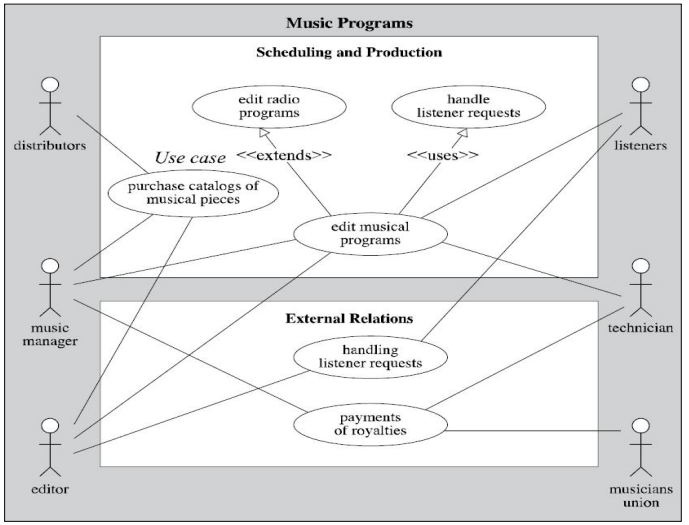
УМЛ – Дијаграм случајева употребе

- Представља случајеве употребе, актере и њихове међусобне односе.
- Садржи
  - случајеве употребе
  - актере
  - пакете
  - подсистеме
  - мељусобне односе

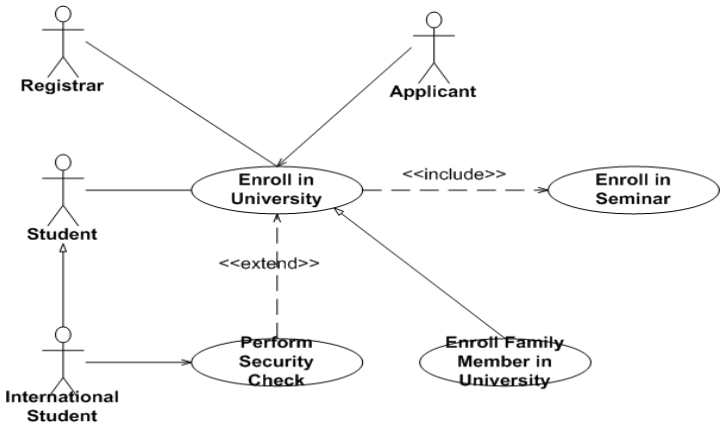
УМЛ – Дијаграм случајева употребе – пример 1



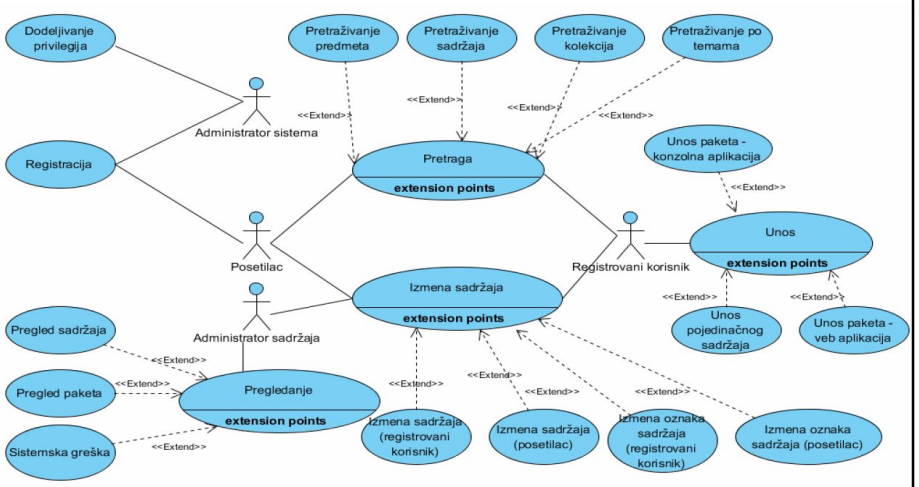
УМЛ – Дијаграм случајева употребе – пример 2



УМЛ – Дијаграм случајева употребе – пример 3



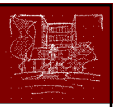
УМЛ – Дијаграм случајева употребе – пример 4



