



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19970

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Ak. god. 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

INŽENJERSTVO ZAHTJEVA

„How To Engineer Software”

Why Should We Care?

- ▶ 18% of projects fail to deliver any usable software
- ▶ Of projects that do deliver, average
 - 42% late
 - 35% over budget
 - 25% under scope
 - Abundance of delivered defects
- ▶ 2019 US software budget ~\$340 billion
 - ~\$61 billion in cancellations
 - ~\$72 billion in cost overruns
 - ~\$41 billion in scope under-runs
 - → Funders expected to pay only ~\$166 billion for functionality actually delivered!

See also:
“Top Five Challenges
in Software Development
Lunch & Learn”



Reference: [Standish13]

[Izvor: Software Engineering 101 | Steve Tockey – YouTube]

„Surova stvarnost”

- *Standish skupina* →
- Izvješća *CHAOS ≡ Comprehensive Human Appraisal for Originating Software*
- <https://www.opendoorerp.com/the-standish-group-report-83-9-of-it-projects-partially-or-completely-fail/>

„Surova stvarnost”

- <https://qracorp.com/wp-content/uploads/2020/10/Leveraging-NLP-in-Requirements-Analysis.pdf> (James Martin)

„How To Engineer Software”

Top Five Root Causes of Poor Performance

- Data supports
 - (1) Vague, ambiguous, incomplete requirements
 - (2) Inadequate project management
- Professional experience suggests
 - (3) Uncontrolled design, code complexity
 - (4) Over-dependence on testing
 - (5) “Self-documenting code” is myth

See also:
“Top Five Challenges
in Software Development
Lunch & Learn”



Višedimenzijska klasifikacija zahtjeva

- Apstraktne izjave o usluzi koju bi programski sustav trebao ponuditi?
ili
- Detaljna formalna definicija funkcije programskog proizvoda?

Višedimenzijska klasifikacija zahtjeva

Primjer 4.1. Ako neka tvrtka želi ponuditi natječaj o izradi velikog, složenog programskog projekta na otvoreno tržište, tada ona nudi apstraktnu specifikaciju koja u najopćenitijim crtama opisuje funkciju konačnog programskog proizvoda. Tada se na natječaj javljaju ponuđači koji predlažu načine na koje će riješiti zadani problem. Nakon što pregleda prijedloge ponuđača, tvrtka odabire izvođača projekta. Zatim taj izvođač piše detaljniju specifikaciju sustava koju tvrtka vrednuje prije nego što se potpisuje ugovor i kreće u izradu projekta.

- → I jedno i drugo smatra se specifikacijom zahtjeva.

Podjela zahtjeva prema razini detalja

(od manje prema većoj razini detalja)

- **korisnički zahtjevi** (engl. *user requirements*)
- **zahtjevi sustava** (engl. *system requirements*)
- **specifikacija programske potpore** (engl. *software specification*)

Podjela zahtjeva prema razini detalja

Korisnički zahtjevi

1. Programski sustav za ministarstvo zdravstva generirat će mjesечna izvješća za upravu u kojima će predočiti cijenu lijekova koji se propisuju za svaku kliniku tijekom tog mjeseca.

Zahtjevi sustava

- 1.1 Na zadnji radni dan u mjesecu, generirat će se sažetak o propisanim lijekovima, njihovoj cijeni i klinikama koje su ih propisale.
- 1.2 Sustav će automatski generirati izvješće spremno za ispis nakon 17:30 na zadnji radni dan u mjesecu.
- 1.3 Izvješće će biti napravljeno i za svaku kliniku posebno i popisat će pojedinačne nazine lijekova, ukupan broj recepata, broj propisanih doza i ukupnu cijenu propisanih lijekova.
- 1.4 Ako su lijekovi dostupni u različitim dozama (npr. 10 mg, 20 mg), zasebna izvješća će se napraviti za svaku postojeću dozu.
- 1.5 Pristup svim izvješćima o cijenama bit će ograničen na sve ovjerene korisnike koji se nalaze na kontrolnoj listi s razinom pristupa: "uprava".

Podjela zahtjeva prema razini detalja

- Dionici koji čitaju ili dorađuju zahtjeve:
 - Klijentski inženjeri
 - Klijentski rukovoditelji i menadžeri
 - Krajnji korisnici sustava
 - Rukovoditelji za pisanje ugovora
 - Specijalisti za oblikovanje sustava – arhitekti
 - Specijalisti za razvoj programske potpore
 - Stručnjaci iz domene primjene sustava

Zahtjevi prema razini detalja

Dionici / Vrsta zahtjeva	Korisnički zahtjevi	Zahtjevi sustava	Specifikacija programske potpore
Klijentski inženjeri			
Klijentski rukovoditelji i menadžeri			
Krajnji korisnici sustava			
Rukovoditelji za pisanje ugovora			
Specijalisti za oblikovanje sustava – arhitekti			
Specijalisti za razvoj programske potpore			
Stručnjaci iz domene primjene sustava			

Zahtjevi prema razini detalja

Dionici / Vrsta zahtjeva	Korisnički zahtjevi	Zahtjevi sustava	Specifikacija programske potpore
Klijentski inženjeri	✗	✗	✗
Klijentski rukovoditelji i menadžeri	✗		
Krajnji korisnici sustava	✗	✗	
Rukovoditelji za pisanje ugovora	✗		
Specijalisti za oblikovanje sustava – arhitekti	✗	✗	✗
Specijalisti za razvoj programske potpore		✗	✗
Stručnjaci iz domene primjene sustava	✗		

Podjela zahtjeva prema sadržaju

- **funkcionalni zahtjevi** (engl. *functional requirements*)
- **nefunkcionalni ili ostali zahtjevi** (engl. *non-functional, other requirements*)
- **zahtjevi domene primjene** (engl. *domain requirements*)

Funkcionalni zahtjevi

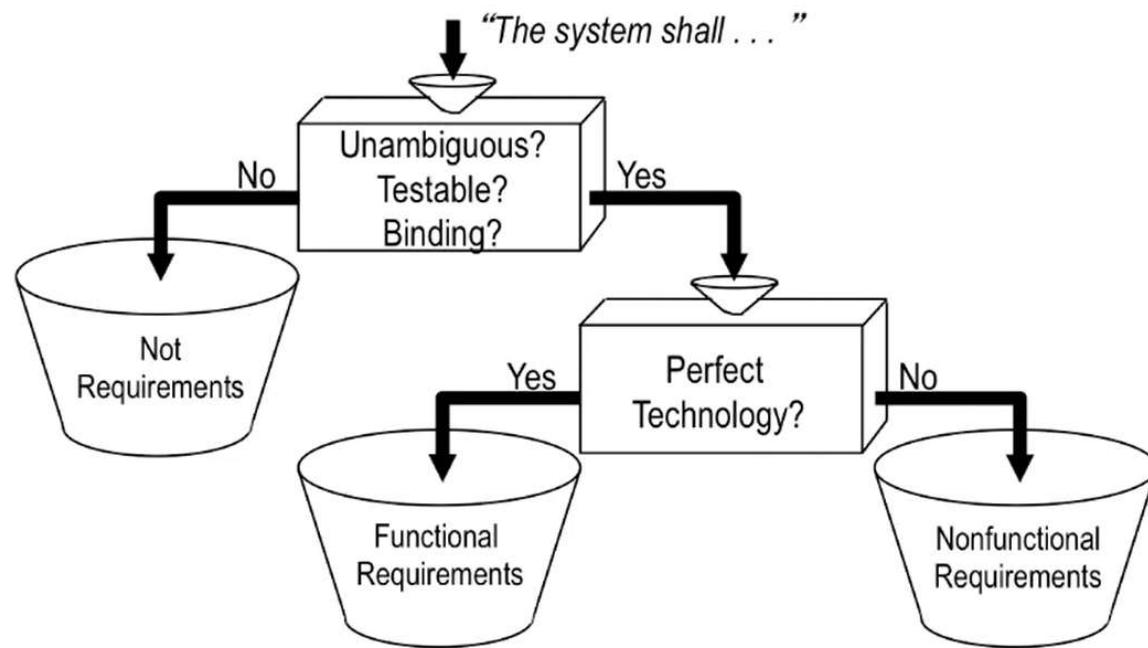
- izjave o:
 - uslugama koje programski proizvod mora pružati,
 - kako će sustav reagirati na određeni ulazni poticaj,
 - kako bi se trebao ponašati u određenim situacijama.
- → potpuni i dosljedni
("kompletni" i "konzistentni")

Nefunkcionalni ili ostali zahtjevi

- ograničenja u uslugama i funkcijama programske potpore (kvantitativno, ako može):
 - **zahtjevi programske potpore** – npr. vrijeme odziva, podržani sustav, komunikacijski protokol i sl.
 - **organizacijski zahtjevi** – npr. uporaba propisanog normiranog procesa razvoja, korištenje uvijek točno određenog programskog jezika pri razvoju.
 - **vanjski zahtjevi** – npr. postizanje međusobne operabilnosti s drugim sustavima, zakonski zahtjevi i drugo.

(Ne)funkcionalni zahtjevi

Semantic Model is Technology-Free



Izvor: How to Engineer Software, Part 2: A Deeper Dive into Semantic Models | Steve Tockey – YouTube

Zahtjevi domene primjene

- pojavljuju se kasnije u procesu otkrivanja zahtjeva:
 - razumljivost vs. implicitnost
- mogu biti novi funkcionalni zahtjevi ili ograničenja na postojeće zahtjeve

Razlikovanje vrste zahtjeva

Primjer 4.3. Hipotetski sustav LIBSYS

Opis: Knjižničarski sustav koji pruža jedinstveno sučelje prema bazama članaka u različitim knjižnicama. Korisnik može pretraživati, spremati i ispisivati članke za osobne potrebe.

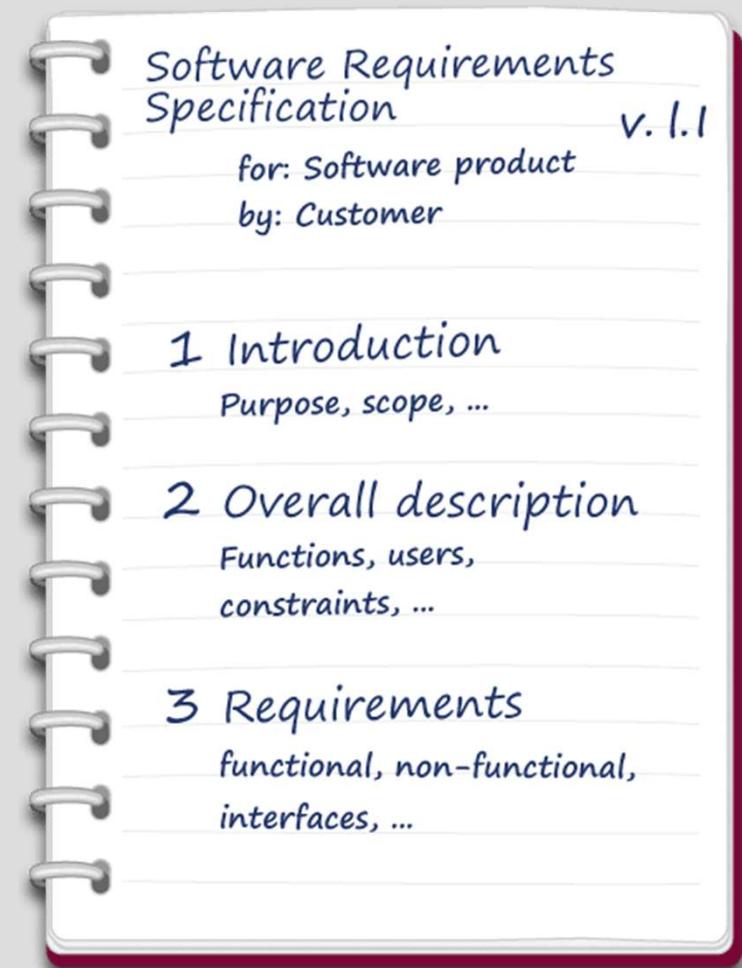
Zahtjevi: Korisnik mora moći pretraživati početni skup baza članaka ili njihov podskup. Sustav mora sadržavati odgovarajuće preglednike koji omogućuju čitanje članaka u knjižnici. Korisničko sučelje LIBSYS sustava treba biti implementirano kao jednostavni HTML bez uporabe okvira ili Java "appleta". Svakoj narudžbi mora se alocirati jedinstveni identifikator (ORDER_ID) koji korisnik mora moći kopirati u svoj korisnički prostor.

Načini specifikacije zahtjeva

- Rečenice prirodnog jezika
- Strukturirani prirodni jezik
- Specifični jezik za opis oblikovanja
- Grafička notacija
- Matematička specifikacija

Dokument zahtjeva

- Kakav treba biti?
- Što treba uključivati?



[Izvor: <https://www.microtool.de/en/knowledge-base/what-is-a-software-requirements-specification>]

Dokument zahtjeva

- Prema normi instituta IEEE (1998.):

Table of Contents

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, acronyms, and abbreviations

1.4 References

1.5 Overview

2. Overall description

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 Constraints

2.5 Assumptions and dependencies

3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendices

Index

[<http://www.cse.msu.edu/~cse870/IEEEExplore-SRS-template.pdf>]

Dokument zahtjeva

- I. Sommerville, Software Engineering, 9th ed., Addison-Wesley, Harlow, England, 2011.:
 1. Predgovor
 2. Uvod
 3. Rječnik pojmove
 4. Definicija korisničkih zahtjeva
 5. Specifikacija zahtjeva sustava
 6. Arhitektura sustava
 7. Modeli sustava
 8. Evolucija sustava
 9. Dodaci
 10. Indeks (pojmove, dijagrama, funkcija).

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

LITERATURA

REFERENCE I LITERATURA

- A. Jović, N. Frid, D. Ivošević: Procesi programskog inženjerstva, 3. izdanje, Sveučilište u Zagrebu, FER ZEMRIS, 2019.
- I. Sommerville, Software engineering, 8th ed., Addison Wesley, 2007.
- G. Overgaard, B. Selic, C. Bock, OMG, 2000.
- S. Tockey: How to Engineer Software, Wiley-IEEE Computer Society Press, 2019.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19970

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Ak. god. 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

INŽENJERSTVO ZAHTJEVA

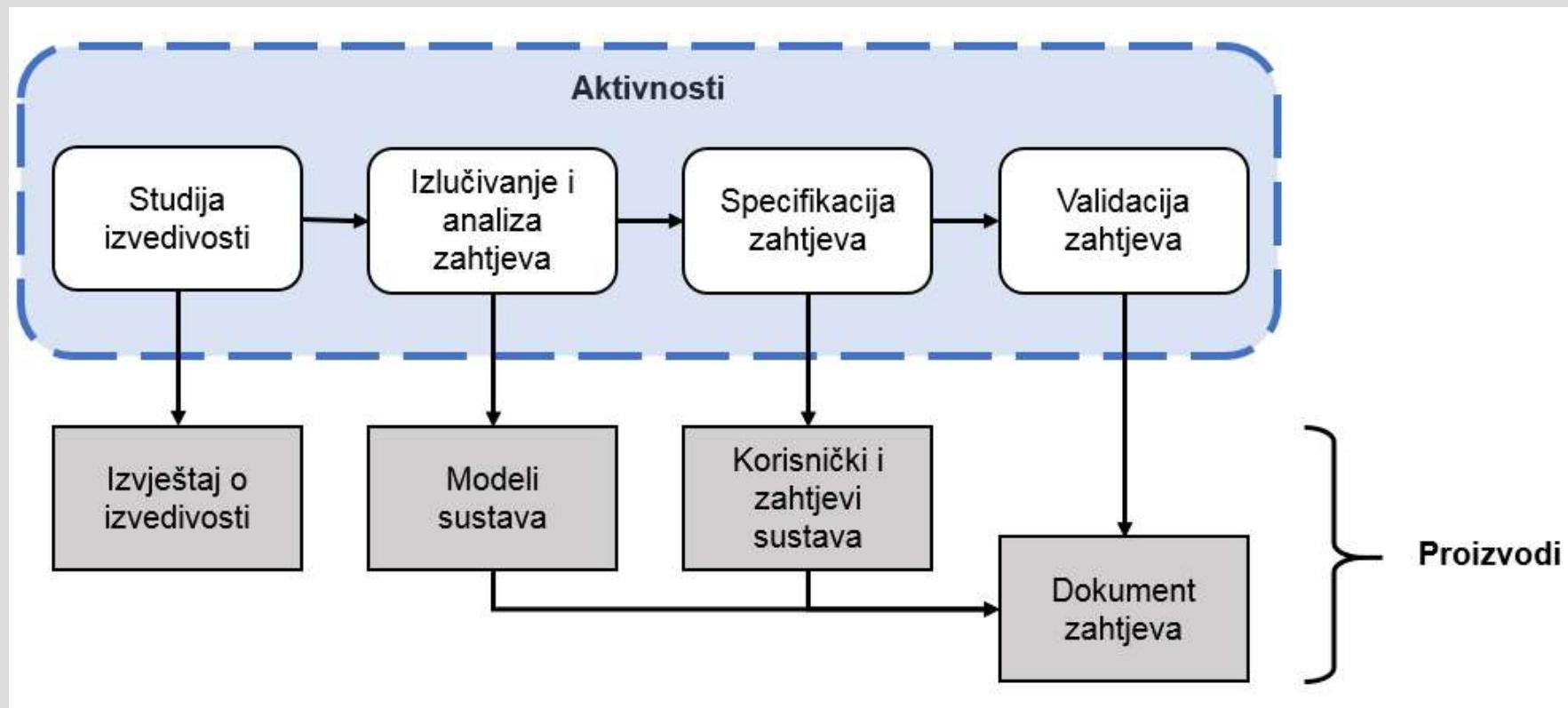
INŽENJERSTVO ZAHTJEVA

PROCESI INŽENJERSTVA ZAHTJEVA

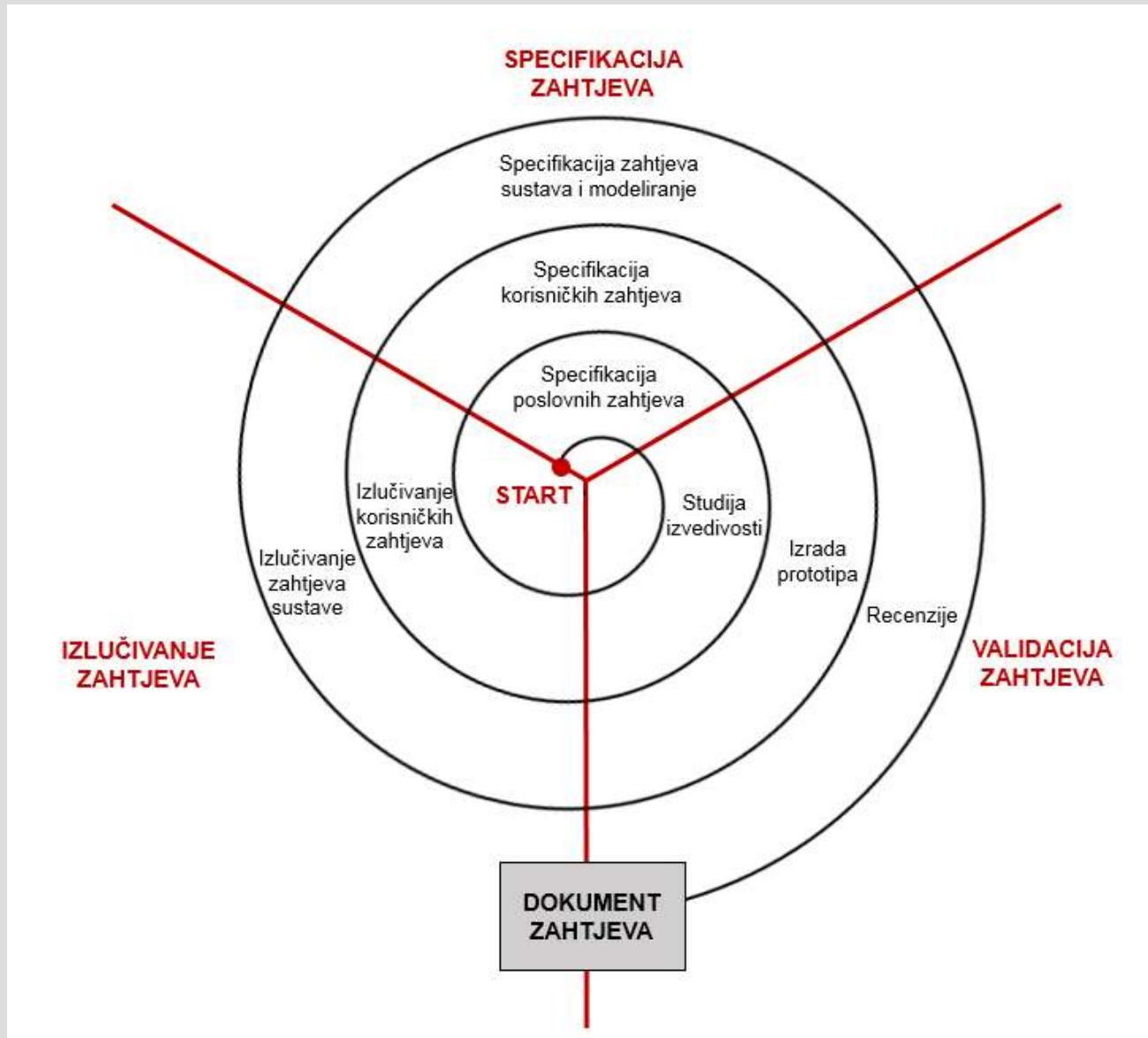
Procesi inženjerstva zahtjeva

- nema jedinstvenog procesa inženjerstva zahtjeva, ali ...
- ... postoje neke **generičke aktivnosti**:
 - **studija izvedivosti** (engl. *feasibility study*)
 - **izlučivanje (otkrivanje) zahtjeva i analiza zahtjeva** (engl. *requirements elicitation and analysis*)
 - **specifikacija (zapisivanje) zahtjeva** (engl. *requirements specification*)
 - **validacija zahtjeva** (engl. *requirements validation*)
 - **upravljanje promjenama u zahtjevima** (engl. *requirements management*)
- dva modela: klasični i spiralni

Klasični model procesa inženjerstva zahtjeva



Spiralni model procesa inženjerstva zahtjeva



Studija izvedivosti

- Isplati li se predloženi sustav finansijski (tj. je li vrijedan uloženih sredstava)?
- Temeljna pitanja:
 - Doprinosi li programski sustav ciljevima organizacije u koju se uvodi?
 - Može li se programski sustav izvesti postojećom tehnologijom i predviđenim sredstvima?
 - Može li se predloženi sustav integrirati s postojećim sustavima organizacije u koju se uvodi?

Izlučivanje i analiza zahtjeva

- uključiti sve ili što više dionika sustava – osoba koje će razvijeni sustav izravno ili neizravno pogoditi

Izlučivanje i analiza zahtjeva

- u ciklusima:



Izlučivanje i analiza zahtjeva

- Koraci u iteracijama su:
 - **izlučivanje (otkrivanje) zahtjeva**
 - sa svim dionicima
 - intervjuiranje, izrada scenarija i etnografija
 - **klasifikacija i organizacija zahtjeva**
 - grupiranje srodnih zahtjeva
 - prema podsustavima → model sustava → razvoj arhitekture
 - **postavljanje prioriteta i pregovaranje**
 - razvrstavanje po prioritetima
 - razrješavanje sporova zbog različitih pogleda
 - **specifikacija zahtjeva**
 - dokumentiranje poznatim tehnikama

Izlučivanje i analiza zahtjeva

- **Pogledi** (engl. *viewpoint*) su način strukturiranja zahtjeva tako da oslikavaju perspektivu i fokus različitih grupa dionika. Svaki pogled uključuje skup zahtjeva sustava.

Izlučivanje i analiza zahtjeva

- Različiti pogledi:
 - ljudi i drugi dionici koji izravno komuniciraju sa sustavom → **pogledi interakcije**
 - dionici koji utječu na zahtjeve, ali ne koriste sustav izravno → **neizravni pogledi**
 - karakteristike domene i ograničenja na sustav → **pogledi domene primjene**
- → strukturiranje aktivnosti izlučivanja i specifikacije zahtjeva

Izlučivanje i analiza zahtjeva

- Metode izlučivanja zahtjeva:
 - intervjuiranje
 - izrada scenarija
 - etnografija

Izlučivanje i analiza zahtjeva

- Intervjuiranje:
 - formalno i neformalno
 - zatvoreno i otvoreno
- nije toliko korisno za razumijevanje zahtjeva u domeni primjene
- nije pogodno za dobivanje uvida u organizacijske zahtjeve i ograničenja

Izlučivanje i analiza zahtjeva

- Scenariji su pažljivo osmišljeni primjeri o stvarnom načinu korištenja sustava.
- → stvarni primjeri su bolji nego apstraktni opisi
- „pričanje priče” ?

Izlučivanje i analiza zahtjeva

- Izrada scenarija:
 - opis početne situacije
 - opis normalnog (standardnog) tijeka događaja
 - opis mogućih odstupanja od normalnog tijeka događaja
 - informaciju o paralelnim aktivnostima
 - opis stanja gdje scenarij završava.

Izlučivanje i analiza zahtjeva

- Primjer scenarija za prikupljanje povijesti bolesti za sustav MHC-PMS.

Početna pretpostavka:

Pacijent se našao s medicinskom sestrom na recepciji koja mu je otvorila zapis u sustavu i prikupila osobne informacije.

Druga medicinska sestra se prijavila u sustav i prikuplja povijest bolesti.

Izlučivanje i analiza zahtjeva

Normalni tijek događaja:

Sestra pretražuje pacijenta po prezimenu. Ako ima više pacijenata s istim prezimenom, pacijenta se dodatno identificira s imenom i datumom rođenja.

Sestra odabire opciju u izborniku za dodavanje povijesti bolesti.

Sestra slijedi niz upita kako bi unijela informacije o: mentalnom zdravlju (tekst), postojećim fizičkim bolestima (odabir iz izbornika), lijekovima (odabir iz izbornika) i alergijama (tekst).

Izlučivanje i analiza zahtjeva

Odstupanja od normalnog tijeka događanja:

Pacijentov zapis ne postoji ili ga se ne može pronaći. Sestra bi trebala napraviti novi zapis.

Pacijentove bolesti ili lijekovi nisu odabrani iz izbornika. Sestra bi trebala izabratи opciju "ostalo" i unijeti tekstualni opis bolesti/lijeka.

Pacijent ne može ili ne želi dati povijest bolesti. Sestra bi trebala unijeti tekstualni opis o tome da pacijent ne može ili ne želi dati informaciju. Sustav treba ispisati uobičajeni, potpisani obrazac na kojem piše da nedostatak informacija znači da će liječenje biti ograničeno ili odgođeno. Sestra treba obrazac predati pacijentu.

Izlučivanje i analiza zahtjeva

Ostale aktivnosti:

Pacijentov zapis se može pregledavati ali ne i mijenjati od strane ostalih zaposlenika dok ga sestra mijenja.

Izlučivanje i analiza zahtjeva

Završno stanje:

Sestra je prijavljena u sustav. Pacijentov zapis koji sadrži povijest bolesti je unesen u sustav. Dnevnik sustava pokazuje početak i kraj sjednice i ime i prezime sestre koja je provela upis podataka.

Izlučivanje i analiza zahtjeva

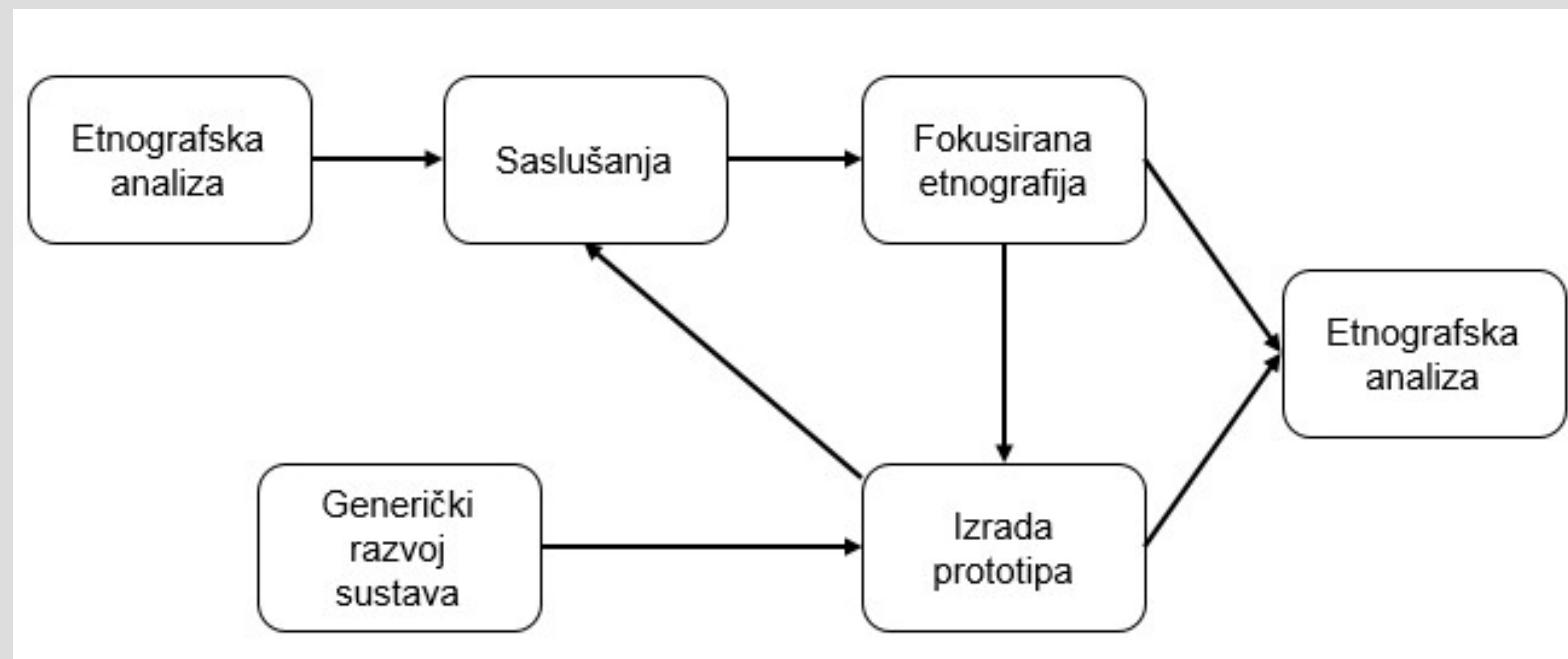
- Etnografija ?

Izlučivanje i analiza zahtjeva

- Etnografija je tehnika opažanja koja se koristi kako bi se bolje razumjeli procesi kod klijenta i kako bi se pomoglo otkriti što je moguće više ispravnih i korisnih zahtjeva. Etnografija podrazumijeva dolazak jednog ili više ljudi iz razvojnog tima u tvrtku gdje će se sustav primjenjivati i uključivanje tih inženjera (tzv. etnografa) u svakodnevne aktivnosti u tom okruženju.
- → Kako društveni i organizacijski kontekst utječe na pojedinu operaciju u sustavu?

Izlučivanje i analiza zahtjeva

- Fokusirana etnografija:



- → fokus na krajnjeg korisnika

Validacija zahtjeva

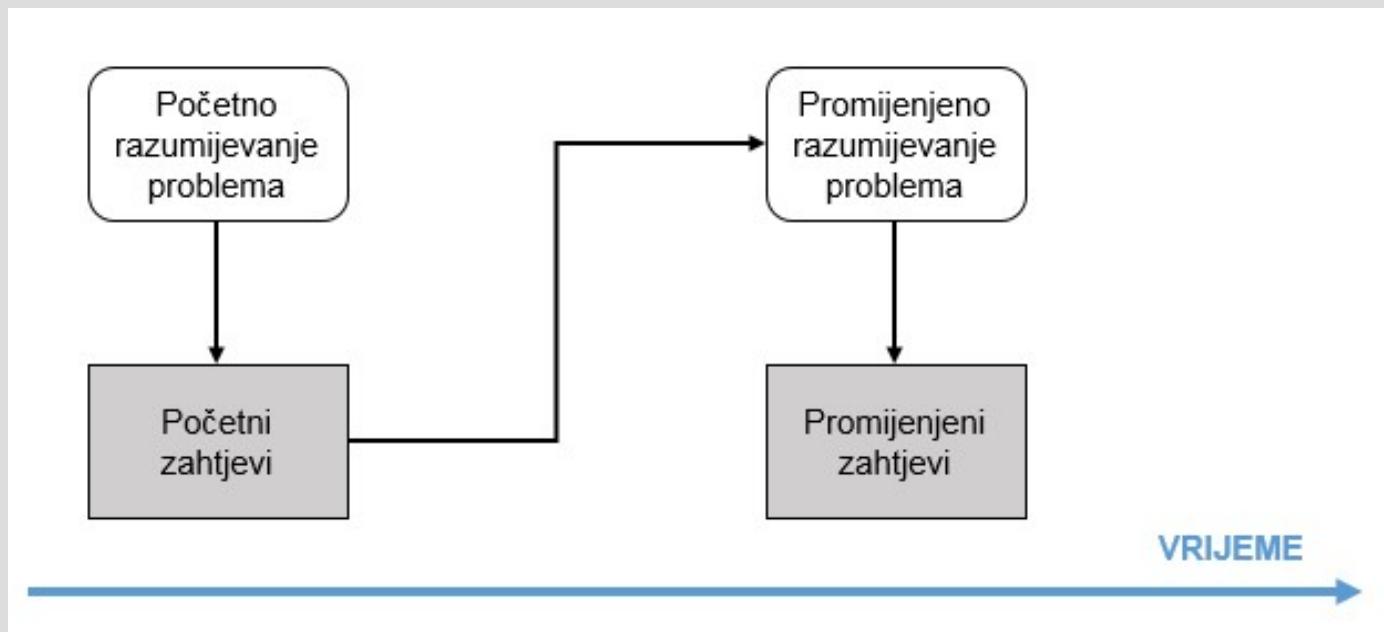
- Tehnike:
 - **recenzija zahtjeva** – detaljna, ručna analiza zahtjeva od strane zajedničkog tima
 - **izrada prototipa** – provjera zahtjeva na izvedenom sustavu
 - **generiranje ispitnih slučajeva** – razvoj ispitnih sekvenci za provjeru zahtjeva.

Validacija zahtjeva

- Provjera više **svojstava**:
 - **valjanost** – sustav ostvaruje funkcionalnost koja podupire potrebe većine dionika
 - **dosljednost/konzistentnost** – ne postoji konflikt u zahtjevima
 - **potpunost/kompletност** – sustav uključuje sve funkcije koje je korisnik tražio
 - **realnost** – sve funkcije se mogu implementirati uz danu tehnologiju i proračun
 - **provjerljivost** – svi zahtjevi se mogu provjeriti
 - **razumljivost** – dokument zahtjeva je jasno napisan
 - **slijedivost** – naveden je izvor dokumenta u slučaju više povezanih dokumenata
 - **prilagodljivost** – zahtjevi se mogu mijenjati bez utjecaja na druge zahtjeve.

Rukovanje promjenama u zahtjevima

- Promjene zbog:
 - promijenjenog modela poslovanja,
 - boljeg razumijevanja procesa tijekom razvoja
 - konfliktnih zahtjeva u različitim pogledima



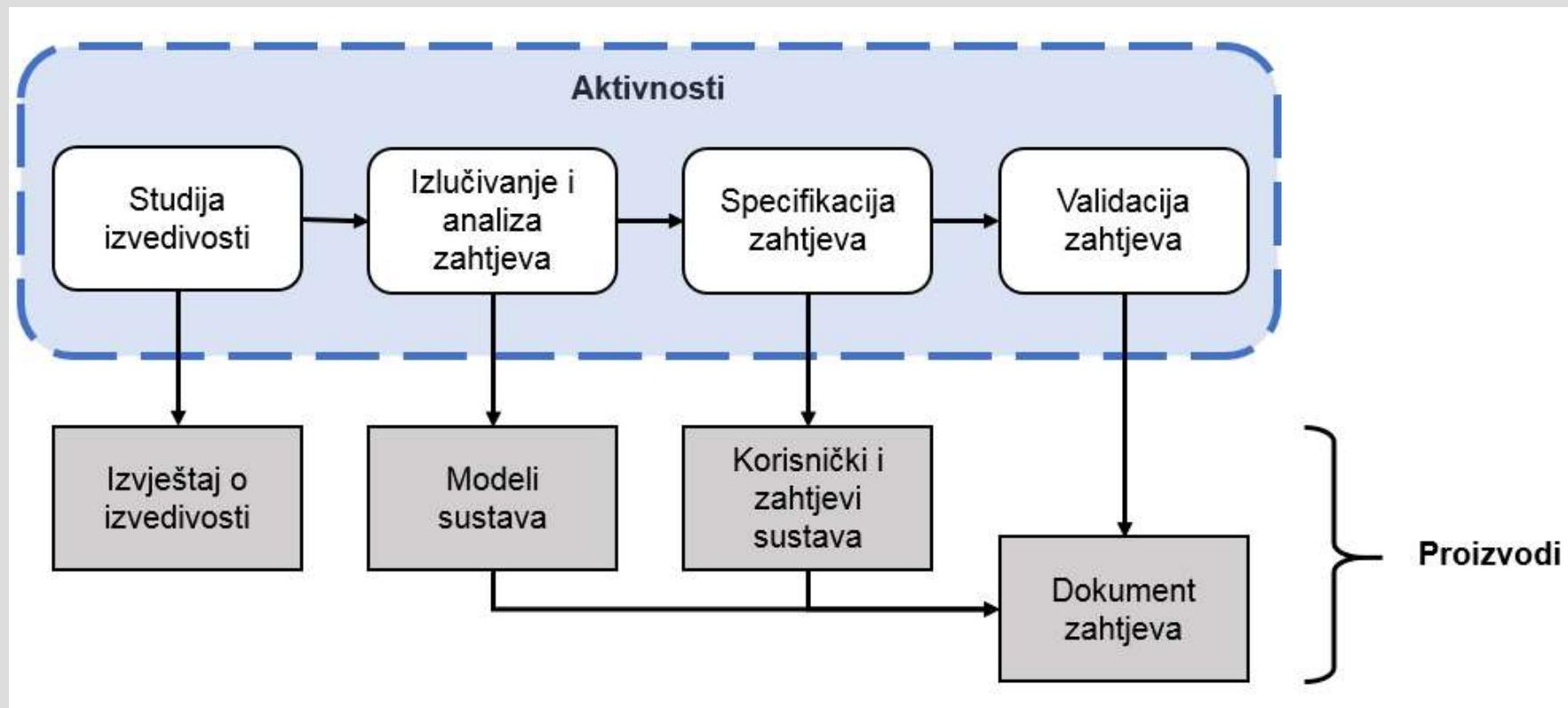
Rukovanje promjenama u zahtjevima

- Zahtjevi nakon uvođenja novog sustava:
 - **okolinom promijenjeni zahtjevi**
 - npr. bolnica mijenja financijski model pokrivanja usluga
 - **novonastali zahtjevi**
 - kupac sve bolje razumije sustav koji se oblikuje
 - **posljedični zahtjevi**
 - promjena procesa rada u organizaciji nastalih upravo uvođenjem novoga sustava
 - **zahtjevi kompatibilnosti**
 - ovisnost o procesima drugih sustava u organizaciji

Rukovanje promjenama u zahtjevima

- Formalni proces uvođenja promjena?
- Isplativost uvođenja promjena?
- Automatizirani proces uvođenja promjena?

Klasični model procesa inženjerstva zahtjeva



OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

LITERATURA

REFERENCE I LITERATURA

- A. Jović, N. Frid, D. Ivošević: Procesi programskog inženjerstva, 3. izdanje, Sveučilište u Zagrebu, FER ZEMRIS, 2019.
- I. Sommerville, Software engineering, 8th ed., Addison Wesley, 2007.
- G. Overgaard, B. Selic, C. Bock, OMG, 2000.
- S. Tockey: How to Engineer Software, Wiley-IEEE Computer Society Press, 2019.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19970

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

Objektno orijentirani razvoj programa

MODELIRANJE SUSTAVA

Referenca

- I. Sommerville, Software engineering, 10th ed., Pearson, 2016.,
<https://iansommerville.com/software-engineering-book/>

Modeliranje sustava

- proces razvoja apstraktnih modela sustava
- svaki model predstavlja različiti pogled ili perspektivu sustava.
- predstavljanje sustava uporabom grafičke notacije - skoro uvijek se temelji na UML-u
- pomoć za razumijevanje sustava analitičaru i za komunikaciju s korisnicima

Postojeći i/ili planirani sustavi

- Za postojeće sustave:
 - Pomažu u pojašnjavanju što sustav radi
 - Temelj rasprave o jakim i slabim stranama sustava što može dovesti do novih zahtjeva
- Za nove sustave:
 - Tijekom inženjerstva zahtjeva
 - Dionici - za razjašnjavanje zahtjeva dionicima
 - Inženjeri - za predstavljanje prijedloga oblikovanja sustava i dokumentiranje ostvarenja
- U procesu *modelno usmjerenog inženjerstva*:
 - Generiranje ostvarenja sustava na temelju modela (djelomično ili u potpunosti)

Gledište na sustav (perspektiva)

- Vanjsko
 - modeliranje konteksta ili okoline sustava
- Interakcijsko
 - modeliranje međudjelovanja između sustava i okoline ili između dijelova sustava
- Strukturno
 - modeliranje organizacije sustava ili strukture podataka koji se obrađuju u sustavu
- Ponašajno
 - modeliranje dinamike ponašanja sustava i načina odgovora na događaje (poticaje)

Vrste UML dijagrama

- Dijagrami aktivnosti
- Dijagrami obrazaca uporabe
- Dijagrami slijeda
- Dijagrami razreda
- Dijagrami stanja

Uporaba grafičkih modela

- Kao potpore raspravi o sustavu
 - Mogu biti nepotpuni i nedovoljno točni
- Kao način dokumentacije sustava
 - Trebaju biti precizni, ali ne moraju biti potpuni
- Kao detaljan opis sustava na temelju kojega se može generirati ostvarenje sustava
 - Moraju biti i točni i potpuni

MODELI KONTEKSTA (ILI KONTEKSTUALNI MODELI)

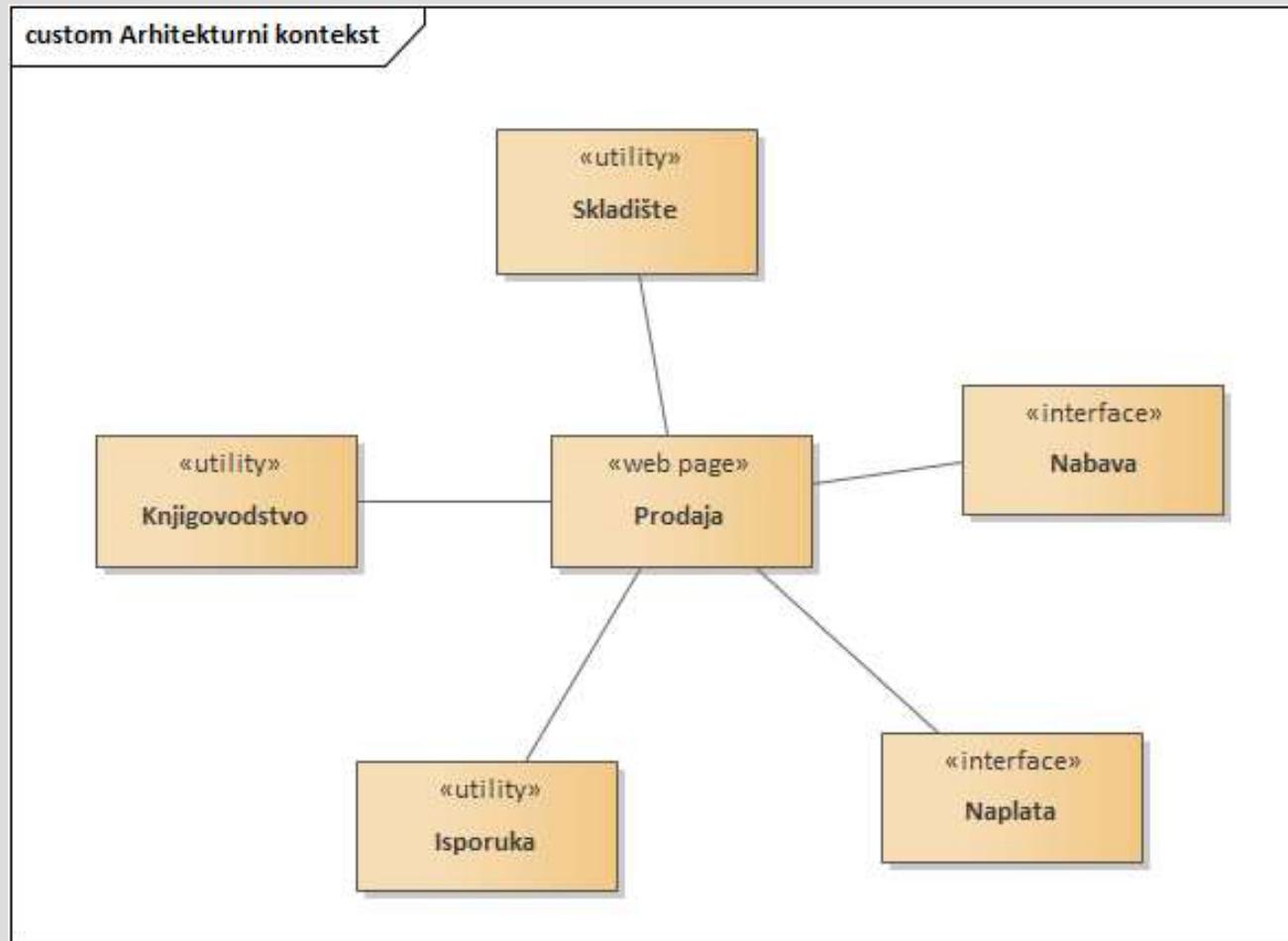
Kontekstualni modeli

- prikaz operativnog konteksta sustava
 - Što se nalazi unutar a što izvan granica sustava?
 - Koji su drugi sustavi u okruženju?
- gledište arhitekture sustava → arhitektturni model

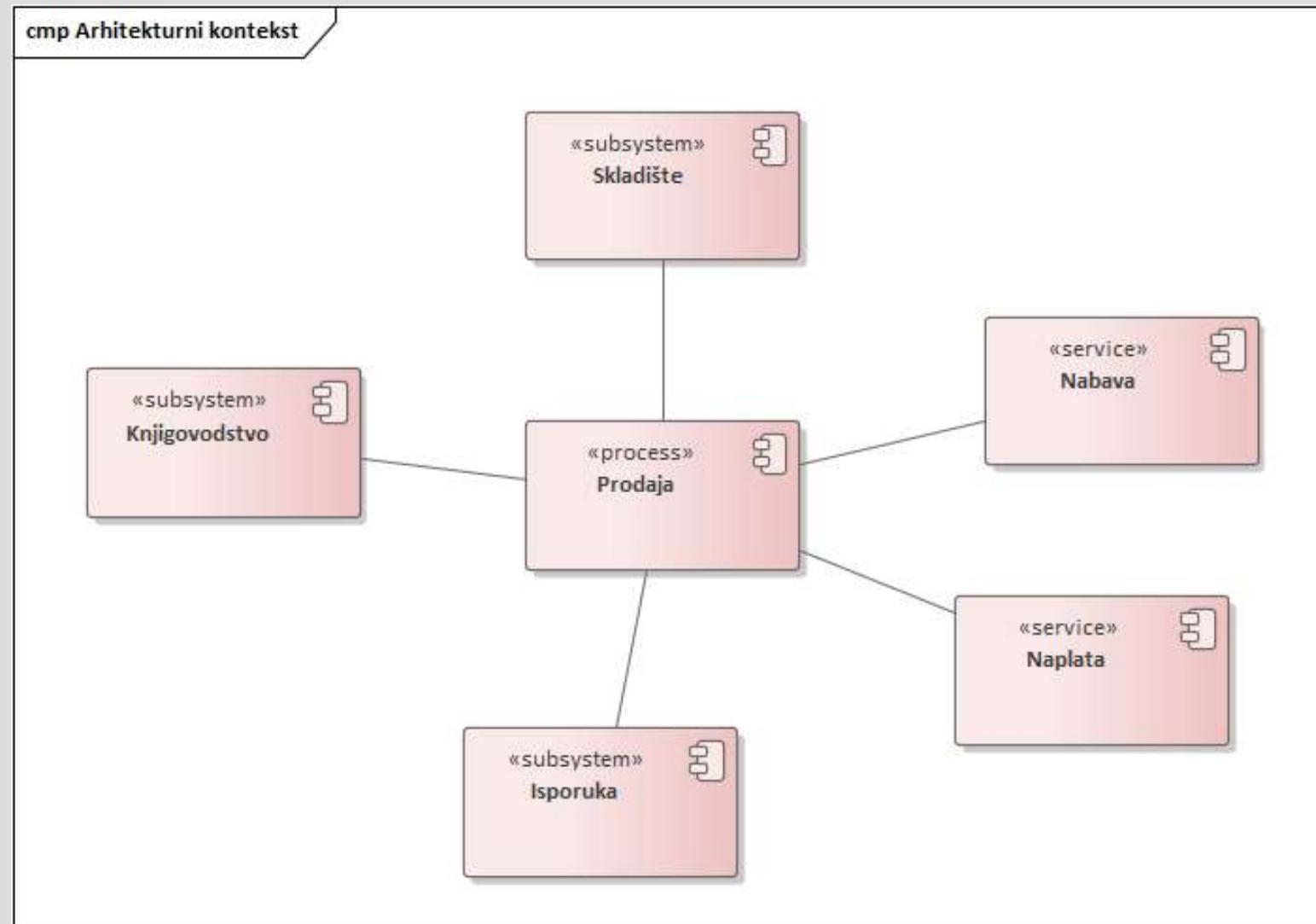
Granice sustava

- prikazuje i druge sustave koji se koriste ili ovise o našem sustavu
- pozicija granice
 - društveno, organizacijsko ili političko pitanje
 - određuje zahtjeve sustava
 - utječe na raspodjelu posla između različitih dijelova organizacije

Arhitekturni kontekst sustava prodaje auto-dijelova



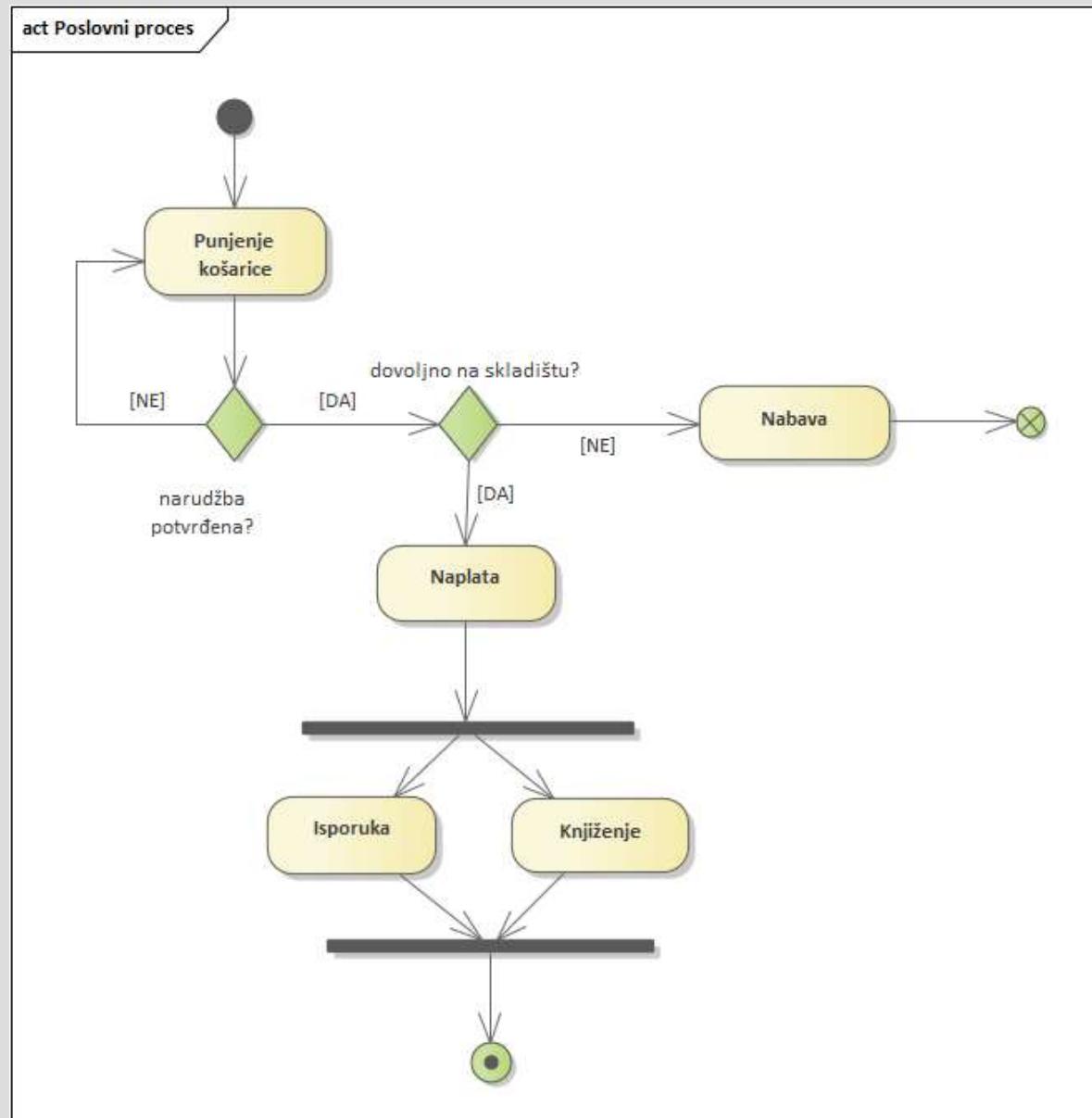
Arhitekturni kontekst sustava prodaje auto-dijelova