УМЛ – Дијаграми интеракције

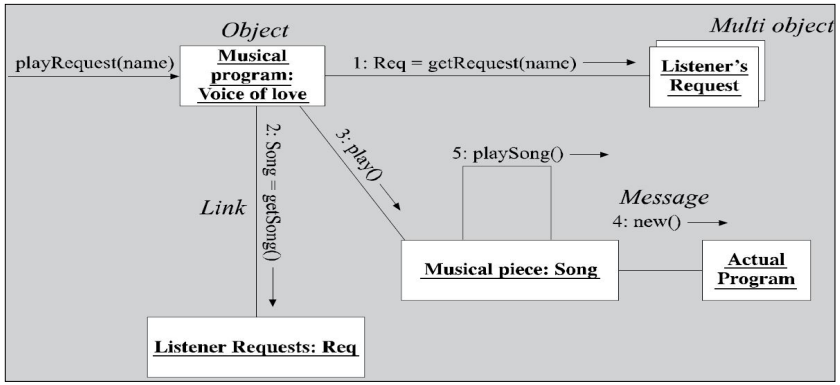
- Дијаграм комуникације (*communication diagram*)
  - ранији назив Дијаграм сарадње (*collaboration d.*)
- Дијаграм интеракција (*interaction diagram* или *interaction overview diagram*)
- Дијаграм секвенце (*sequence diagram*)
- Дијаграм времена (*timing diagram*)

УМЛ – Дијаграм комуникације

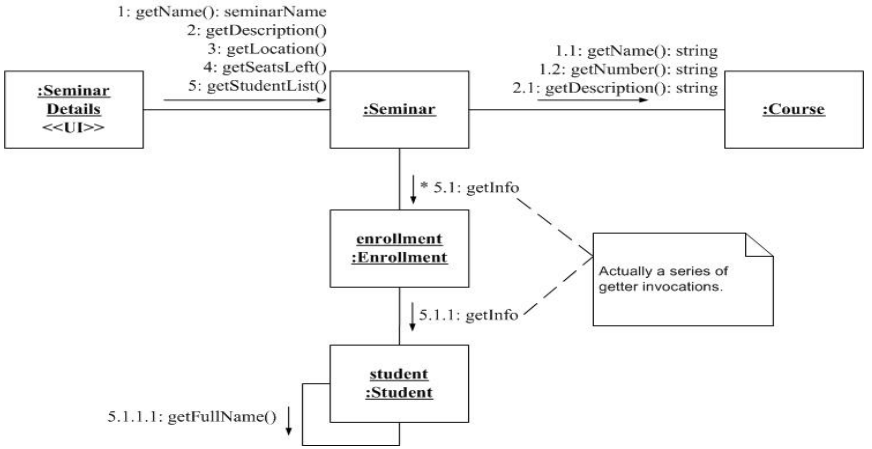


- Представља објекте, њихове међусобне односе и поруке које размењују. Пажња се по правилу посвећује структурној организацији објеката који учествују у размени порука.
- Садржи
  - објекте (може и актере)
  - поруке
  - коментаре и напомене

УМЛ – Дијаграм комуникације – пример 1



УМЛ – Дијаграм комуникације – пример 2



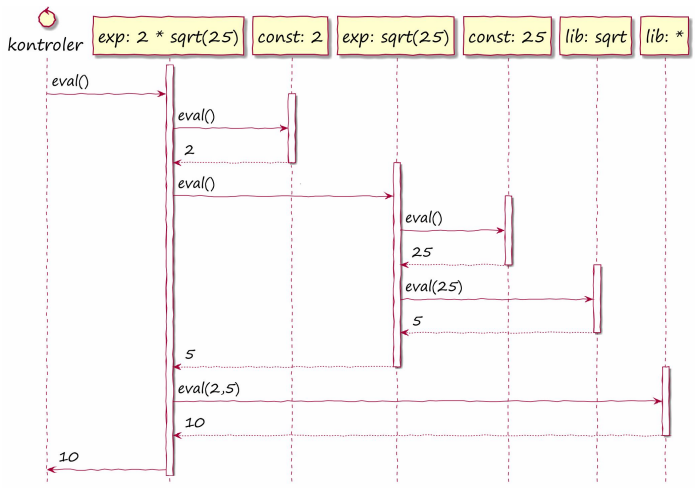
УМЛ – Дијаграм интеракција

- Варијанта дијаграма активности у којој је акценат на управљању процесима или системом. Сваки чвор/активност у дијаграму може да представља неки други дијаграм интеракција или активности
- Садржи
  - објекте
  - мање дијаграме активности или интеракција
  - случајеве употребе
  - ток одвијања процеса (протока података)
  - гранања и спајања
  - почетак и крај

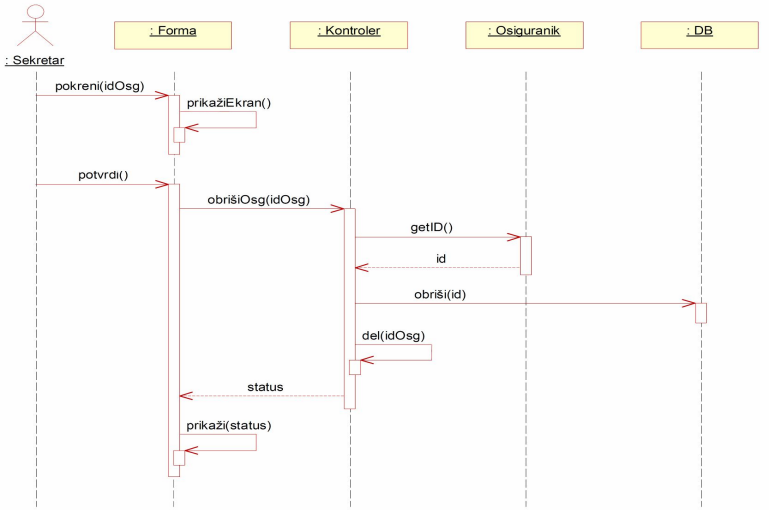
УМЛ – Дијаграм секвенце

- Представља случајеве употребе, актере и њихове међусобне односе
- Садржи
  - случајеве употребе
  - актере
  - пакете
  - подсистеме
  - међусобне односе