Gledište poslovnog procesa

- Kako se sustav koristi u širem poslovnom procesu → procesni model
- UML dijagram aktivnosti se koristi za prikaz poslovnog procesa

Poslovni kontekst prodaje autodijelova



MODEL I INTERAKCIJE

Modeli interakcije

- Modeliranje interakcije korisnika pomaže u definiranju korisničkih zahtjeva.
- Modeliranje interakcije između (dijelova) sustava naglašava probleme komunikacije koji se mogu pojaviti.
- Modeliranje interakcije između komponenti pomaže u otkrivanju hoće li predviđena struktura sustava postići željene performanse i pouzdanost.
- UML dijagrami obrazaca uporabe i dijagrami slijeda koriste se za modeliranje interakcija

Modeliranje obrazaca uporabe

- Obrasci uporabe
 - razvijeni kao potpora izlučivanju zahtjeva
- Svaki obrazac uporabe:
 - Predstavlja zasebni zadatak koji uključuje vanjsku interakciju sa sustavom
- Aktori:
 - ljudi
 - drugi sustavi
- Prikaz:
 - dijagramima
 - tekstualno

Dijagrami slijeda

- Modeliranje interakcije:
 - između aktora i objekata sustava
- Slijed interakcije koji se događa:
 - unutar obrazaca uporabe
- Objekti i aktori:
 - na vrhu dijagrama sa svojim životnim linijama
- Interakcije:
 - prikazane označenim strelicama

STRUKTURNI MODELI

Strukturni modeli

- Prikazuju organizaciju sustava u pogledu komponenti sustava i njihovih odnosa
- Mogu biti:
 - Statički – prikaz strukture oblikovanja sustava
 - Dinamički – prikaz organizacije sustavu tijekom rada
- Pri raspravi i oblikovanju arhitekture sustava

Dijagrami razreda

- Korišteni kod razvoja objektno-usmjerenog modela:
 - prikazuju razrede i njihove odnose
- Razred objekta:
 - predstavlja opću definiciju neke vrste objekta u sustavu
- Veze između razreda:
 - Pokazuju odnose između razreda
- U ranim stupnjevima razvoja:
 - Objekti predstavljaju neki entitet iz stvarnog svijeta

PONAŠAJNI MODELI

Ponašajni modeli

- Modeli dinamike ponašanja sustava pri radu:
 - prikazuju što se događa ili bi se trebalo događati prilikom odgovaranja sustava na poticaje iz okoline
- Dvije vrste poticaja:
 - **podaci** koji dolaze na obradu
 - **događaji** koji potiču ili okidaju procese u sustavu (mogu sadržavati i podatke)

Modeliranje upravljano podacima

- Mnogi poslovni procesi:
 - Upravljeni podacima koji ulaze u sustav
 - S malo obrade vanjskih događaja
- Modeli upravljeni podacima:
 - Pokazuju slijed akcija:
 - pri obradi ulaznih podataka
 - Pri generiranju izlaznih podataka
- Korisni pri analizi zahtjeva:
 - Prikaz cijelog ciklusa obrade unutar sustava

Modeliranje upravljano događajima

- Sustavi za rad u stvarnom vremenu:
 - Vođeni događajima
 - Malo obrade podataka
- Modeli upravljeni događajima:
 - pokazuju kako sustav reagira na vanjske i unutarnje događaje ili poticaje
- Pretpostavka postojanja ograničenog broja stanja sustava
 - prijelazi između stanja poticani događajima

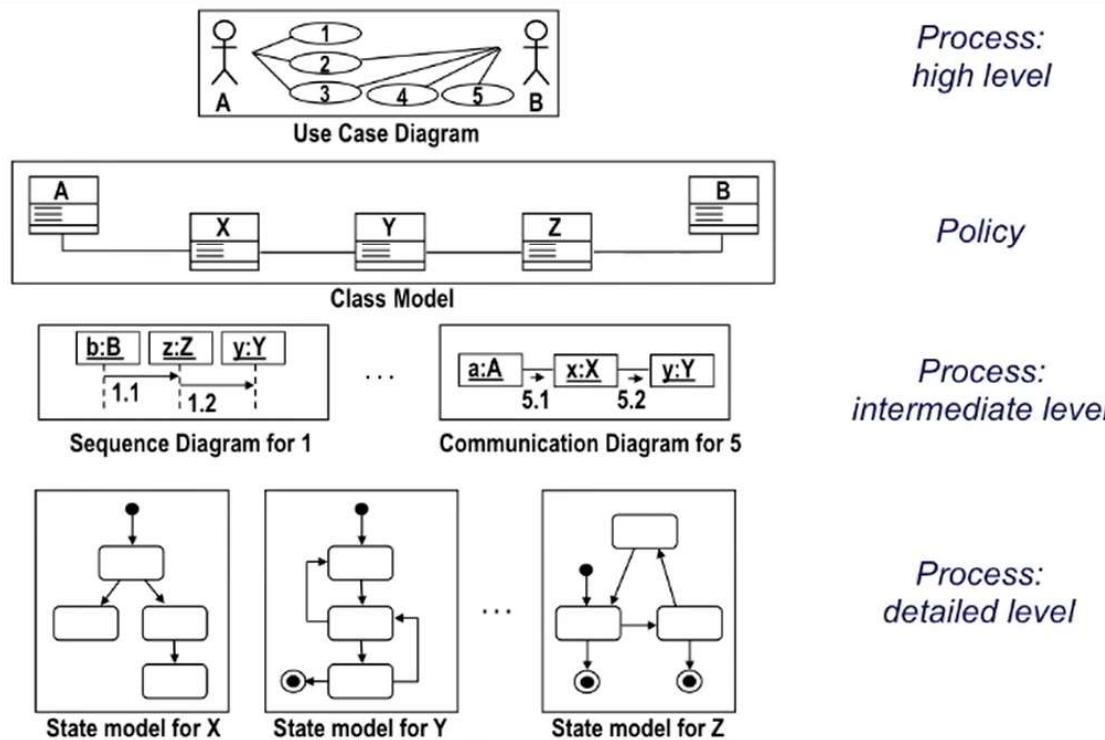
Modeli stroja stanja

- Prikazi stanja kao čvorova i događaja kao lukova između čvorova
- Dijagrami stanja: integralni dio UML standarda

„SEMANTIČKI MODEL”

Semantički model

Semantic Model Content



MODELNO-USMJERENO INŽENJERSTVO

Modelno-usmjерено inženjerstvo

- engl. *model-driven engineering* (MDE)
- rezultat: model, a ne program
- izvršni programi se automatski generiraju iz modela
- podizanje razine apstrakcije programskog inženjerstva iznad razine:
 - detalja programskog jezika
 - detalja ciljne implementacijske platforme

Primjena modelno-usmjerenog inženjerstva

- u ranoj fazi razvoja
- nije potpuno jasno koliko će još imati značajnog utjecaja na prakse programskog inženjerstva

Primjena modelno-usmjerenog inženjerstva

- prednosti
 - Pogled s viših razina apstrakcije
 - Generiranja koda bi značilo bržu prilagodbu sustava novim platformama.
- mane
 - Model apstrahiranja problem nisu nužno pravi modeli za implementaciju
 - Dodatni trošak razvoja prevodilaca za različite platforme

Modelno-usmjerena arhitektura

- engl. *model-driven architecture* (MDA)
 - preteča šireg pojma modelno usmjerenog inženjerstva
- za opis sustava koristi podskup UML modela
- modeli se stvaraju na različitim razinama apstrakcije
- platformski neovisni model ~> (bez ručne intervencije) funkcionalan program ?

Vrste modela (I)

- Računalno neovisan model
 - engl. *computation independent model* (CIM)
 - modeliranje domenskih apstrakcija → *domenski model*
- Platformski neovisan model
 - engl. *platform independent model* (PIM)
 - ne dotiče se implementacije
 - statička struktura sustava
 - ponašajni opis, kako reagira na vanjske i unutarnje događaje/poticaje

Vrste modela (II)

- Platformsko specifični model
 - engl. *platform specific model* (PSM)
 - transformacija platformsko neovisnog modela u zasebne platformsko specifične modele za svaku platformu
 - Mogu postojati i slojevi koji postepeno uključuju platformsko specifične detalje

Agilne metode i Modelno-usmjerena arhitektura

- iscrpno unaprijedno modeliranja je u proturječju s temeljnim idejama agilnog manifesta
- kad bi se postigla potpuna (100%-tna) automatizacija transformacija i cijeli programski kôd generirao iz platformsko neovisnog modela MDA bi se mogao koristiti u agilnom procesu jer ne bi bilo potrebno posebno kodiranje

Usvojenost modelno-usmjerene arhitekture

- ograničenja:
 - potrebna je potpora specijaliziranih alata za transformacije modela
 - potrebna je prilagodba postojećih alata (kojih nema puno) različitim okolinama
 - za dugoročne sustave razvijene modelno-usmjerenim pristupom potrebno je razvijati svoje alate

Usvojenost modelno-usmjerenе arhitekture

- koristi:
 - modeli olakšavaju rasprave o načinu oblikovanja programske potpore, ali te apstrakcije nisu one prave za implementacijske probleme
 - za većinu složenih sustava implementacija je puno manji problem od inženjerstva zahtjeva, sigurnosti, pouzdanosti, integracije s postojećim sustavima i procesa ispitivanja

Usvojenost modelno-usmjerene arhitekture

- stanje:
 - opravdanost pristupa neovisnosti o platformi vrijedi samo za velike dugoročne sustave
 - za manje proizvode i informacijske sustave ušteda uporabe modelno-usmjerene arhitekture se poništava troškovima njezinog uvođenja i podržavanja
 - široka prihvaćenost agilnih metoda je bacila modelno-usmjereni pristup u drugi plan

Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - interakcija
 - strukture
 - ponašanja

Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - prikazuju kako se sustav pozicionira unutar radne okoline i procesa
 - interakcija
 - strukture
 - ponašanja

Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - interakcija
 - koriste se dijagrami obrazaca uporabe i dijagrami slijeda
 - dijagrami obrazaca uporabe načelno prikazuju interakcije aktora sa sustavom
 - dijagrami slijeda prikazuju detalja interakcije
 - strukture
 - ponašanja

Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - interakcija
 - strukture
 - koriste se dijagrami razreda prikazuju organizaciju i strukturu sustava
 - ponašanja

Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - ...
 - ponašanja
 - dinamika kod izvršavanja sustava može se prikazati s gledišta obrade podataka ili događaja/poticaja
 - dijagrami aktivnost \rightarrow modeliranje obrade podataka
 - dijagrami stanja \rightarrow modeliranje ponašanja temeljenog na unutarnjim ili vanjskim događajima

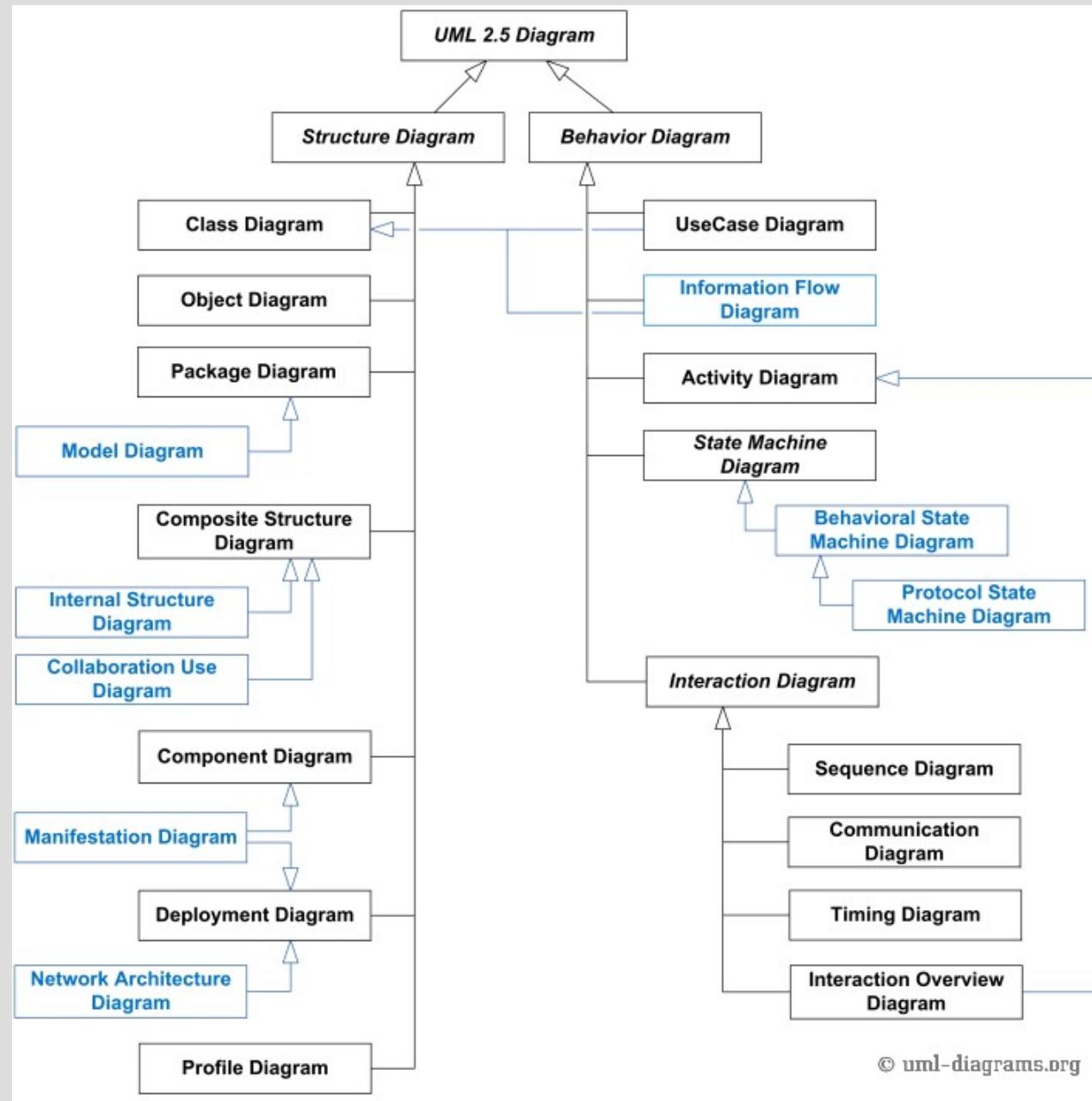
Zaključno

- model \equiv apstraktan pogled koji zanemaruje detalje
 - međusobno komplementarni modeli se razvijaju za prikaz:
 - konteksta
 - prikazuju kako se sustav pozicionira unutar radne okoline i procesa
 - interakcija
 - koriste se dijagrami obrazaca uporabe i dijagrami slijeda
 - dijagrami obrazaca uporabe načelno prikazuju interakcije aktora sa sustavom
 - dijagrami slijeda prikazuju detalja interakcije
 - strukture
 - koriste se dijagrami razreda prikazuju organizaciju i strukturu sustava
 - ponašanja
 - dinamika kod izvršavanja sustava može se prikazati s gledišta obrade podataka ili događaja/poticaja
 - dijagrami aktivnost se mogu koristiti za modeliranje obrade podataka
 - dijagrami stanja se koriste za modeliranje ponašanja temeljenog na unutarnjim ili vanjskim događajima

Zaključno

- Modelno-usmjereno inženjerstvo je pristup razvoju programske potpore gdje se sustav prikazuju u obliku skupa modela koji bi se automatski trebali pretvarati u izvršni kôd

Vrste dijagrama



Enterprise Architect

- <https://sparxsystems.com/>



“ It's a great tool, it provides all the essential features and more at a very reasonable price.

Keith McMillan | Adept Technologies Inc >>

Official Version: 15.2 Build 1558 2-Feb-2021

Enterprise Architect

Firstly... What is UML?

The Object Management Group (OMG) specification states:

"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

The important point to note here is that UML is a 'language' for specifying and not a method or procedure. The UML is used to define a software system; to detail the artifacts in the system, to document and construct - it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology (such as the Rational Unified Process) - but in itself it does not specify that methodology or process.

UML defines the notation and semantics for the following domains:

- The User Interaction or Use Case Model - describes the boundary and interaction between the system and users. Corresponds in some respects to a requirements model.
- The Interaction or Communication Model - describes how objects in the system will interact with each other to get work done.
- The State or Dynamic Model - State charts describe the states or conditions that classes assume over time. Activity graphs describe the workflows the system will implement.
- The Logical or Class Model - describes the classes and objects that will make up the system.
- The Physical Component Model - describes the software (and sometimes hardware components) that make up the system.
- The Physical Deployment Model - describes the physical architecture and the deployment of components on that hardware architecture.

<https://sparxsystems.com/resources/tutorials/uml/part1.html>

The UML also defines extension mechanisms for extending the UML to meet specialized needs (for example Business Process Modeling extensions).

Part 2 of this tutorial expands on how you use the UML to define and build actual systems.

REFERENCE I LITERATURA

- A. Jović, N. Frid, D. Ivošević: Procesi programskog inženjerstva, 3. izdanje, Sveučilište u Zagrebu, FER ZEMRIS, 2019.
- I. Sommerville, Software engineering, 10th ed., Pearson, 2016.,
<https://iansommerville.com/software-engineering-book/>
- S. Tockey: How to Engineer Software, Wiley-IEEE Computer Society Press, 2019.

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19970

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UNIFIED MODELING LANGUAGE (UML)

Unified Modeling Language

- UML je jezik za:
 - vizualizaciju;
 - specifikaciju;
 - oblikovanje i
 - dokumentiranje artefakata programske potpore.
- Omogućava različite poglede na model
- ‘de facto’ standardni jezik programskog oblikovanja
- UML je posebice prikladan za specificiranje *objektno usmjerenе* arhitekture programske potpore.
- Dijelovi UML-a pogodni su u specificiranju i drugih arhitektura.
- Omogućava:
 - Višestruke međusobno povezane poglede
 - Poluformalnu semantiku izraženu kao meta-model
 - Jezik za opis formalnih logičkih ograničenja

Temelji UML-a

- **UML** je jezik za:
 - Vizualizaciju
 - Specifikaciju
 - Oblikovanje (i konstruiranje)
 - Dokumentiranje artefakata programske potpore.
- UML je posebice prikladan za specificiranje **objektno usmjerene arhitekture programske potpore**.
- Dijelovi UML-a pogodni su u specificiranju i drugih arhitektura.



- **Arhitekture programske potpore** je struktura ili strukture sustava koji sadrži elemente, njihova izvana vidljiva obilježja i odnose između njih.

Prednosti UML-a

- Otvoren standard.
- Podupire cijeli životni ciklus oblikovanja programske potpore.
- Podupire različite domene primjene.
- Temeljen je na iskustvu i potrebama zajednice oblikovatelja i korisnika programske potpore.
- Podržavaju ga mnogi alati.

Mnogo dionika, mnogo pogleda

- Arhitekturu programske potpore različito vidi:
 - Korisnik-kupac
 - Projekt manager
 - Inženjer sustava
 - Osoba koje razvija sustav
 - Arhitekt
 - Osoba koja održava sustav (engl. *maintainer*)
 - Drugi dionici
- **Višedimenzionalna stvarnost**
- Mnogo dionika rezultira u
 - višestrukim pogledima, višestrukim nacrtima

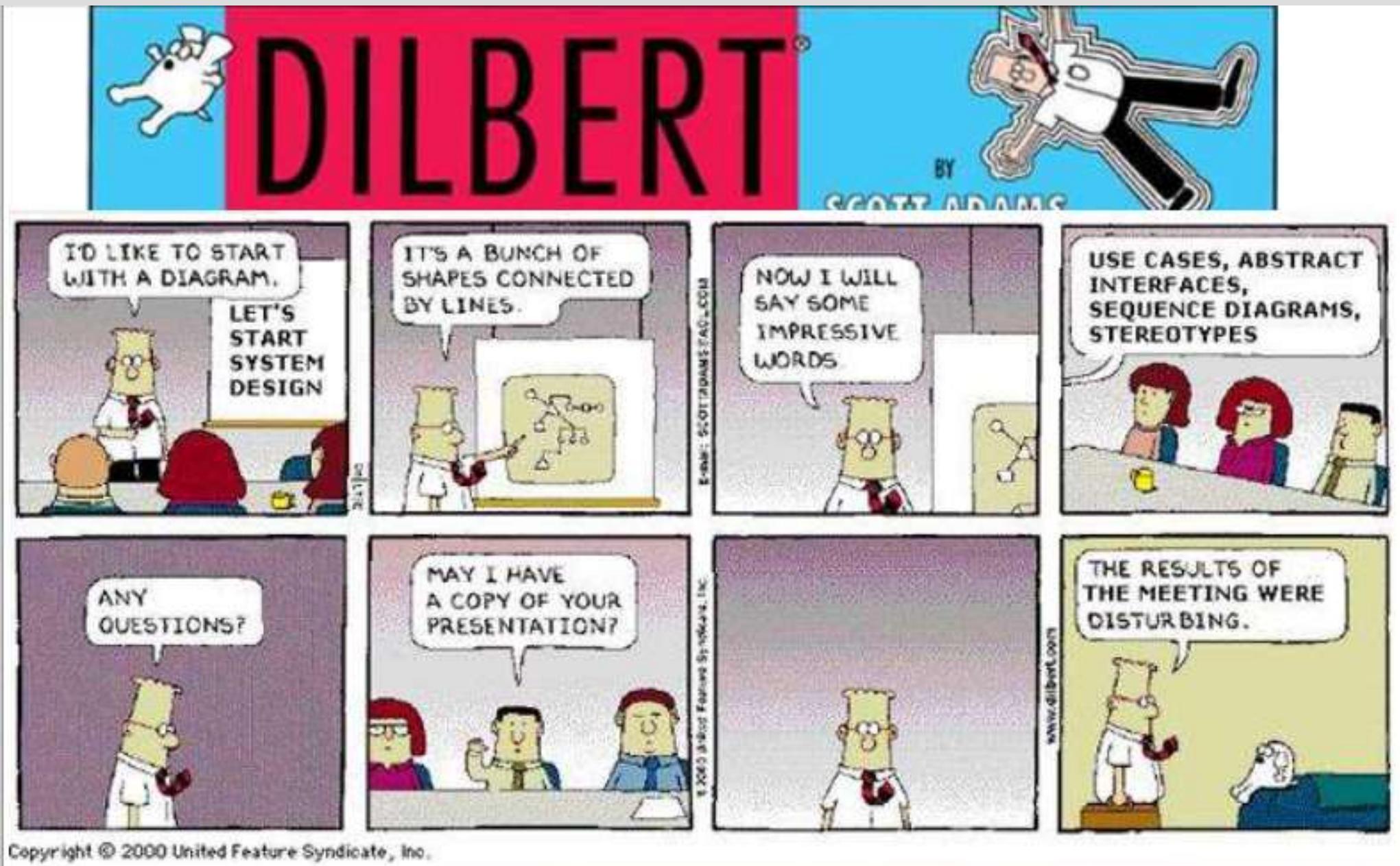
Koliko pogleda ?

- Pogledi odgovaraju nekom kontekstu.
- Svi sustavi ne trebaju sve poglede:
 - Jedan procesor, nije potreban nacrt razmještaja procesora.
 - Jedan proces, nije potreba nacrt razmještaja procesa.
 - Vrlo mali programi, nije potreban nacrt implementacije.
- Neki dodatni pogledi:
 - Pogled podataka, pogled sigurnosti u sustavu
- Svaki pogled dokumentira se nacrtom – **DIJAGRAMOM.**
- Više pogleda (dijagrama) u okviru nekog konteksta = **MODEL**
- **Model je apstraktan (reduciran, pojednostavljen, smanjen, ogranicen) opis sustava iz odredene perspektive.**

UML dijagrami

- Pojedini dijagram je pogled u model
 - Prikazan s polazišta određenog dionika
 - Daje određeno predstavljanje sustava
 - Semantički je dosljedan s ostalim pogledima
- U okviru UML v1.3 postoji devet standardnih dijagrama:
 - Statički pogledi: obrazaca uporabe, razreda, objekata, komponenti, razmještaja
 - Dinamički pogledi: slijeda, komunikacije, stanja, aktivnosti
 - U RUP procesu svakoj aktivnosti pridružen je jedan ili više modela za opis, koji su dokumentirani s jednim ili više dijagrama (pogleda, engl. *views*).

Percepcija UML-a



Potreba

- Osmisliti jezik koji osigurava jednostavan rječnik i pravila kombiniranja riječi u svrhu komunikacije.
- U jeziku modeliranja rječnik i pravila su usmjerena na konceptualnu i fizičku reprezentaciju sustava.
- UML standard
 - "A language provides a *vocabulary* and *the rules for combining words* [...] for the purpose of *communication*. A *modeling language* is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. A modeling language such as the UML is thus a standard language for *software blueprints*."

From "UML user guide"

Osnovna svojstva

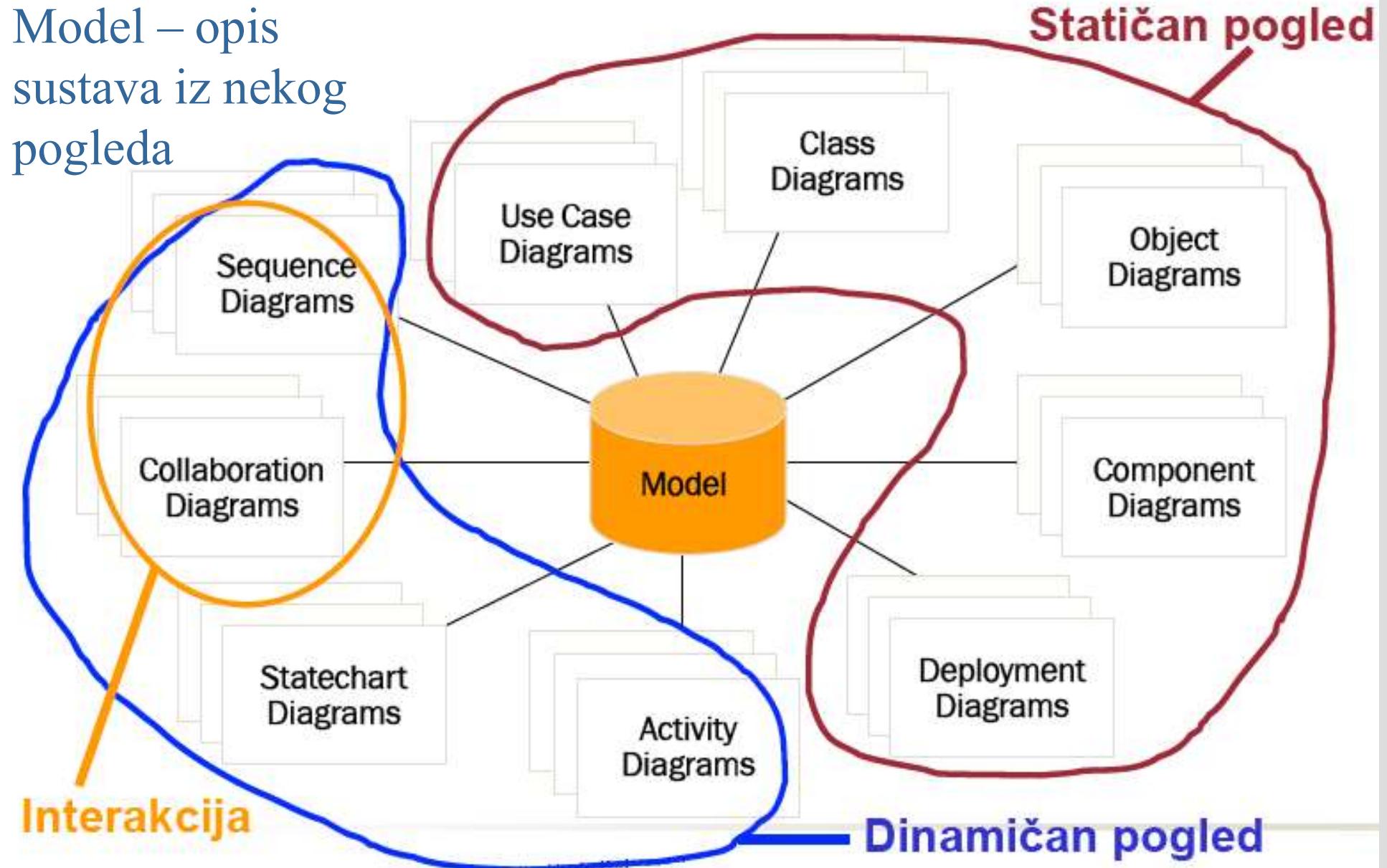
- Osnovne ideje:
 - obuhvatiti i opisati poslovne procese
 - poboljšati komunikaciju
 - pomoći u borbi s kompleksnošću
 - definirati logičku arhitekturu sustava omogućiti ponovno korištenje
- Namjena:
 - vizualizacija
 - specificiranje
 - konstrukcija
 - dokumentiranje

Osnovni elementi UML-a

- Stvari (*engl. things*)
 - strukturne stvari (*engl. structural things*)
 - razred; sučelje; suradnja; obrasci uporabe; aktivni razred; komponenta; čvor
 - aktori, signali, procesi i niti, aplikacije, dokumenti, datoteke, biblioteke, stranice, tablice
 - stvari ponašanja (*engl. behavioral things*)
 - interakcija; stanje
 - stvari grupiranja (*engl. grouping things*)
 - organizacija elemenata u grupe (samo konceptualno, hijerarhija), paketi
 - stvari označavanja (*engl. annotation things*)
 - opisni elementi, formalni/neformalni opis
- Relacije (*engl. relationships*)
 - Pridruživanja (*eng. association*)
 - Poopćenja (*eng. generalization*)
 - Realizacije (*eng. realization*)
 - Ovisnosti (*eng. dependency*)
- Dijagrami (*engl. diagrams*)
 - Dijagram razreda; objekata; slučajeva korištenja; toka; suradnji; stanja; aktivnosti; komponenata;....

Modeli, pogledi, dijagrami

Model – opis
sustava iz nekog
pogleda



Dijagrami

- Pogled na model
 - Iz perspektive dionika
 - Djelomična reprezentacija sustava
 - Semantički dosljedan s ostalim pogledima
- Standardni dijagrami (UML 1.1)
 - statični: dijagram obrazaca uporabe, dijagram razreda, dijagram objekata, dijagram komponenata, dijagram razmještaja
 - dinamični: dijagram slijeda, komunikacijski dijagram, dijagram stanja (“statechart”), dijagram aktivnosti
 - uz manje promjene najčešće upotrebljavani dijagrami i u novijim inačicama UML-a
- Specifičnost dijagrama
 - Svaki ima vlastitu sintaksu i semantiku ☺ ?
- Evolucija UML-a
 - veljača 2009. UML 2.2
 - ožujak 2011. UML 2.4-beta
 - → standard 2.5

Kada koristimo određene vrste dijagrama?

- **U izradi projekata i projektne dokumentacije** naglasak je najčešće na sljedećim dijagramima:
 - **dijagram obrazaca uporabe**
 - **dijagram slijeda**
 - **dijagram aktivnosti**
 - **dijagram razreda**
- **Za rješavanje arhitekture sustava** koriste se i drugi dijagrami:
 - **dijagram objekata, dijagram stanja, dijagram aktivnosti, komunikacijski dijagram, dijagram komponenti**
- **U implementacijskom dijelu dokumentacije** koristi se i **dijagram razmještaja**

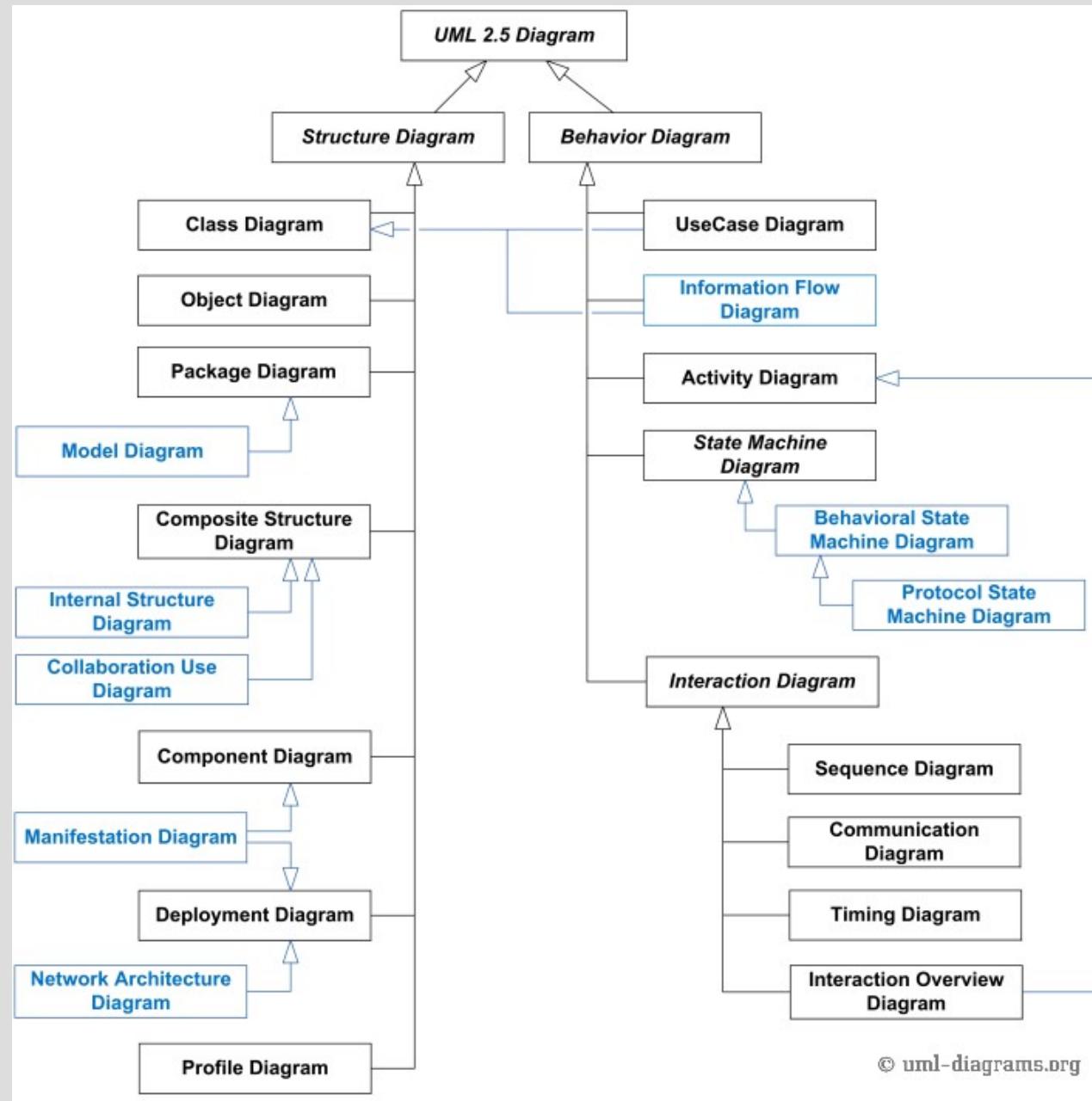
UML CASE-alati

- Enterprise Architect
- Visual Paradigm
- ArgoUML
- Astah Community
- plantUML
- Draw.io
- StarUML
- MS Visio
- ...

UML Standard

- Trenutno: 2.5.1
- Specifikacija:
<https://www.omg.org/spec/UML/2.5.1/PDF>
(796 stranica)

Vrste dijagrama



Enterprise Architect

- <https://sparxsystems.com/>

The logo for Enterprise Architect. It features a blue circular icon with three white curved segments forming a stylized 'S' or gear-like shape. To the right of the icon, the words "ENTERPRISE" and "ARCHITECT" are written in large, bold, blue and orange sans-serif fonts respectively. Below this, the words "create | verify | share" are written in a smaller, orange sans-serif font.

ENTERPRISE ARCHITECT
create | verify | share

“ It's a great tool, it provides all the essential features and more at a very reasonable price.

Keith McMillan | Adept Technologies Inc >>

Official Version: 15.2 Build 1558 2-Feb-2021

Enterprise Architect

Firstly... What is UML?

The Object Management Group (OMG) specification states:

"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."

The important point to note here is that UML is a 'language' for specifying and not a method or procedure. The UML is used to define a software system; to detail the artifacts in the system, to document and construct - it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology (such as the Rational Unified Process) - but in itself it does not specify that methodology or process.

UML defines the notation and semantics for the following domains:

- The User Interaction or Use Case Model - describes the boundary and interaction between the system and users. Corresponds in some respects to a requirements model.
- The Interaction or Communication Model - describes how objects in the system will interact with each other to get work done.
- The State or Dynamic Model - State charts describe the states or conditions that classes assume over time. Activity graphs describe the workflows the system will implement.
- The Logical or Class Model - describes the classes and objects that will make up the system.
- The Physical Component Model - describes the software (and sometimes hardware components) that make up the system.
- The Physical Deployment Model - describes the physical architecture and the deployment of components on that hardware architecture.

<https://sparxsystems.com/resources/tutorials/uml/part1.html>

The UML also defines extension mechanisms for extending the UML to meet specialized needs (for example Business Process Modeling extensions).

Part 2 of this tutorial expands on how you use the UML to define and build actual systems.

REFERENCE I LITERATURA

- A. Jović, N. Frid, D. Ivošević: Procesi programskog inženjerstva, 3. izdanje, Sveučilište u Zagrebu, FER ZEMRIS, 2019.
- I. Sommerville, Software engineering, 8th ed., Addison Wesley, 2007.
- G. Overgaard, B. Selic, C. Bock, OMG, 2000.
- S. Tockey: How to Engineer Software, Wiley-IEEE Computer Society Press, 2019.

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI OBRAZACA UPORABE

Obrasci uporabe

- Koristi se naziv: **dijagram obrazaca uporabe** ili **dijagram slučaja korištenja** za engleski pojam ***use case diagram***
- Ostali hrvatski termini:
 - Dijagram korištenja sustava
 - Dijagram slučaja uporabe
- Dva značenja riječi „obrazac uporabe”
 - Pojedini dio funkcionalnosti sustava (pojedini obrazac uporabe):
 - Više funkcionalnosti + aktori:

Obrasci uporabe

- Pogled na sustav koji naglašava njegovo vanjsko ponašanje prema korisniku
- Dijeli funkcionalnost sustava u skup obrazaca uporabe (transakcija) koji su značajni korisniku (aktoru)
- Skup obrazaca uporabe opisuje sve moguće *interakcije sustava*.
- Uz obrasce uporabe, dodatno se mogu koristiti i dijagrami slijeda kako bi se detaljno opisao tijek događaja.

Obrasci uporabe

- Dijagram obrazaca uporabe opisuje ponašanje sustava iz perspektive vanjskog korisnika, tj. “Tko može činiti što u sustavu”
- Pritom se na sustav gleda kao na crnu kutiju određene razine apstrakcije
- **Korisnici sustava** se u dijagramu nazivaju **aktori** i mogu biti ljudi (klijenti sustava), vanjski računalni sustavi, unutarnji računalni sustavi i sl.
- Aktor je uloga koju korisnik ima pri komunikaciji sa sustavom.

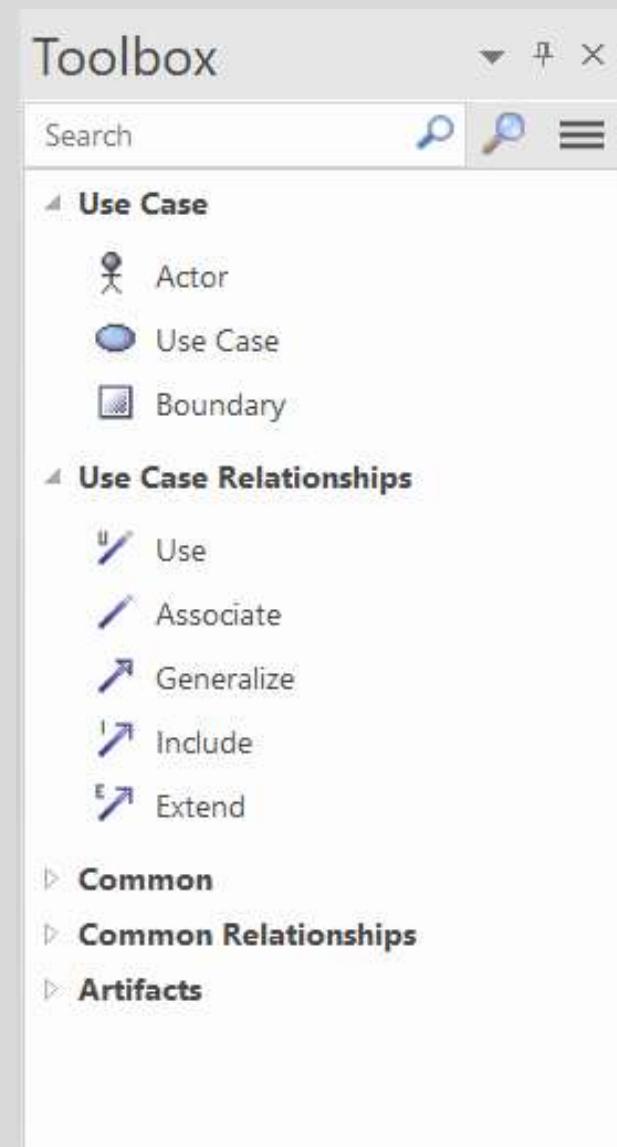


Obrasci uporabe

- Modeliraju se zahtjevi korisnika i scenariji ispitivanja sustava
- Ne koristi se nijedan implementacijsko-specifičan jezik i uvijek se izrađuje na određenoj razini detalja, razine se ne miješaju na istom dijagramu
- **Razina apstrakcije je najčešće visoka, konceptualna**
- **Služe za opis funkcionalnih zahtjeva projekta** i to:
 - Svih aktora, bili oni aktivni aktori (inicijatori) ili pasivni aktori (sudionici)
 - Svih načina rada sustava - scenariji
- Prema potrebi, crta se više dijagrama da se obuhvate svi dijelovi sustava

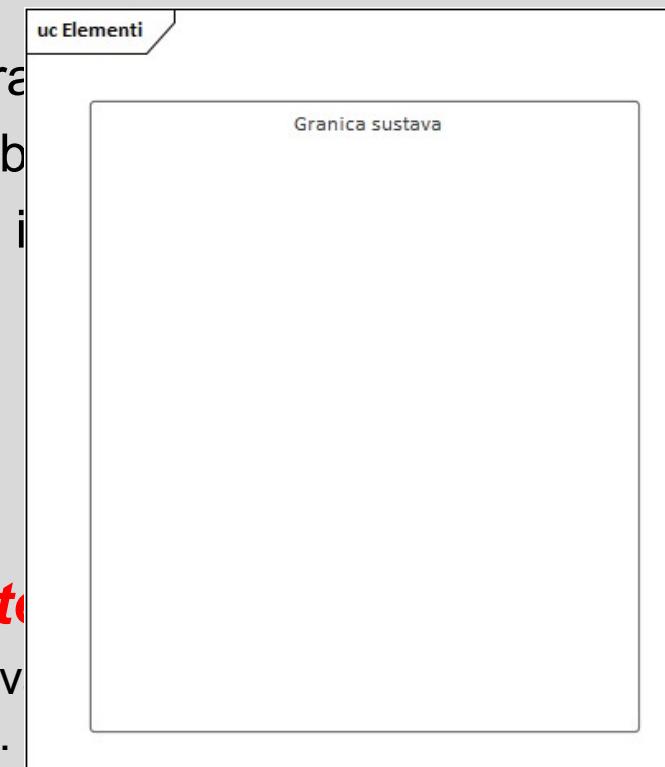
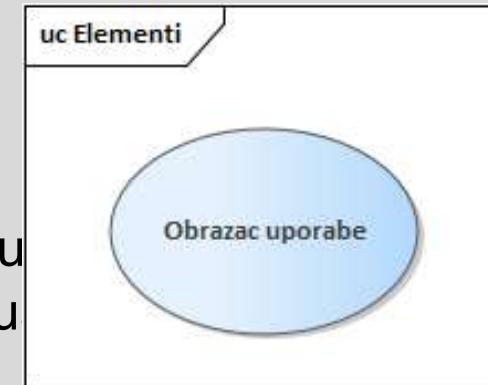
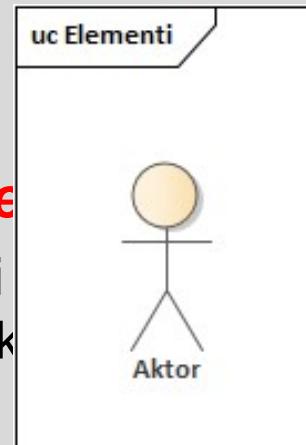
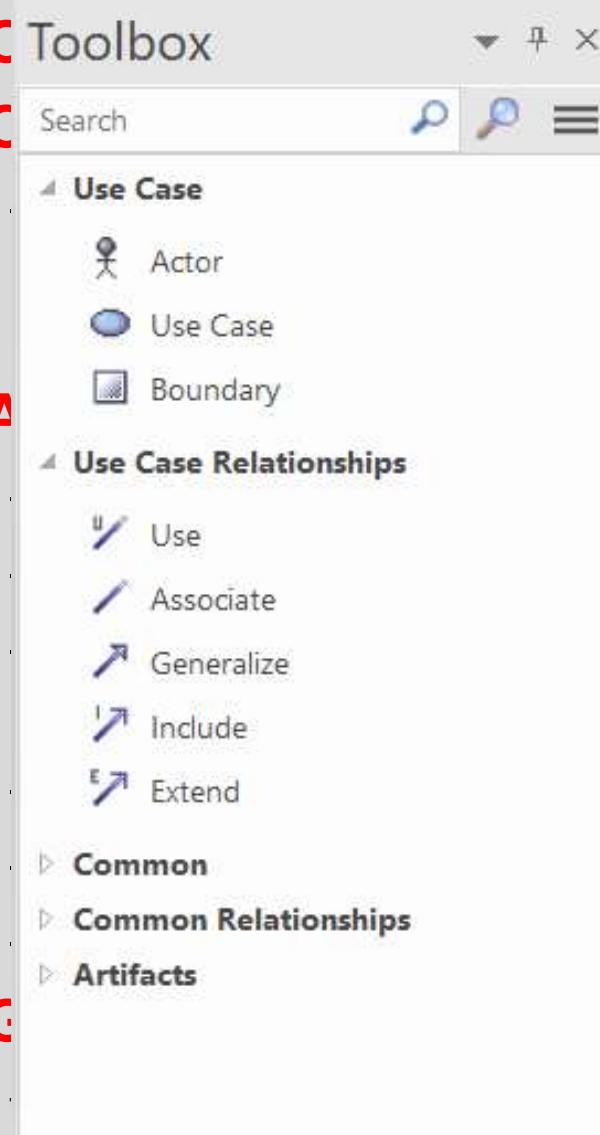
Sadržaj obrazaca uporabe

- Elementi
- Odnosi



Elementi obrazaca uporabe

- C Toolbox



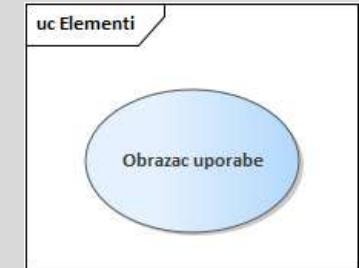
interakciji s fizikalnim sustavom.

Elementi obrazaca uporabe

- **Osnovni elementi:**

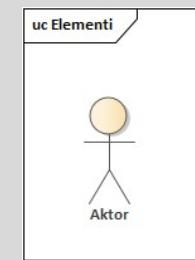
- **Obrazac uporabe (engl. *use case*)**

- Niz akcija ili funkcionalnosti koje sustav ili drugi entitet obavlja u interakciji s aktorima sustava



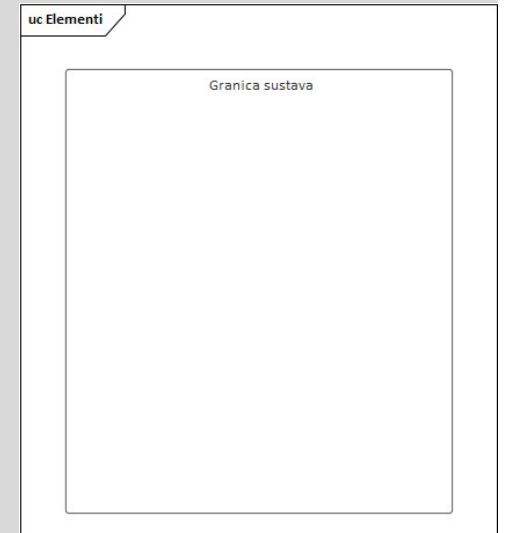
- **Aktor (engl. *actor*)**

- Vanjski objekt koji komunicira sa sustavom
 - Jedinstveno ime (+ po potrebi opis)
 - Jednoznačan skup uloga koje imaju korisnici u interakciji s obrascima uporabe.
 - Uloga korisnika u sustavu
 - Jeden korisnik – više aktora
 - Vanjski sustav



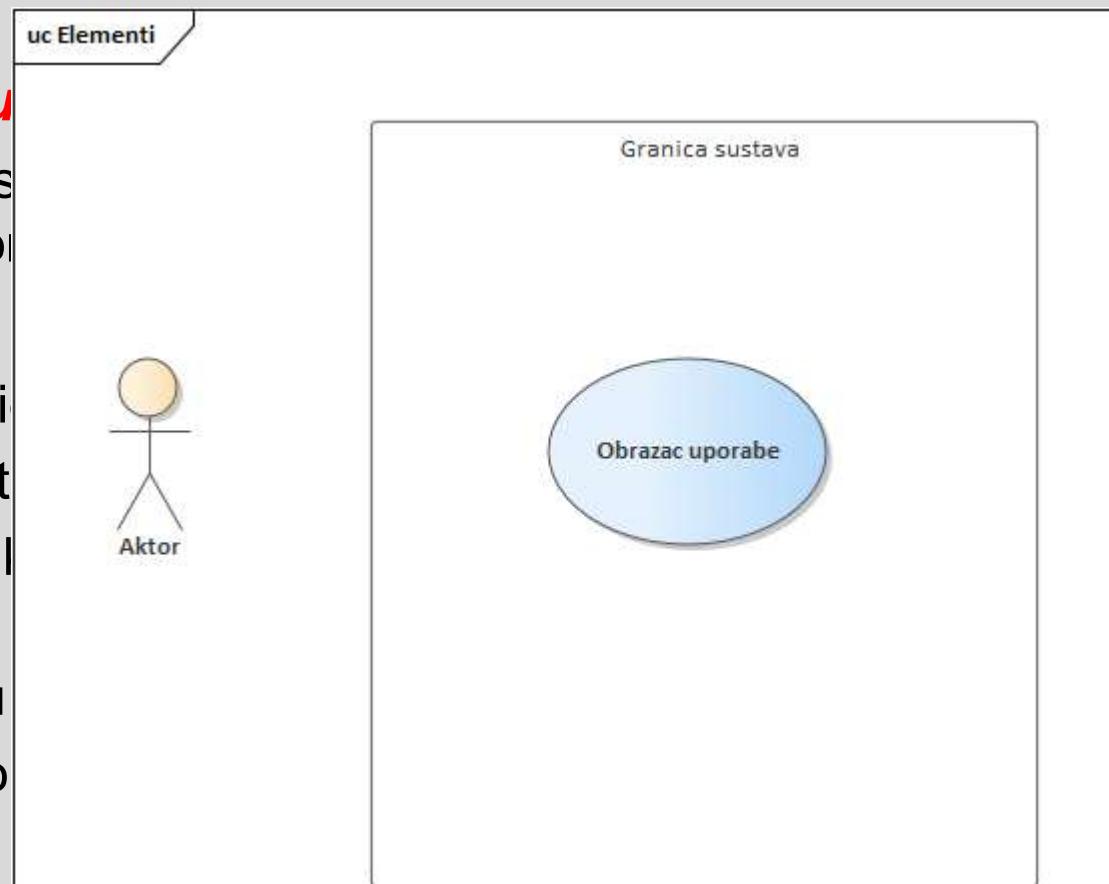
- **Granica sustava (engl. *system boundary*)**

- granica između fizičkog sustava i različitih aktora koji su u interakciji s fizičkim sustavom.