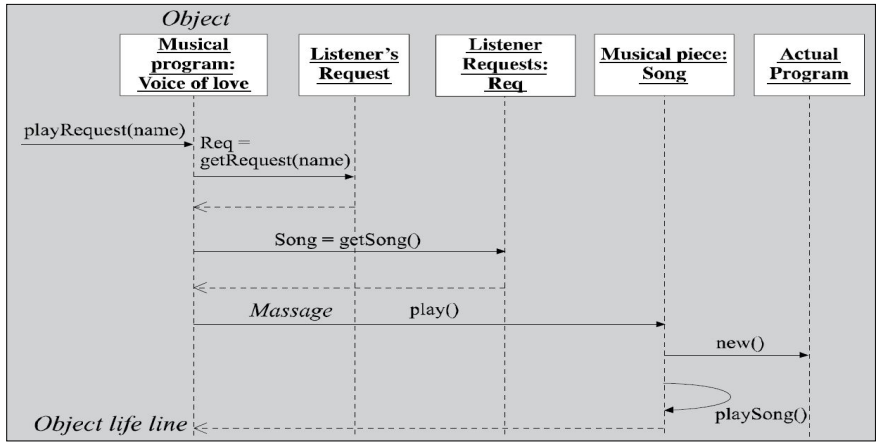
УМЛ – Дијаграм секвенце – пример 1



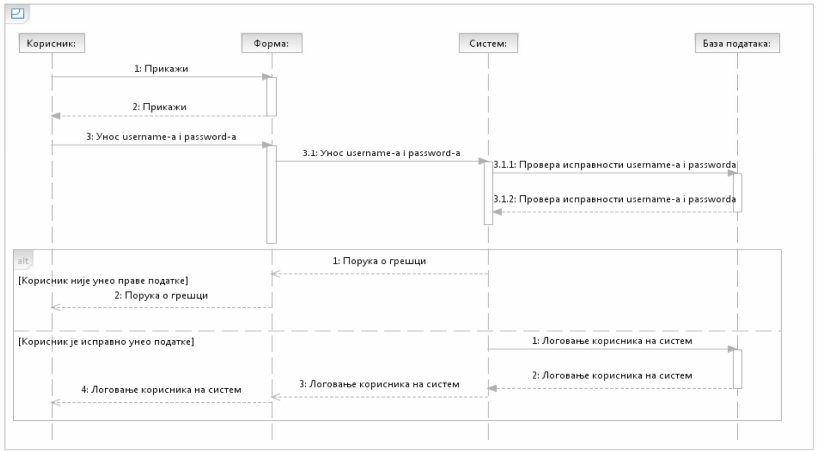
УМЛ – Дијаграм секвенце – пример 2



УМЛ – Дијаграм секвенце – пример 3



УМЛ – Дијаграм секвенце – пример 4



УМЛ – Дијаграм времена

- Представља промене стања или услова објеката током времена. Уобичајено се употребљава за представљање промена стања у зависности од спољахњих догађаја.
- Садржи
  - проток времена
  - спољашње догађаје
  - промен стања

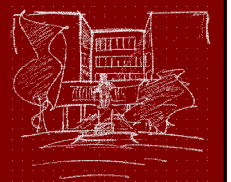
Литература за тему

- *Simon Bennett, Steve McRobb, Ray Farmer,* ***Object Oriented Systems Analysis and Design (Using UML)****, McGraw Hill, 2002*
- *Ambler*, http://www.agilemodeling.com/essays/umlDiagrams.htm
- *OMG UML*: http://www.uml.org/

Помињани извори

- *Avison, Fitzgerald,* ***Information Systems Development (3.ed)****, McGraw Hill, 2003*
- *Booch,* ***Object oriented design with applications (1.ed)****, Benjamin/Cummings, 1991.*
- *Coad, Yourdon,* ***Object Oriented Analysis (2.ed)****, Prentice Hall, 1991.*
- *Martin, Odell,* ***Object Oriented Information Engineering****, Prentice Hall, 1992.*
- *Yourdon, Argila,* ***Case Studies in Object-Oriented Analysis and Design****, Prentice Hall, 1996*

Хвала на пажњи!



МАТФ
Универзитет у Београду
Математички факултет