Elementi obrazaca uporabe

- **Osnovni elementi:**
- **Obrazac uporabe (engl. use case)**
 - Niz akcija ili funkcionalnosti koju obavlja u interakciji s aktorom.
- **Aktor (engl. actor)**
 - Vanjski objekt koji komunicira s sistemom.
 - Jedinstveno ime (+ po potrebi podjela na uloge).
 - Jednoznačan skup uloga i interakcija s obrascima uporabe.
 - Uloga korisnika u sustavu.
 - Jedan korisnik – više aktora.
 - Vanjski sustav
- **Granica sustava (engl. system boundary)**
 - granica između fizičkog sustava i različitih aktora koji su u interakciji s fizičkim sustavom.



Obrazac uporabe

- Imenovanje: **glagol + imenica** ili **glagolska imenica**
- Definira doseg sustava i funkcionalnost koju podržava u sustavu te elemente o kojima ovisi
- Potpomaže aktorima u svrhu ostvarivanja ciljeva
 - Obrasci uporabe predstavljaju funkcionalnosti sustava i odgovornosti (*engl. responsibilities*)

Aktori

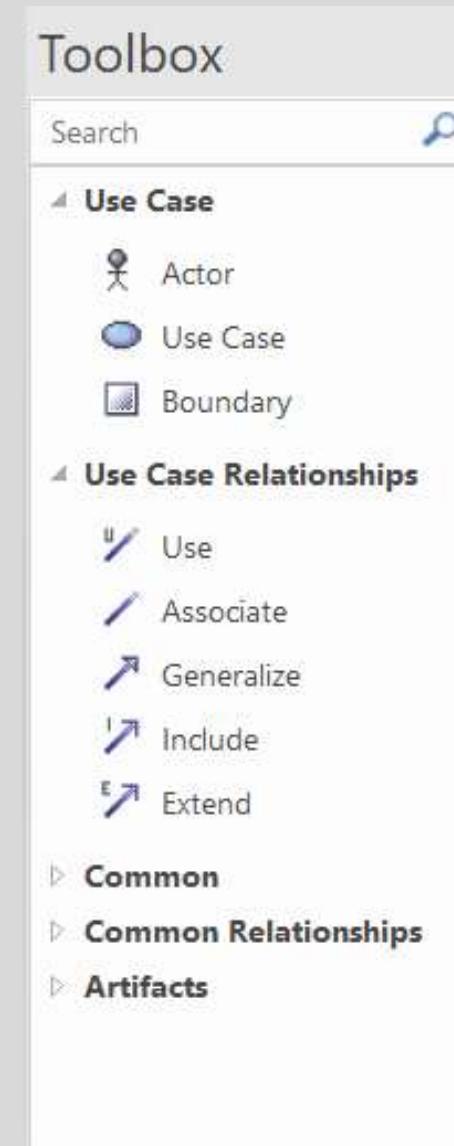
- **Za imenovanje upotrijebiti imenice**
- Jedinstven naziv na razini dijagrama
- Potrebno opisati njihovu ulogu u interakciji sa sustavom
- Definirati opseg sustava, identificirati elemente na rubu sustava i elemente o kojima sustav ovisi
- Obrasci uporabe se pišu iz točke gledišta aktora!
 - Što radi koji aktor?
 - Za aktivnost aktora definira se obrazac uporabe
 - Opišite ju tekstom
 - Zadržite razinu apstrakcije

Organizacija i odnosi

- Hijerarhijska organizacija:
 - Dozvoljeno slaganje dijagrama u pakete (engl. *packages*)
 - Određivanje odnosa među dijagramima (engl. *relations*)
- Odnosi među elementima:
 - *include* (obuhvaćanje, uključivanje podaktivnosti)
 - *extend* (proširivanje mogućnosti)
 - *generalization* (poopćenje, generalizacija)
 - *association* (pridruživanje)

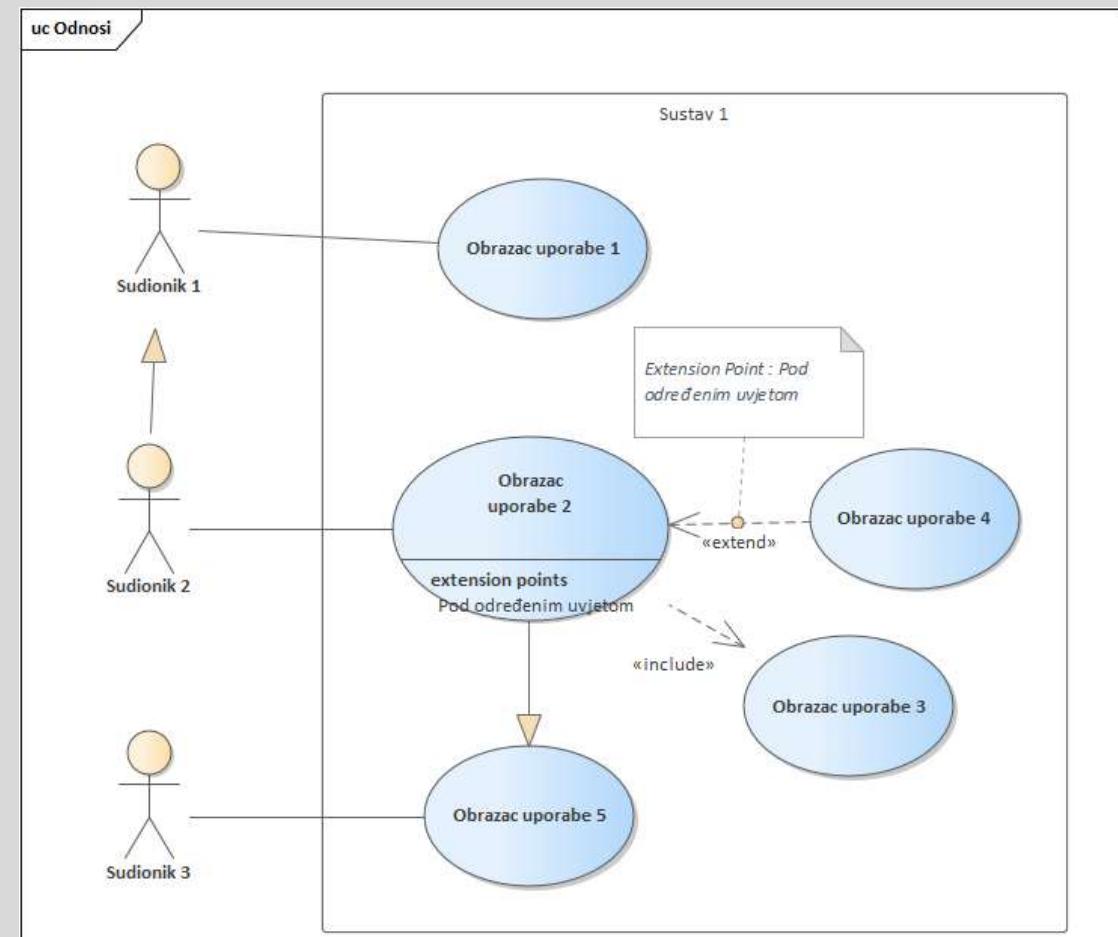
Odnosi

- **Pridruživanje ili asocijacija (*association*)**
 - Komunikacija instance aktora i instance obrasca uporabe
- **Proširenje (*extend*)**
 - Odnos od proširene uporabe do osnovne uporabe
- **Obuhvaćanje/uključivanje, nužno sadrži (*include*)**
 - Odnos od osnovnog obrasca do uključenog obrasca
 - Definira ponašanje uključenog obrasca u odnosu na osnovni obrazac uporabe.
 - Osnovni obrazac sadrži ponašanje definirano u drugom obrascu.
- **Generalizacija ili poopćenje (*generalization*)**
 - Odnos između više općenitog i više specifičnog



Odnosi

- Tko sve koristi „Obrazac uporabe 1”?
- Kada se koristi „Obrazac uporabe 3”?
- Kada se koristi „Obrazac uporabe 4”?



Odnos pridruživanja

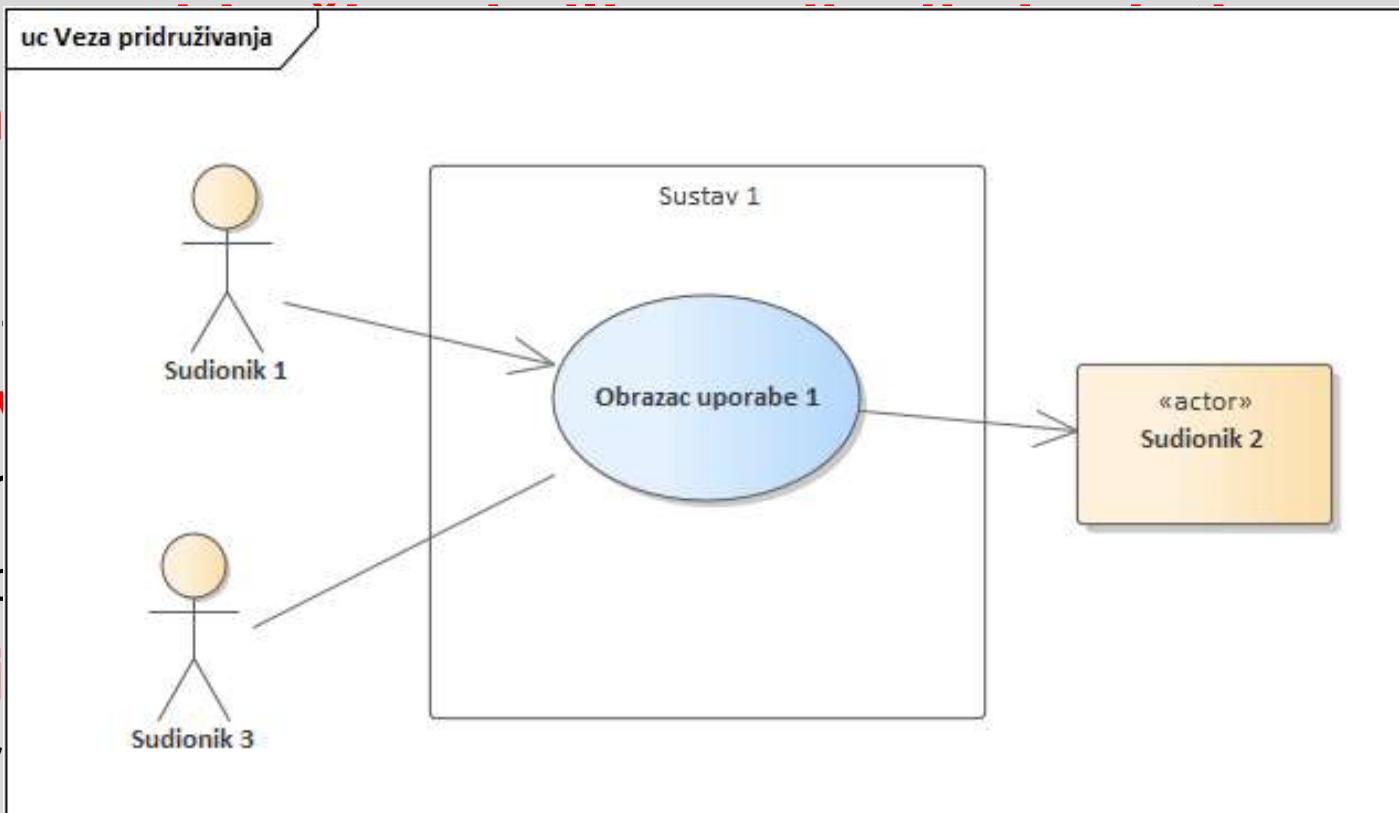
- **Odnos pridruživanje ili asocijacija koristi se za komunikaciju između aktora i obrasca uporabe**
- Ako nije drugačije navedeno, komunikacija aktor-obrazac uporabe je dvosmjerna
- **Aktivni aktori** (inicijatori): smjer strelice je od aktora prema obrascu uporabe
 - npr. klijent, administrator
- **Pasivni aktori** (sudionici) smjer strelice je od obrasca uporabe prema aktoru
 - npr. arhiva podataka, baza podataka, pisač
- Aktor može biti aktivan i pasivan istodobno.
 - Prepoznaje se po dvosmjernoj vezi

Vrste aktora

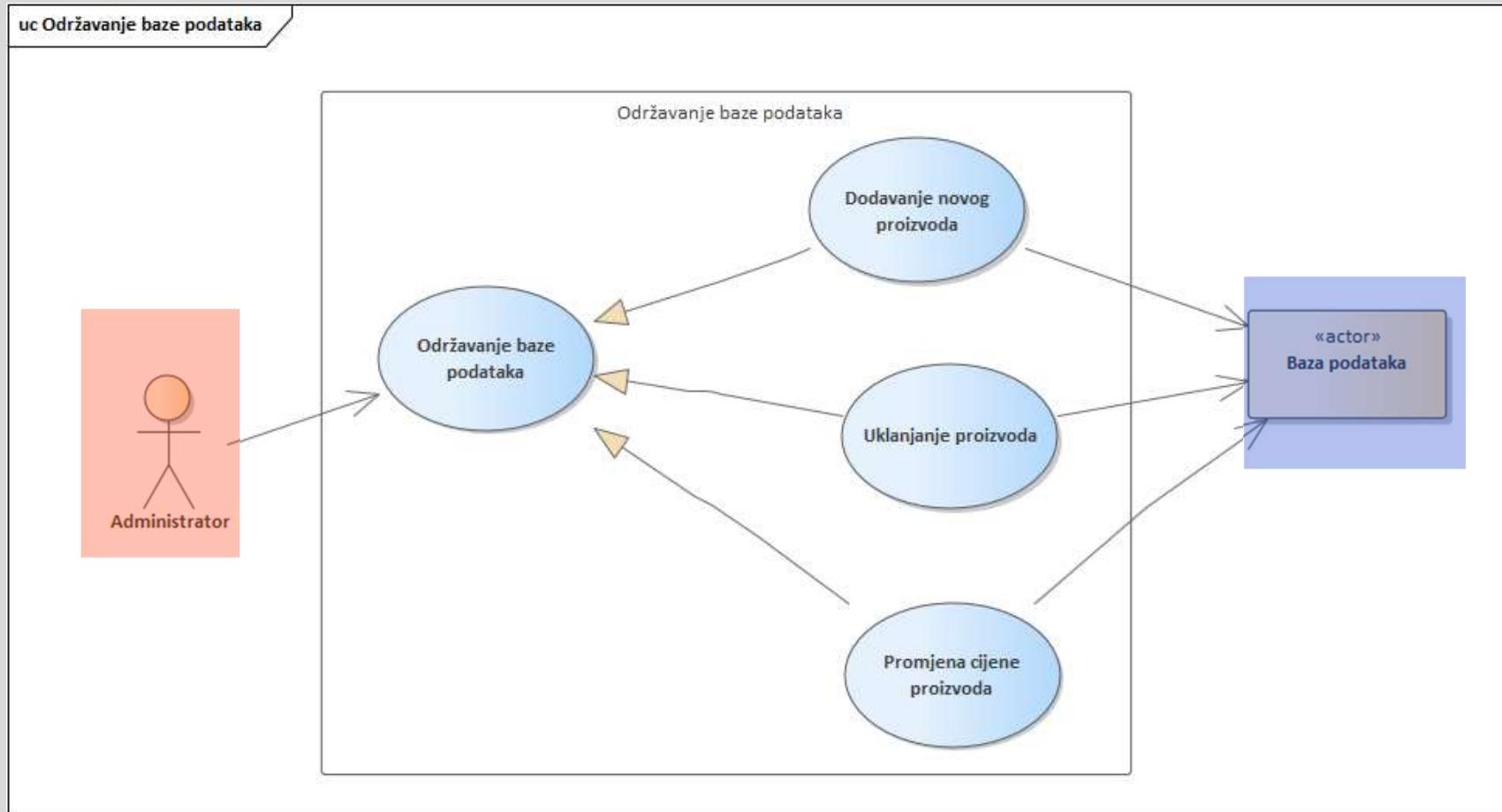
- Primarni – generiraju podatke ili primaju informacije
 - **Aktivni aktori** – inicijatori: smjer strelice od aktora prema obrascu uporabe
 - npr. klijent, administrator
 - **Pasivni aktori** – sudionici: smjer strelice od obrasca uporabe prema aktoru
 - npr. arhiva podataka, baza podataka, pisač
- Sekundarni – pružaju podršku radu sustava

Odnos pridruživanja

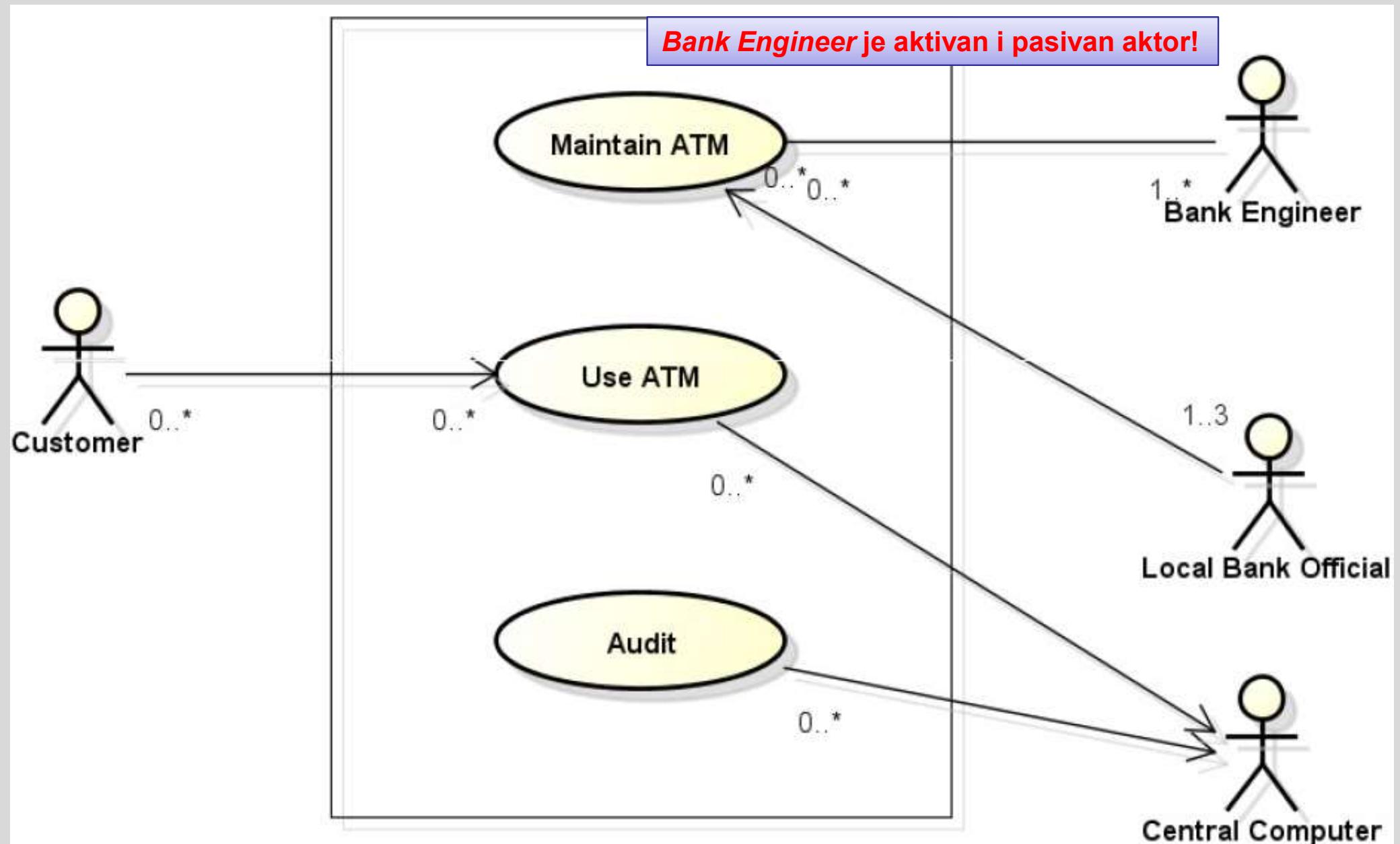
- **Odnos pridruživanja** je veza između aktora i sistemom.
- Ako aktor upravlja sistemom
- **Aktivni sudionici** premeštaju se u sistem
– npr. administrator
- **Pasivi sudionici** uporabljaju sistem
– npr. arhiva podataka, baza podataka, pisač
- Aktor može biti aktivan i pasivan istodobno.
 - Prepoznaje se po dvosmjerenoj vezi



Aktivni i pasivni aktori - primjer



Primjer aktivnih i pasivnih aktora

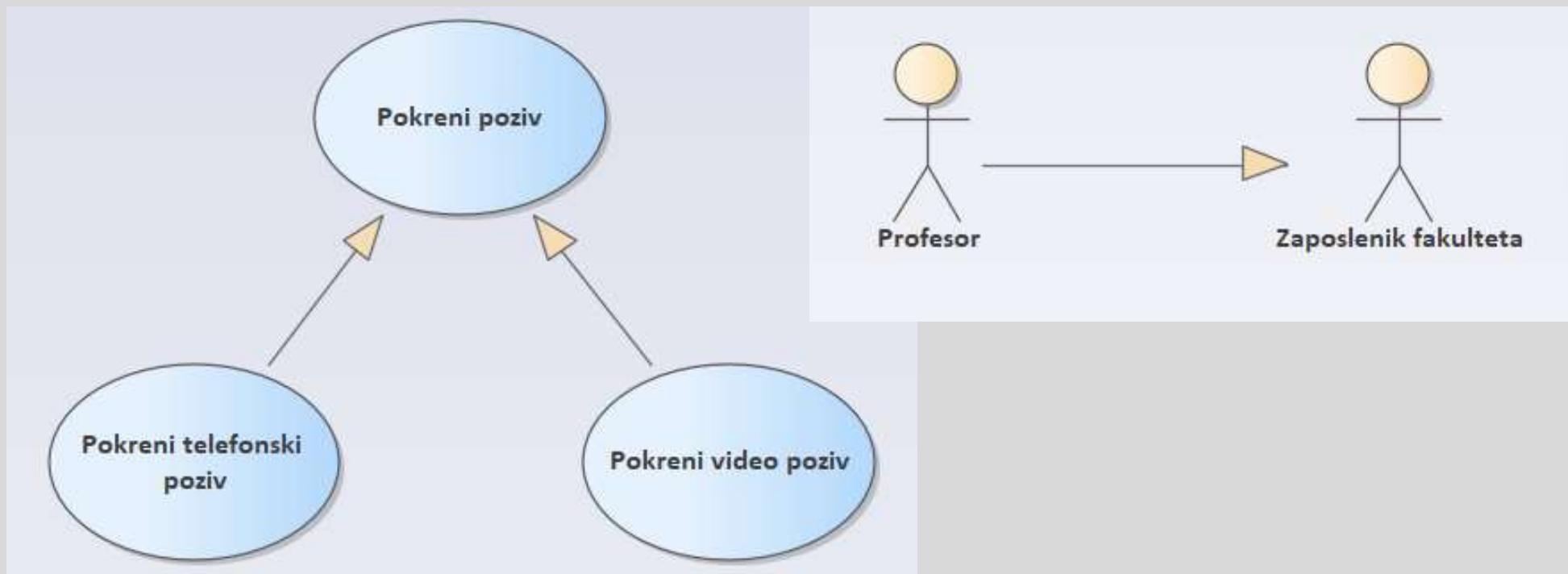


Odnos poopćenja

- Poopćenje prikazuje hijerarhijski odnos između više općenitog obrasca uporabe i više detaljnog obrasca uporabe, a za razliku od ostalih odnosa, moguća je i među aktorima
- Obično se koristi kad se ne zna ništa detaljnije o međusobnom odnosu osim toga da je jedan obrazac specijalan slučaj drugoga

Odnos poopćenja

- Pritom specifičniji obrazac dodaje nove ili mijenja dijelove postojeće funkcionalnosti apstraktnijeg obrasca:

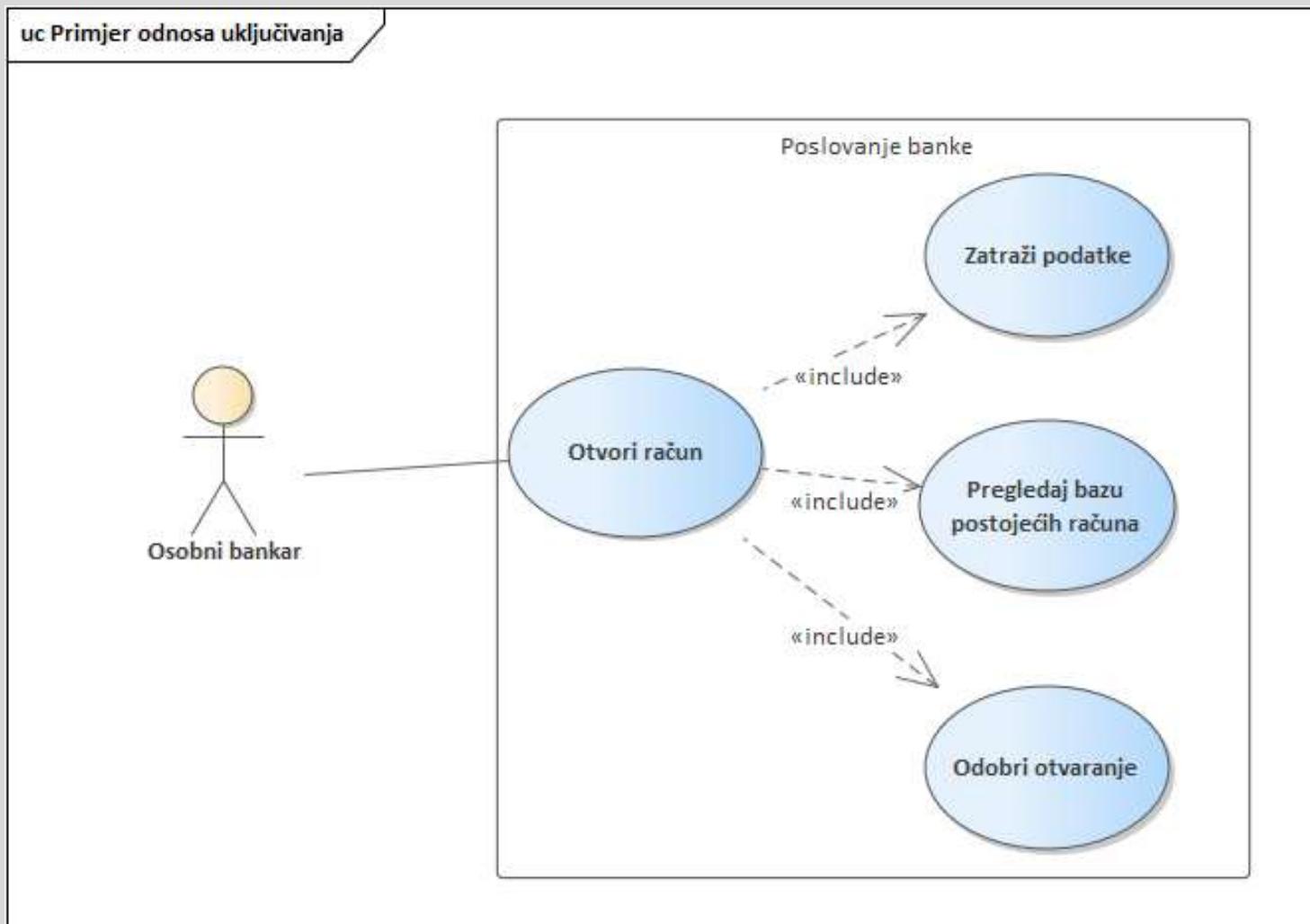


Odnos obuhvaćanja/uključivanja

- Veza od osnovnog obrasca uporabe prema uključenom obrascu uporabe
- Osnovni obrazac obavlja cijeli uključeni obrazac u nekom neodređenom trenutku izvođenja

Odnos obuhvaćanja/uključivanja

- Primjer ispravne uporabe:

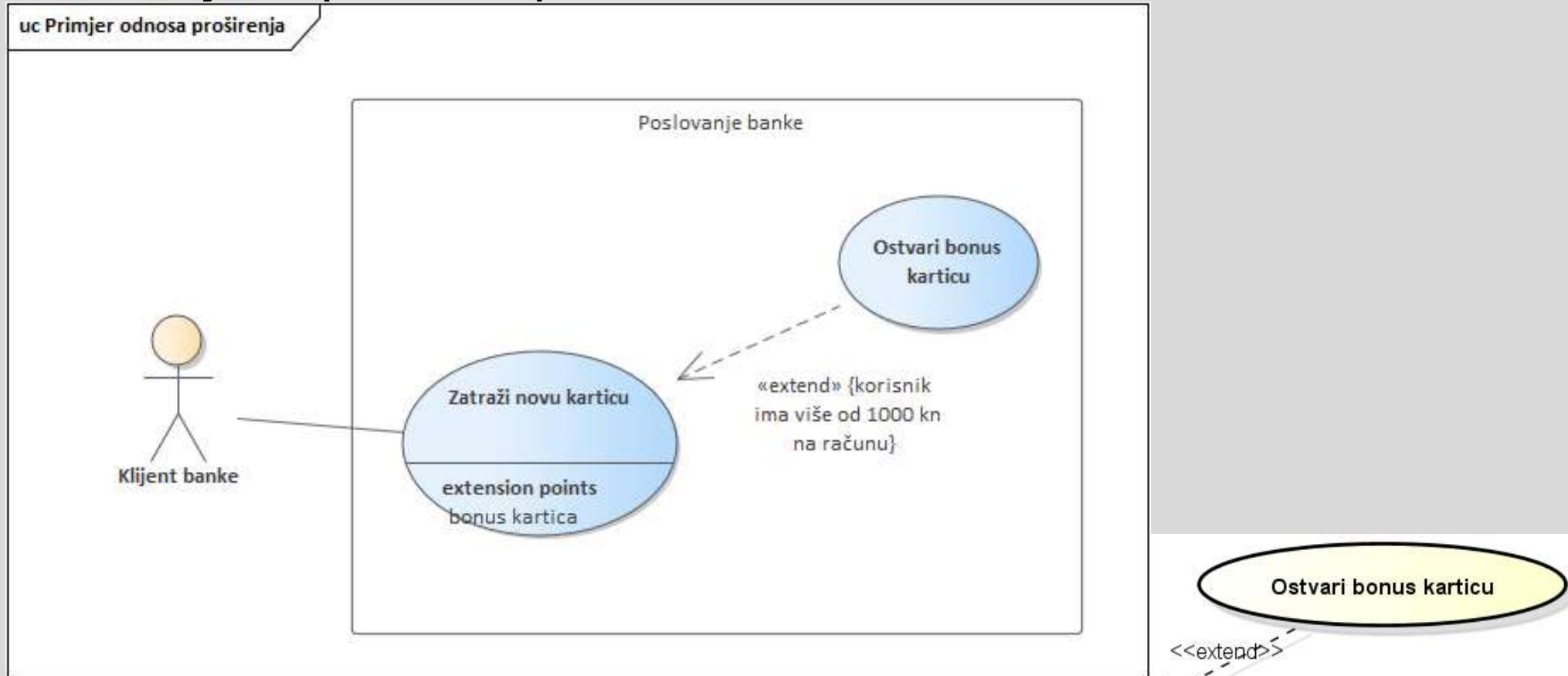


Odnos proširenja

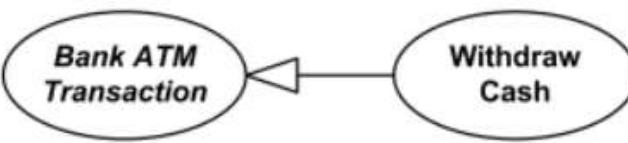
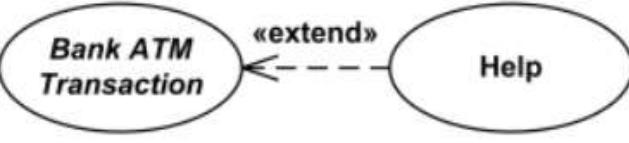
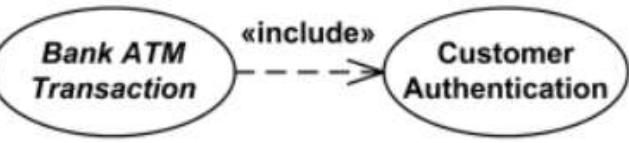
- Opcionalan odnos, specifično proširenje osnovnog obrasca uporabe s nekom funkcionalošću
- Veza od proširujućeg obrasca prema osnovnom obrascu (proširenom)
- Zadaje se uvjet proširenja na odnosu *extend* kao i točka proširenja (*extension point*) na osnovnom obrascu

Odnos proširenja

- Primjer ispravne uporabe:



Razlikovanje odnosa

Generalization	Extend	Include
		
Base use case could be abstract use case (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (abstract use case).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

<https://www.uml-diagrams.org/use-case.html#generalization>

<https://www.uml-diagrams.org/use-case-include.html>

https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/extend.html

Razlikovanje odnosa

Poopćenje	Proširenje	Obuhvaćanje
Osnovni obrazac uporabe može biti nepotpun (apstraktan) ili potpun (konkretan).	Osnovni obrazac uporabe je potpun (konkretan).	Osnovni obrazac uporabe je nepotpun (apstraktan).
Specijalni obrazac uporabe je nužan ako je osnovni obrazac uporabe apstraktan.	Proširujući obrazac uporabe je opcionalan, dodatan.	Uključeni obrazac uporabe je nužan.
Nema posebnog mesta odnosno točke specijalizacije.	Postoji točka proširenja.	Nema izričite točke uključenja, ali postoji mjesto proširenja.
Nema uvjeta specijalizacije.	Mogu postojati posebni uvjeti proširenja.	Ne izričitog uvjeta uključenja.

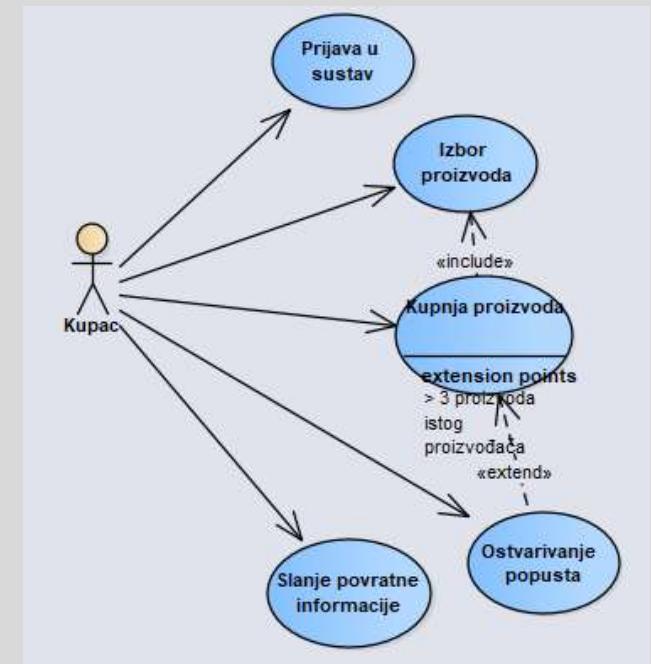
Dijagram obrazaca uporabe

- engl. ***use-case diagram***
- Treba razlikovati:
 - Pojedini dio funkcionalnosti sustava koji je naznačen nekim *obrascem uporabe* (engl. ***use-case***):
 - npr.: izdvojenu funkcionalnost kupnje proizvoda:



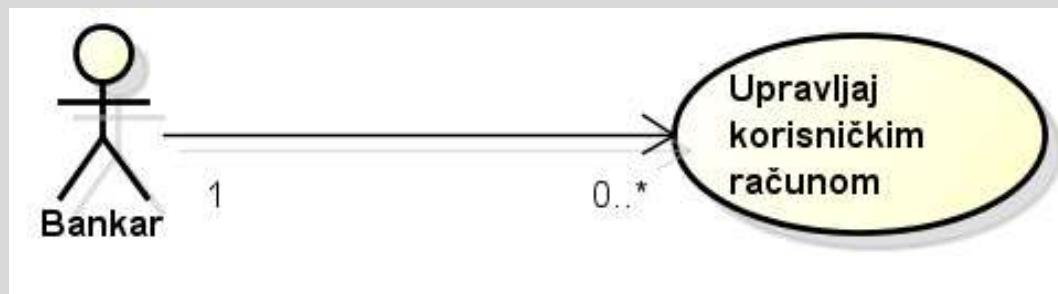
Dijagram obrazaca uporabe

- engl. *use-case diagram*
- Treba razlikovati:
 - Više povezanih funkcionalnosti koji su prikazani unutar *dijagrama obrazaca uporabe*:
 - npr. skup svih funkcionalnosti vezanih uz kupnju proizvoda:



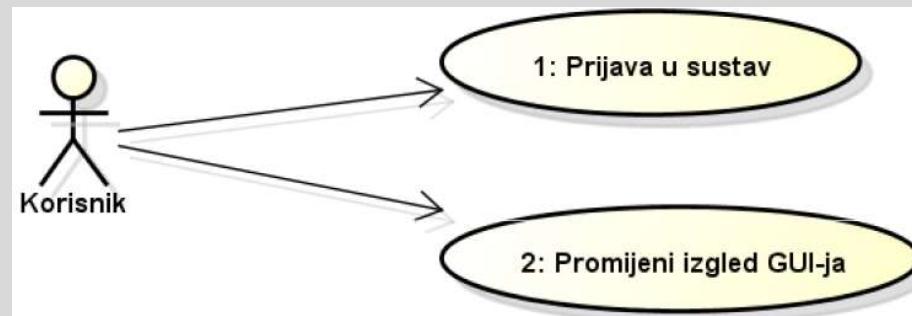
Višestrukost

- **Višestrukost (*multiplicity*) određuje broj aktora i obrazaca uporabe za koje je zadano pridruživanje**
- Postoje sljedeće mogućnosti (na bilo kojoj strani pridruživanja):
 - 1 točno jedan (ako ništa ne piše na pridruživanju onda se 1 podrazumijeva)
 - * više (nedefinirano koliko mnogo)
 - n bilo koji točno određen broj, npr. 0, 1, 3, 15
 - n1..n2 između jednog i drugog broja, npr. 1..3
 - n1.. * između jednog broja i neodređenog broja, npr. 0..*



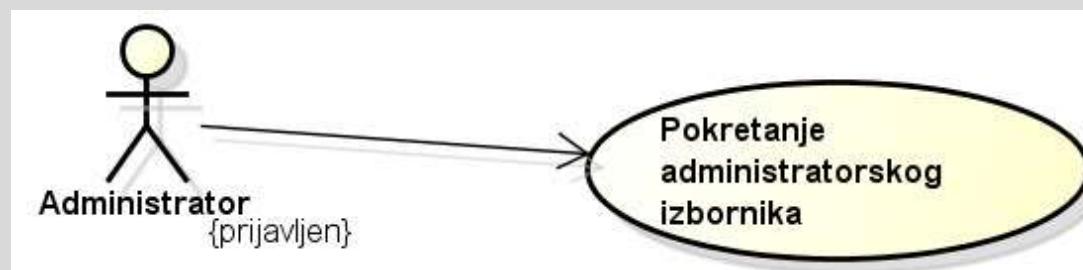
Dodatno

- U projektima se radi lakšeg snalaženja treba dodati **redni broj ispred naziva obrasca uporabe**. Taj broj upućuje na odgovarajući tekstualni opis scenarija:



Dodatno

- **Uvjeti (*constraint*) pod kojim se neki obrazac uporabe događa** u pravilu se ne crtaju na dijagramu već samo u tekstualnom opisu, ali ako se neki uvjet hoće posebno istaknuti, onda se on navodi u vitičastim zagradama:



Laboratorijske vježbe

UML OBRASCI UPORABE U DOKUMENTU ZAHTJEVA

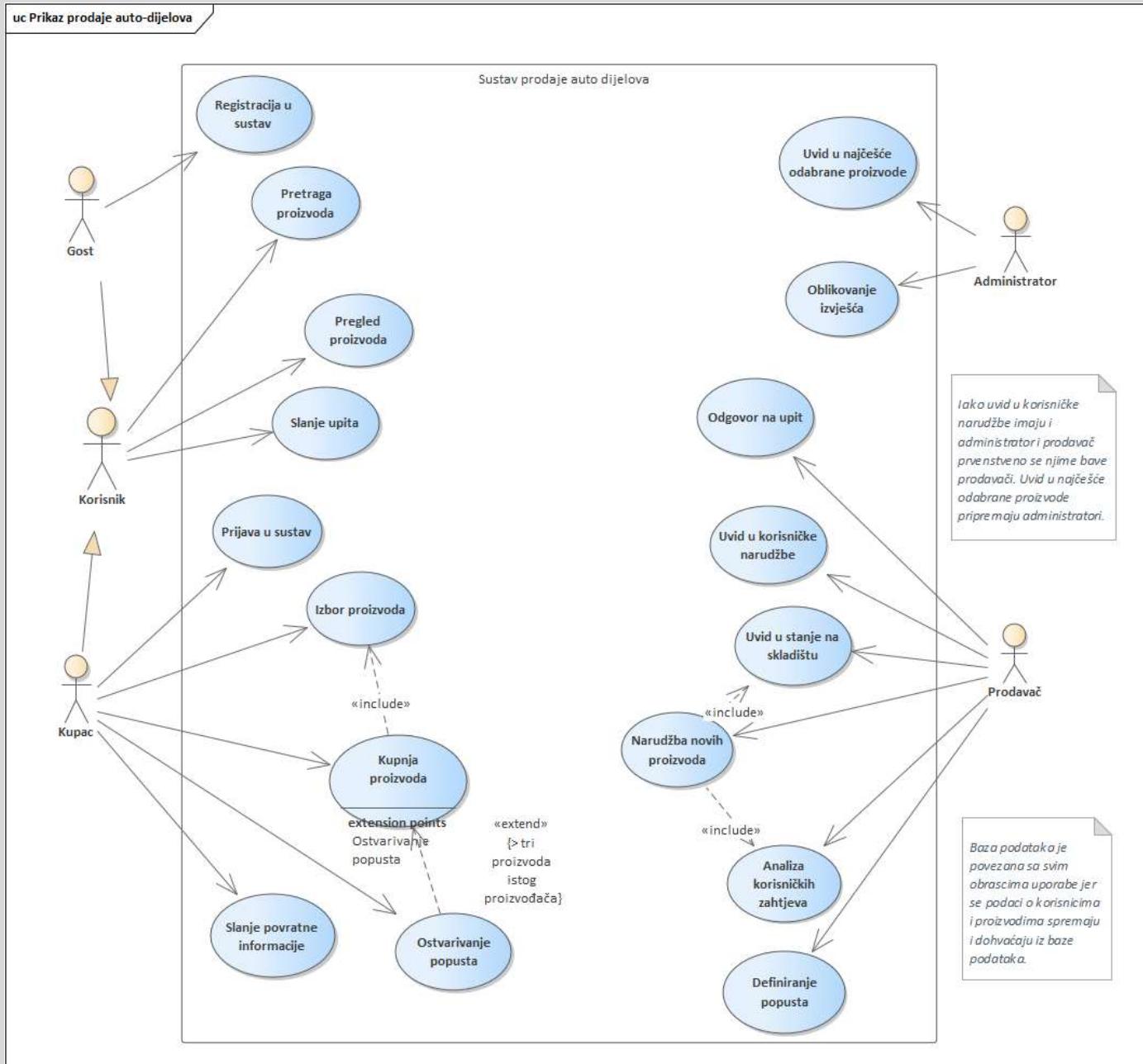
Dodatno (I)

- U projektima stavlja redni broj ispred naziva obrazca uporabe. Taj broj upućuje na odgovarajući tekstualni opis scenarija:
 - Pretraga proizvoda (UC1)
 - Pregled proizvoda (UC2)
 - Slanje upita (UC3)
 - Registracija u sustav (UC4)
 - Prijava u sustav (UC5)
 - Izbor proizvoda (UC6)
 - Kupnja proizvoda (UC7)
 - ...
- Jednom dodijeljeni redni broj jednoznačno određuje obrazac uporabe tijekom cijelog procesa izrade projekta.

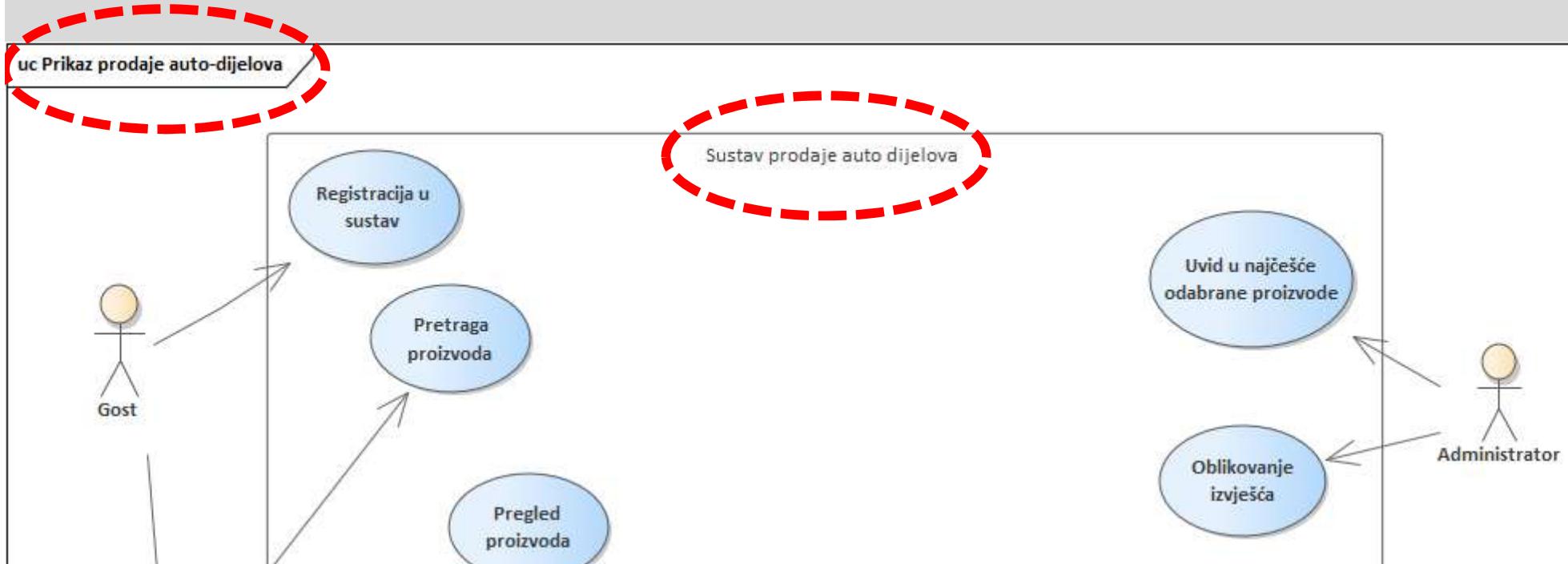
Dodatno (II)

- Svi UML dijagrami trebaju imati odgovarajući naziv
 - definirati u sklopu funkcionalnosti alata za modeliranje
- U dokumentaciji – u potpisima slika uvrštenih dijagrama.

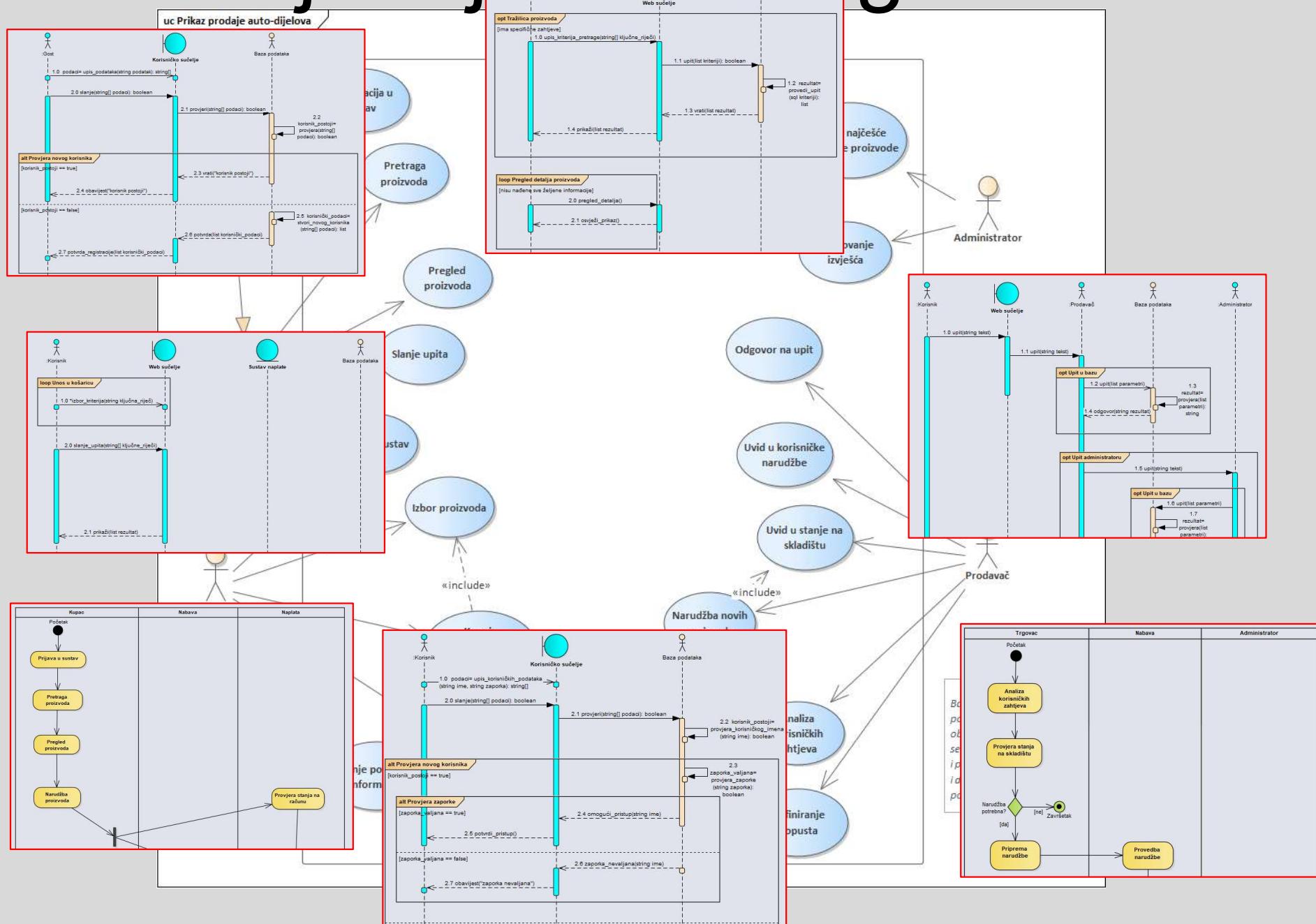
Primjer cjelokupnog sustava



Naziv dijagrama



Primjer cjeleokupnog sustava



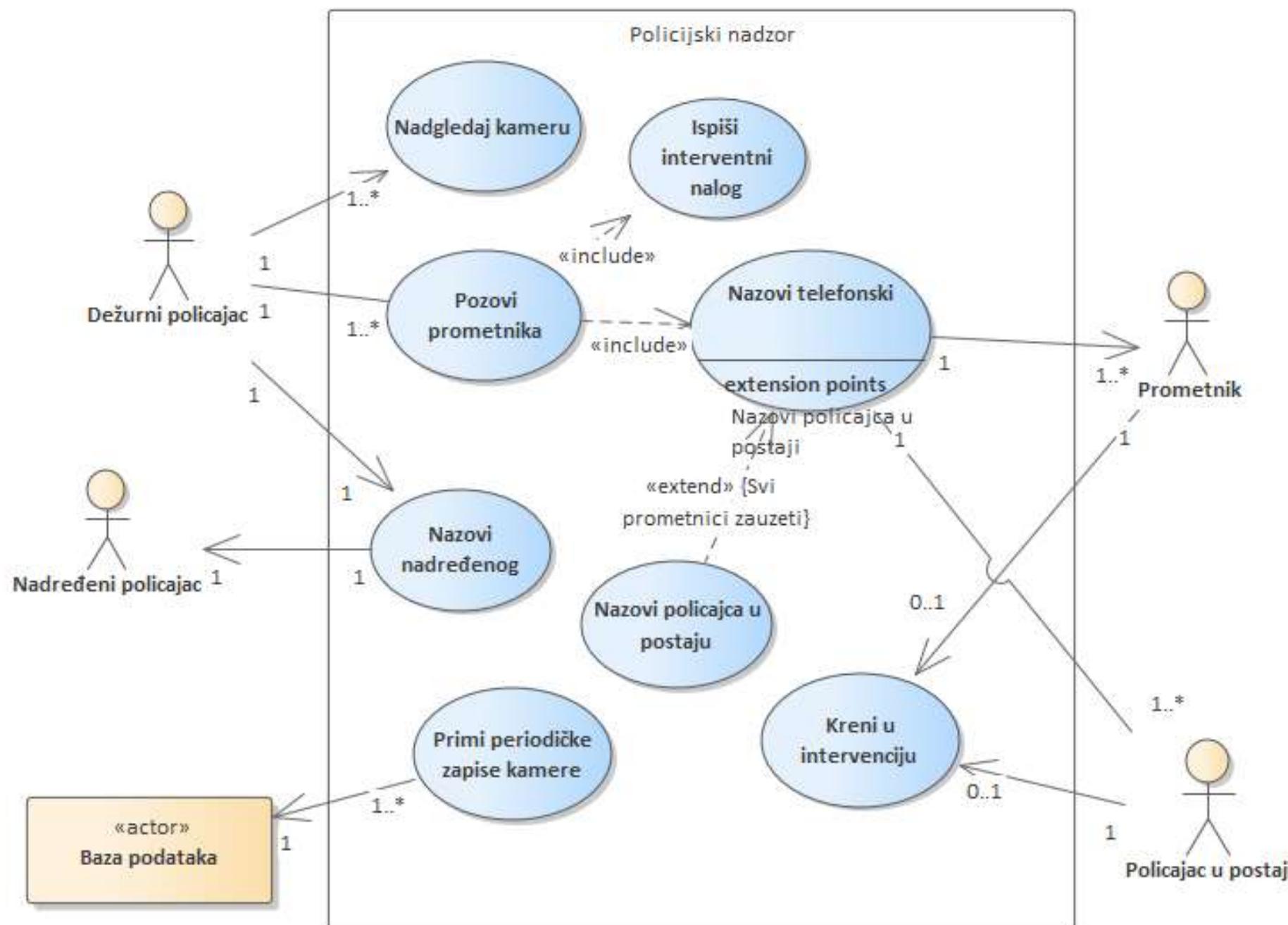
Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

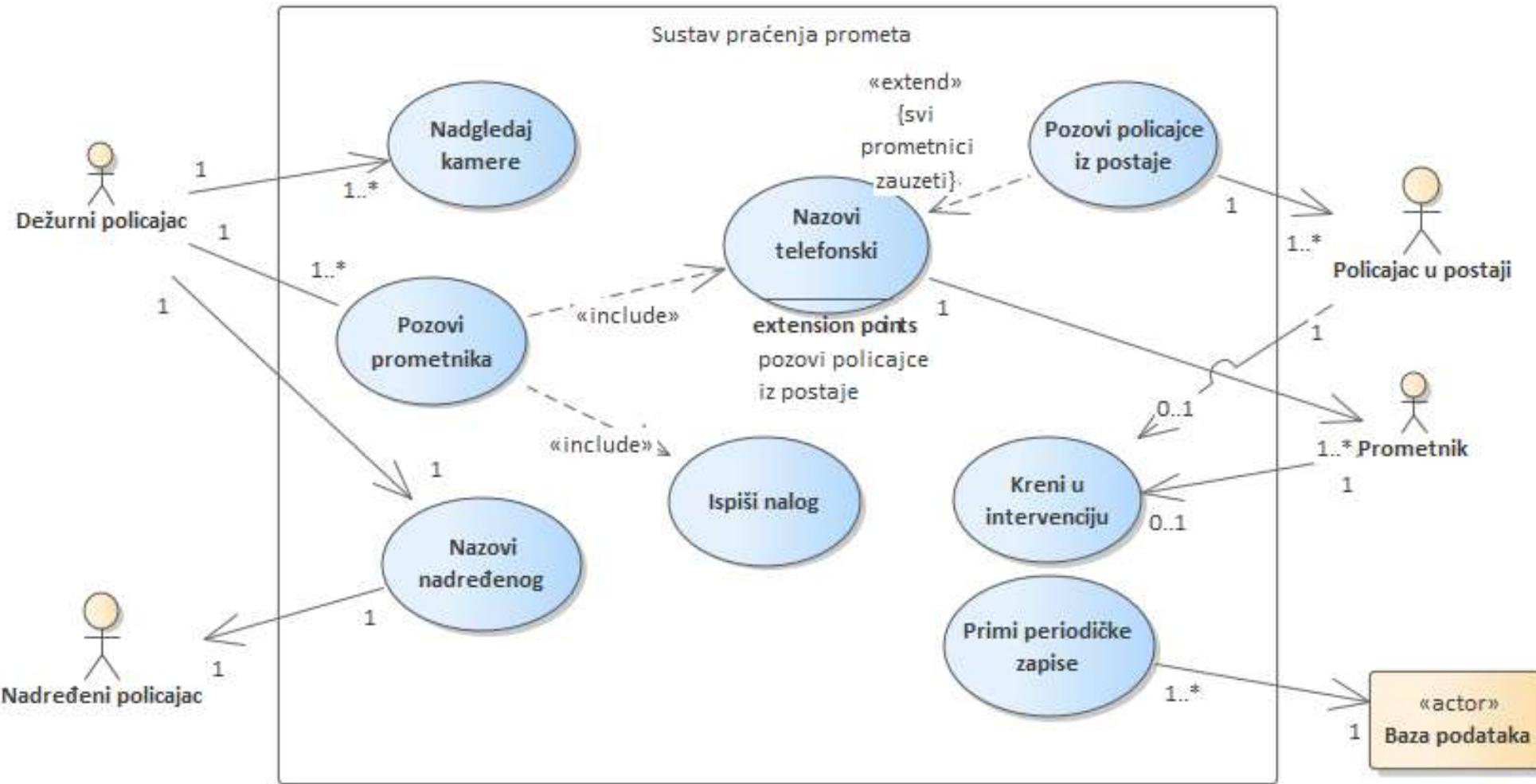
PRIMJERI MODELIRANJA

Dijagram obrazaca uporabe – Primjer 1

Sustav praćenja prometa koriste dežurni policajac i prometnici. Jedan dežurni policajac nadgleda više kamera u nadzornom centru. Osim nadgledanja, dežurni policajac može pozvati prometnike na teren ili nazvati nadređenog policajca. Pozivanje prometnika na teren uključuje telefonski poziv i ispis interventnog naloga. U specijalnom slučaju, kad su svi prometnici zauzeti prilikom telefonskog poziva, dežurni policajac poziva umjesto njih kolege policajce u postaji. Prometnik ili policajac u postaji mogu primiti poziv od dežurnog policajca (jedan ili više njih primaju poziv od jednog dežurnog policajca) i po potrebi krenuti u neku intervenciju. Sustav praćenja prometa sadrži i bazu podataka koja prima periodički zapise s jedne ili više nadzornih kamera.



Uč Nadgledanje prometa

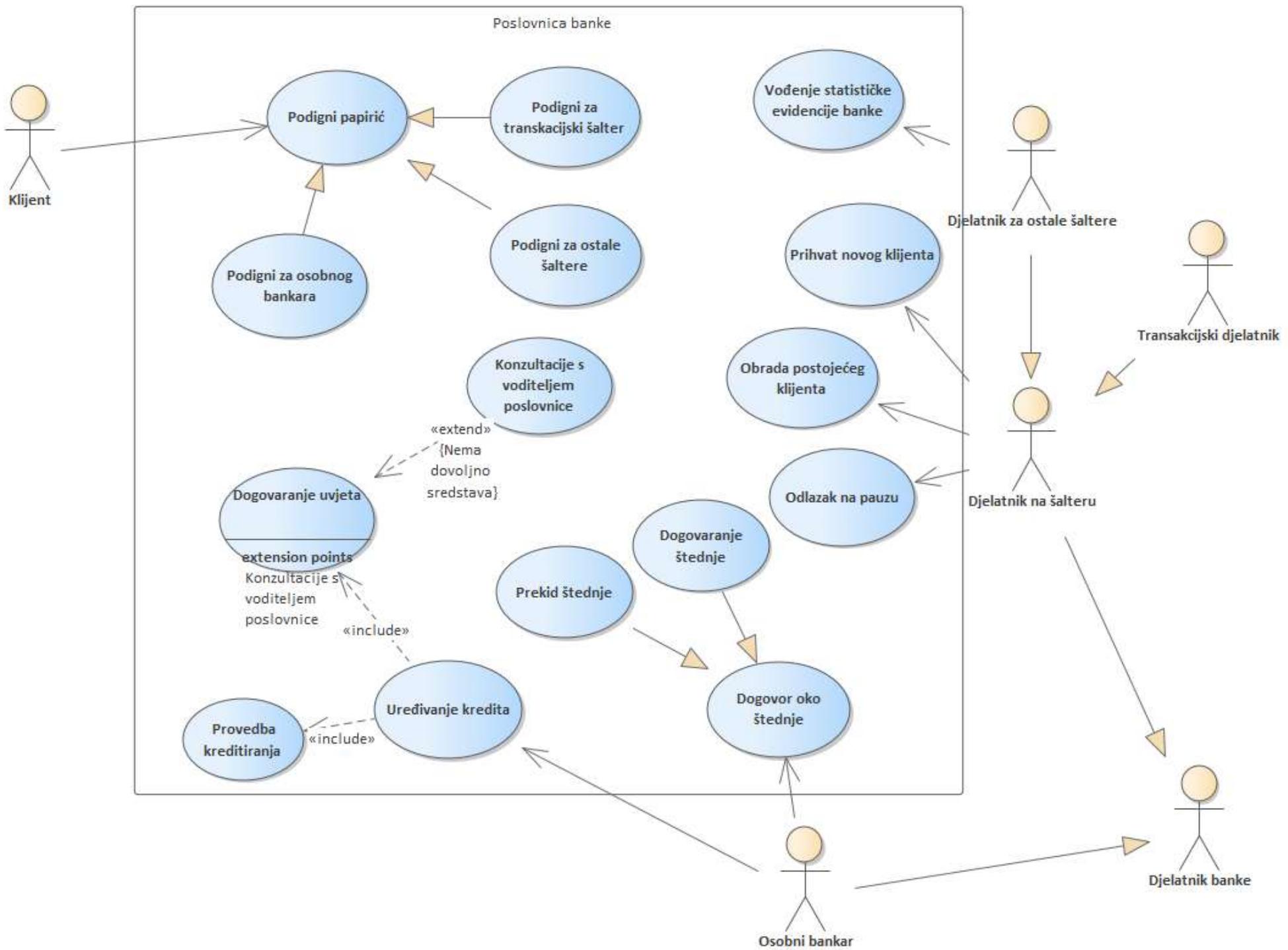


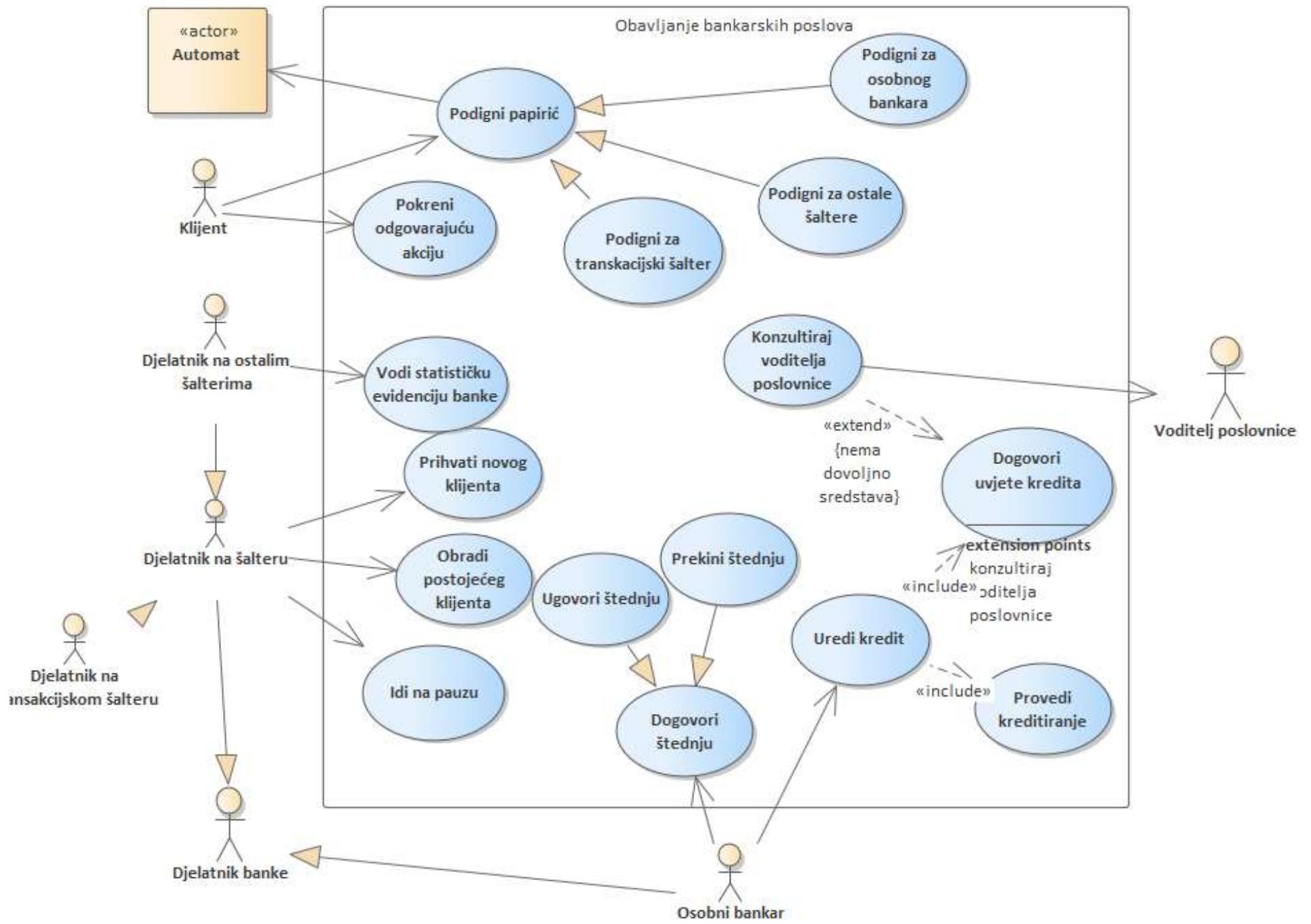
Komentari na rješenje Primjera 1

- Nadređenog policajca je bolje navesti kao aktora, iako on nema neku konkretnu akciju u sustavu, budući da ga se predloženi sustav tiče.
- Općenito, bolje da je dijagram bogatiji tekstom nego da je više oskudan, zato što scenarije moraju dobro razumjeti svi!

Dijagram obrazaca uporabe – Primjer 2

Klijent ulaskom u banku ima mogućnost podizanja papirića s rednim brojem na automatu. Prilikom odabira papirića, klijent može odabrati papirić za transakcijski šalter, ostale šaltere ili za osobnog bankara. Postoje dvije vrste djelatnika na šalterima. Jedni rade na transakcijskom šalteru, a drugi na ostalim šalterima. Obje vrste djelatnika imaju mogućnost prihvata novog klijenta, obrade postojećeg klijenta ili odlaska na pauzu. Djelatnik na ostalim šalterima dodatno može voditi statističku evidenciju banke. Osobni bankar je posebna vrsta djelatnika banke, koji ima mogućnosti uređivanja kredita i dogovora oko štednje s klijentom. Dogovor oko štednje može značiti prekid štednje i ugovaranje štednje. Uređivanje kredita uključuje dogovaranje uvjeta kreditiranja i provedbu kreditiranja. U specijalnom slučaju, kad klijent želi dogоворити uvjete kreditiranja, a nema dovoljno sredstava na računu za pokrivanje prve rate kredita, osobni bankar se može konzultirati s voditeljem poslovnice. Klijent može izvršiti sve akcije koje mu omogućavaju djelatnici banke nakon što je podignuo papirić.





Komentari na rješenje 2. primjera

- Voditelj poslovnice može se uvesti kao dodatni aktor.
- Kao poseban aktor može se staviti i automat za izdavanje papirića
- Može se uvesti dodatni obrazac uporabe “Pokreni odgovarajuću akciju” kojim se može naznačiti da klijent može pokrenuti bilo koju akciju koju mu omogućavaju djelatnici banke nakon uzimanja papirića:
 - taj obrazac uporabe se može povezati s obrascima: “Uređivanje kredita”, “Dogovor oko štednje”, “Prihvatanje novog klijenta” i “Obrada postojećeg klijenta” uz pomoć odnosa poopćenja/specijalizacije.

REFERENCE I LITERATURA

- A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014., dostupno u Skriptarnici i knjižnici Fakulteta elektrotehnike i računarstva, te Nacionalnoj i sveučilišnoj knjižnici u Zagrebu
- <https://www.uml-diagrams.org>

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI SLIJEDA

Dijagrami slijeda

- Dijagram slijeda (engl. *sequence diagram*) služi za prikaz slijeda događaja među **objektima pojedinih razreda** kao i među **aktorima**.
- **Usmjeren na vremenski prikaz**
- Ponašanje sustava modelira se dinamičkim interakcijama između **objekata pojedinih razreda** i **između aktora preuzetih iz dijagrama obrazaca uporabe**.
- Tijek interakcije na dijagramima je od gore lijevo prema dolje desno.

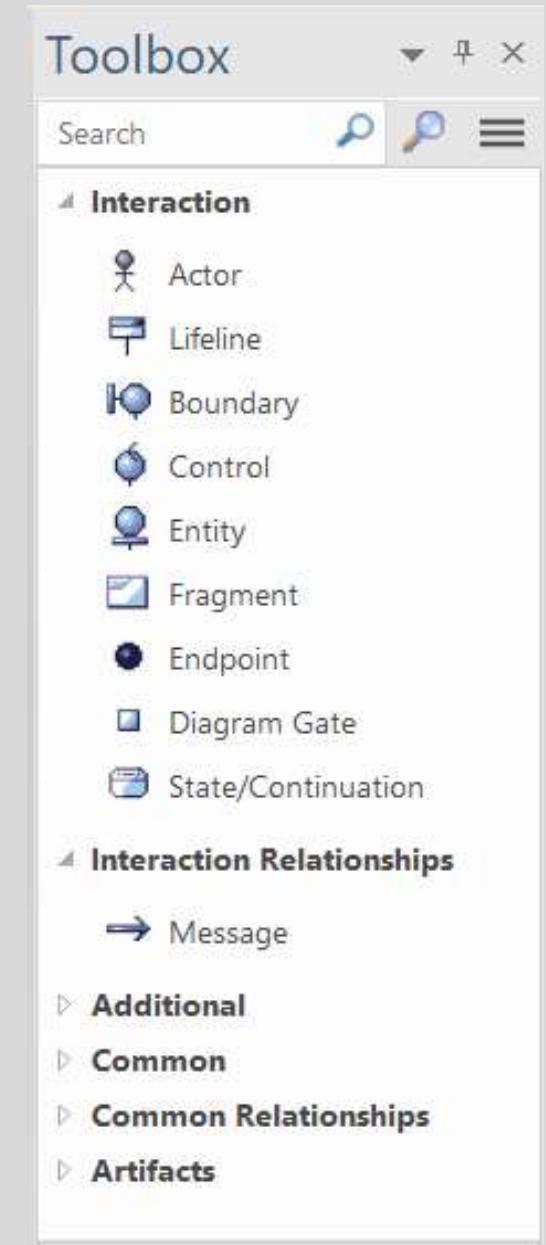
Dijagrami slijeda

- Elementi:

- Aktor
 - Životna crta
 - Granica
 - Upravljački element
 - Entitet
 - Fragment
 - Završna točka
 - Vrata
 - Stanje/Kontinuitet

- Odnosi:

- Poruka

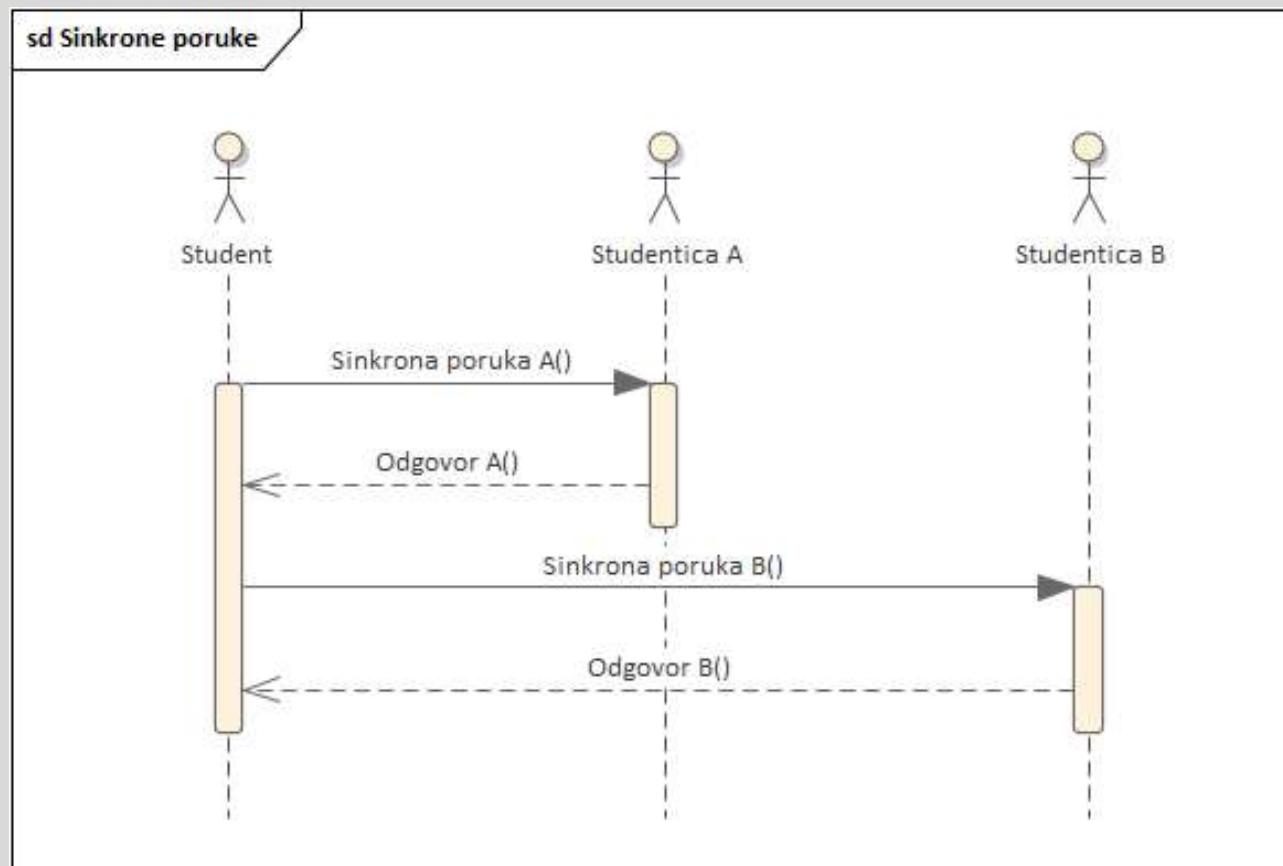


Dijagram slijeda - primjeri

- Iz Enterprise Architect dokumentacije:
 - osnovni primjer:
 - https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/sequencediagram.html
 - korištenje bloka *CombinedFragment*:
 - https://sparxsystems.com/resources/gallery/diagrams/software/sw-sequence_diagram_with_fragment.html
 - referenciranje na druge dijagrame:
 - https://sparxsystems.com/resources/gallery/diagrams/software/sw-sequence_diagram_with_references.html

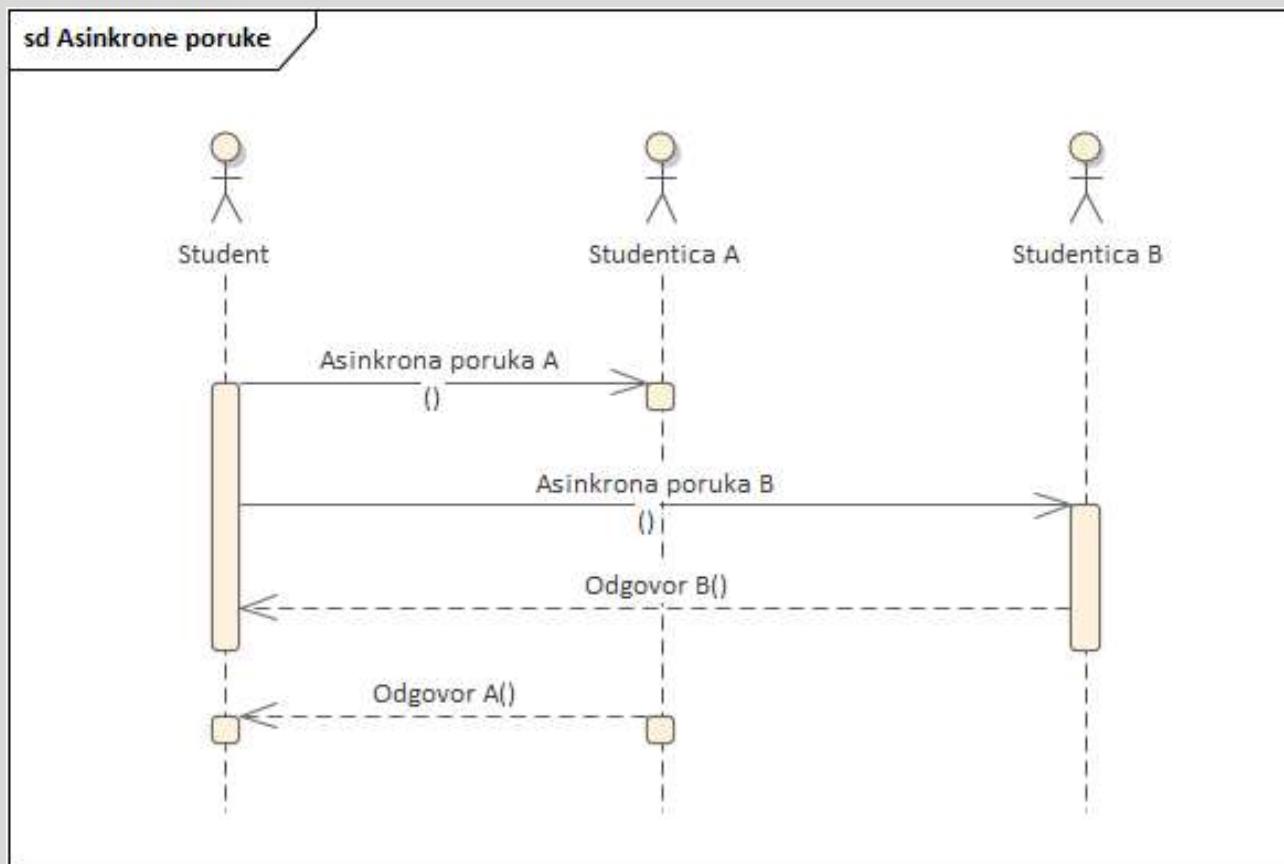
Poruke između objekata: sinkrona poruka

- **Strelica s punim vrhom – sinkrona poruka.**
- Objekt koji šalje poruku čeka odgovor i tek kad ga dobije nastavlja s radom.



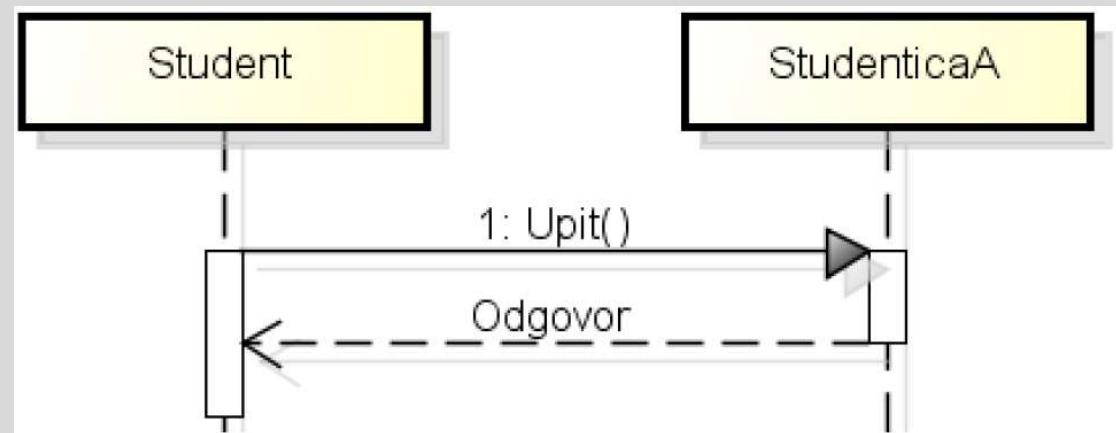
Poruke između objekata: asinkrona poruka

- **Strelica s običnim vrhom – asinkrona poruka**
- Objekt koji ju šalje nastavlja s radom i ne čeka na odgovor, koji može, ali i ne mora doći.



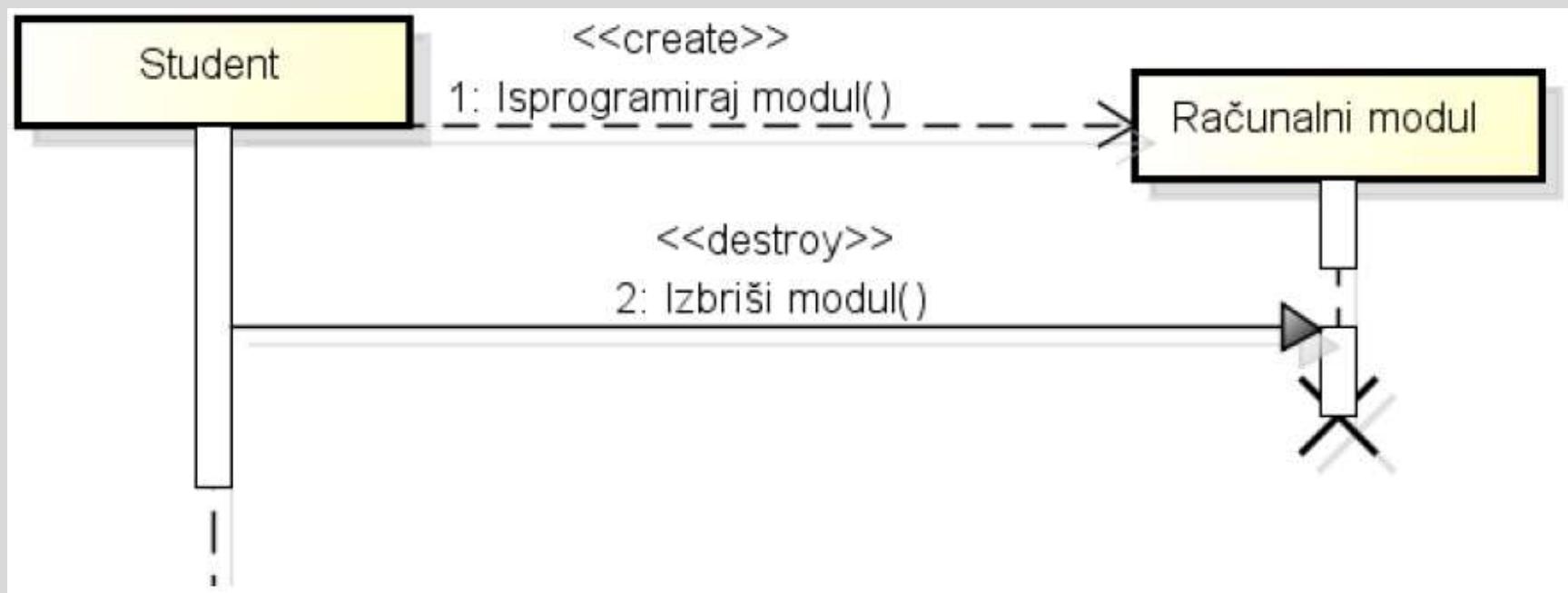
Poruke između objekata: povratna poruka

- **Isprekidana strelica – povratna poruka:**
- Povratna poruka ne mora se crtati kod sinkronih poruka ako se ne vraća rezultat (kao kod *void* funkcija), ali ju je svejedno bolje nacrtati.
- Kod asinkronih poruka, povratna poruka je implicitna odmah nakon poziva, ne crta se!
- Ako odgovor na asinkronu poruku postoji, on se može crtati kao nova poruka (sinkrona ili asinkrona) ili kao povratna poruka.



Poruke među objektima: stvaranje i uništavanje objekata

- **Stvaranje objekata**
 - Jedan objekt može u tijeku izvođenja stvoriti drugi objekt
- **Uništavanje objekata**
 - Objekt završi s izvođenjem ili vanjski objekt ga uništi



Poruke među objektima

- Opći izgled poruke:

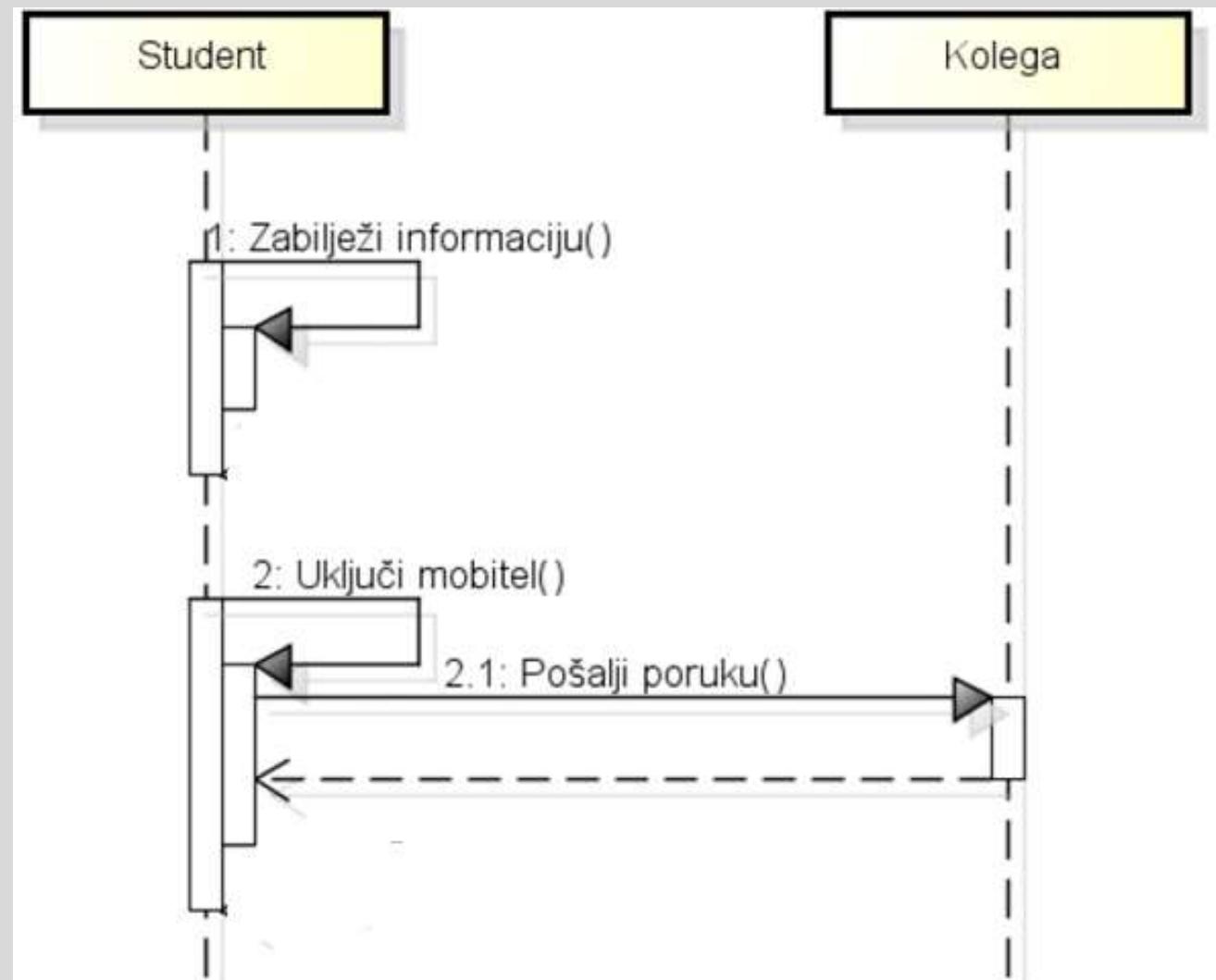


- Broj ispred poruke označava njezin redni broj u dijagramu, nije ga nužno navoditi na sekv. dijagramima
- "[uvjet]" - navodi se uvjet pod kojim se poruka šalje
- "tip" je povratni tip poruke, npr. int, double i slično
- "argumenti" je popis argumenata, npr. (String tekst, int broj)
- Može se izostaviti sve ostalo, ali treba se dati **naziv** poruci
- Osim toga, može se dodati znak * (zvjezdica) prije [uvjet], npr. *[a > 5] pošalji_sadržaj(String sadržaj, int a) : void
- * označava petlju – poruka se šalje onoliko puta koliko je zadano uvjetom [uvjet]

Poruke među objektima

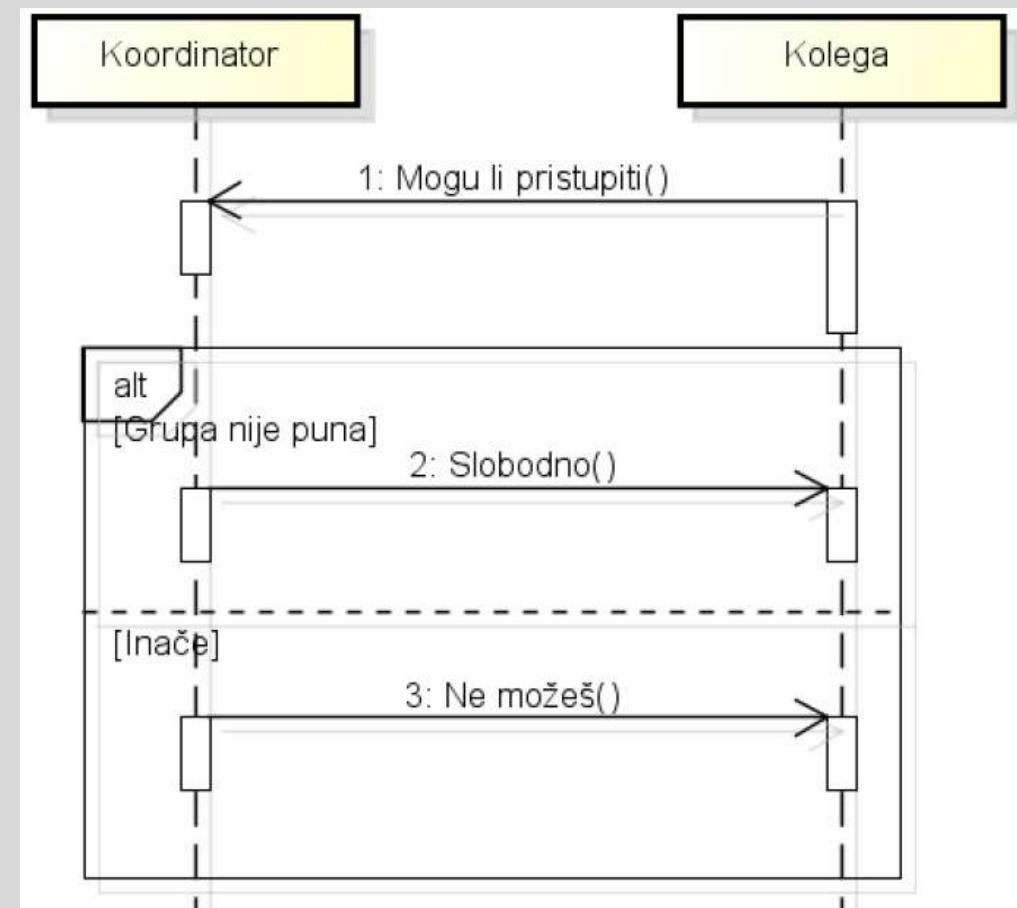
- **Poruka samom sebi (engl. *self-call*).**

- Objekt poziva svoj interni postupak ili proceduru, interni postupak može pozivati neki drugi postupak (ili slanje poruke drugim aktorima/objektima).
- U alatima za UML modeliranje ostvaruje se pomoću sinkrone poruke u (samo se treba usmjeriti nazad u samog sebe).



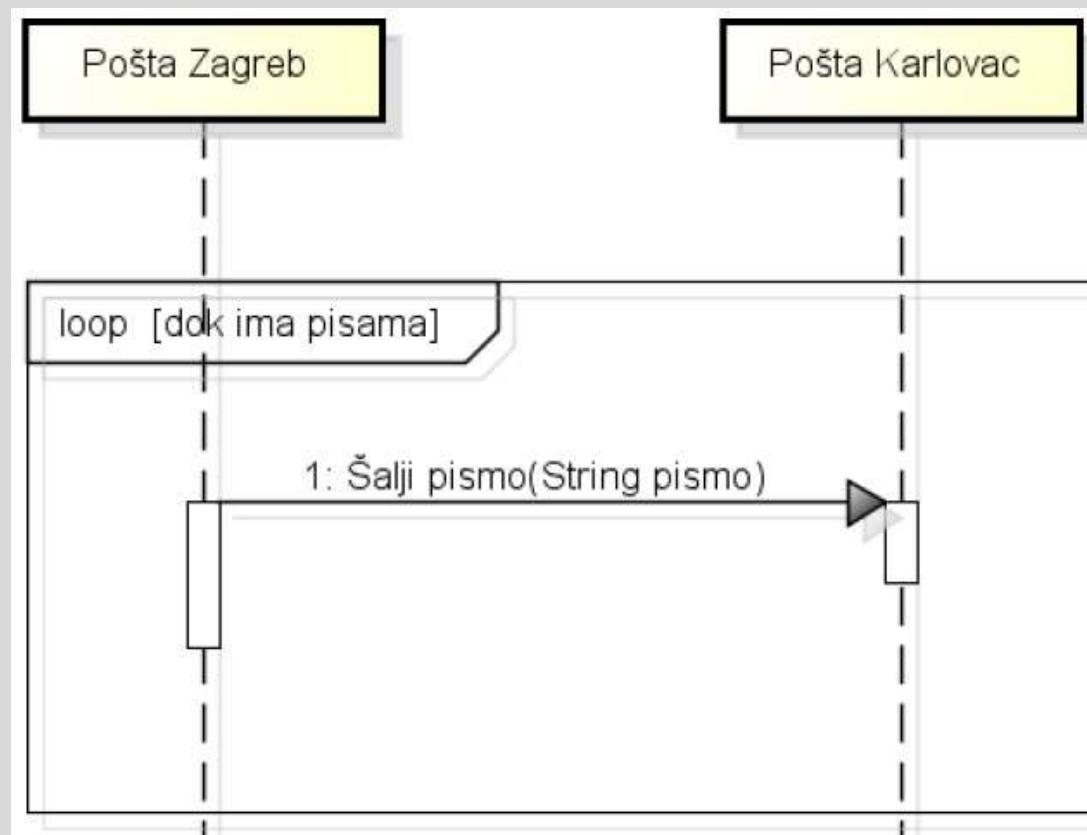
Ostale značajnije poruke (1)

- **Uvjetno grananje** podržano je pravokutnikom “Combined fragment”, navodi se operator alt (slično switchu) ili opt (slično jednom if).
- Pod karticom “Operand” navode se **sve opcije grananja, tj. uvjeti (“Guard”)** koji su pišu unutar uglatih zagrada.



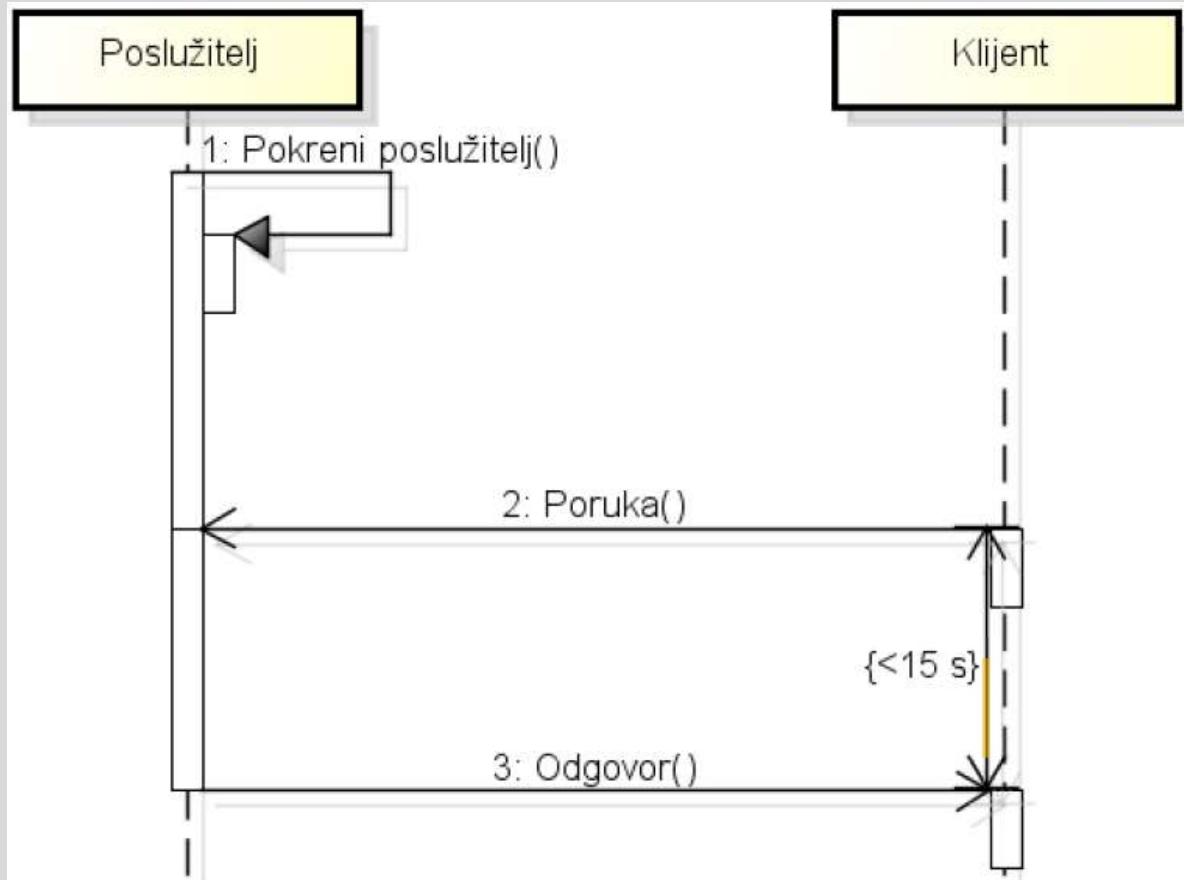
Ostale značajnije poruke (2)

- Normirana notacija **petlje** od verzije UML-a 2.0+ je pravokutnik s oznakom “loop” i uvjetom na izvođenje.
- Sve poruke unutar pravokutnika izvode se sve dok je zadovoljen uvjet petlje.



Ostale značajnije poruke (3)

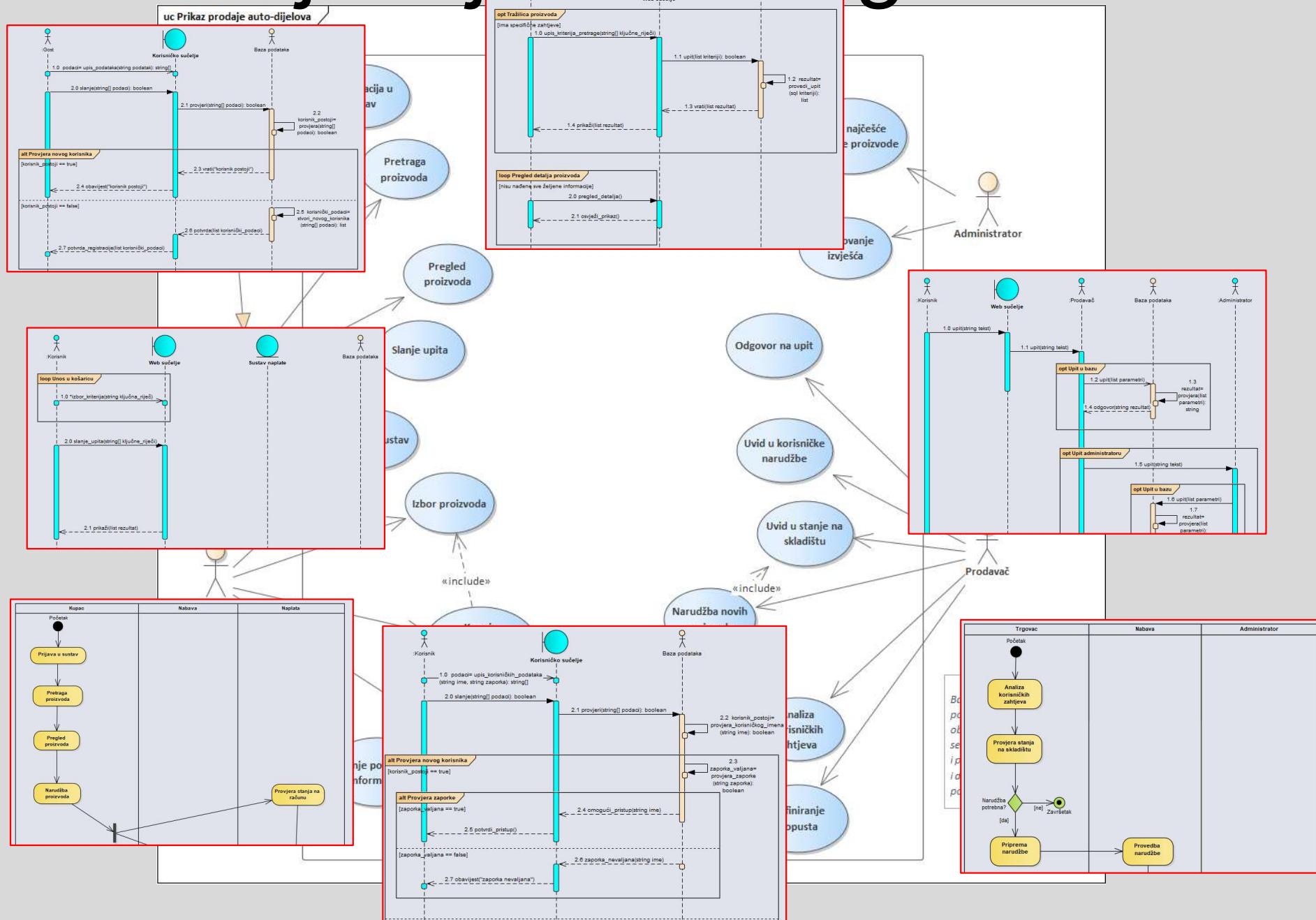
- **Čekanje određeno vrijeme na primitak poruke**
- Primjer, klijent čeka do 15 s na odgovor poslužitelja (i dok čeka nešto drugo obavlja). {<15 s}



Laboratorijske vježbe

UML DIJAGRAMI SLIJEDA U DOKUMENTU ZAHTJEVA

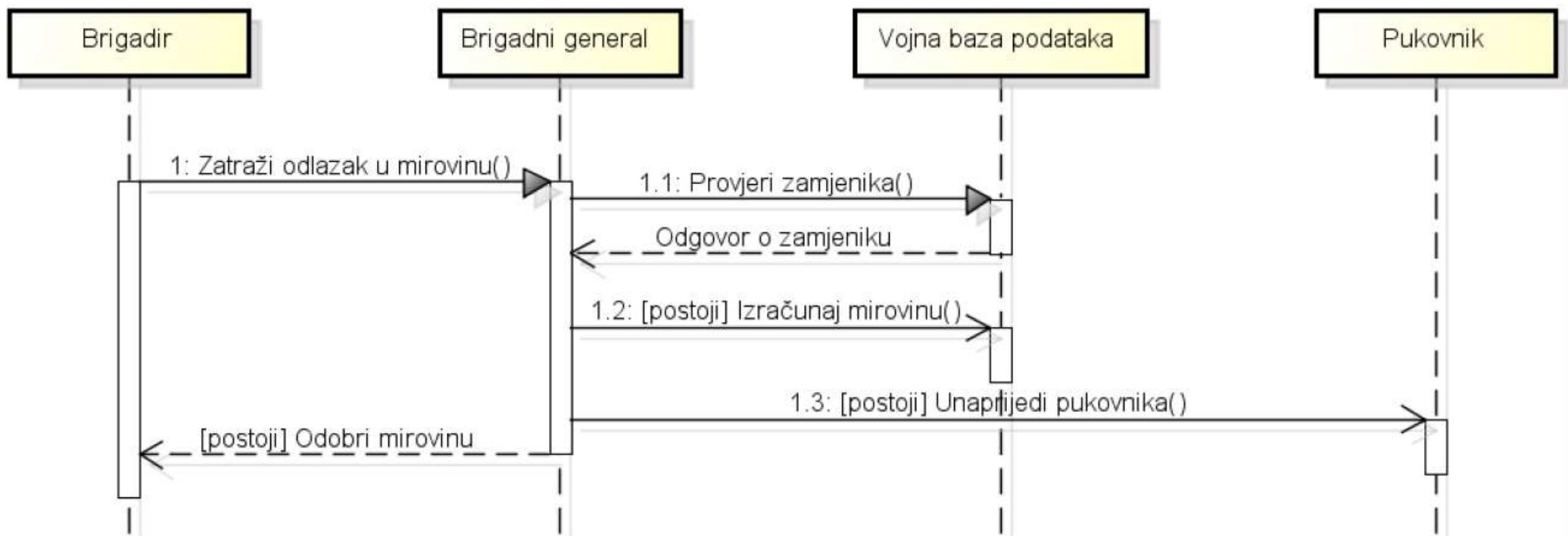
Primjer cjeleokupnog sustava



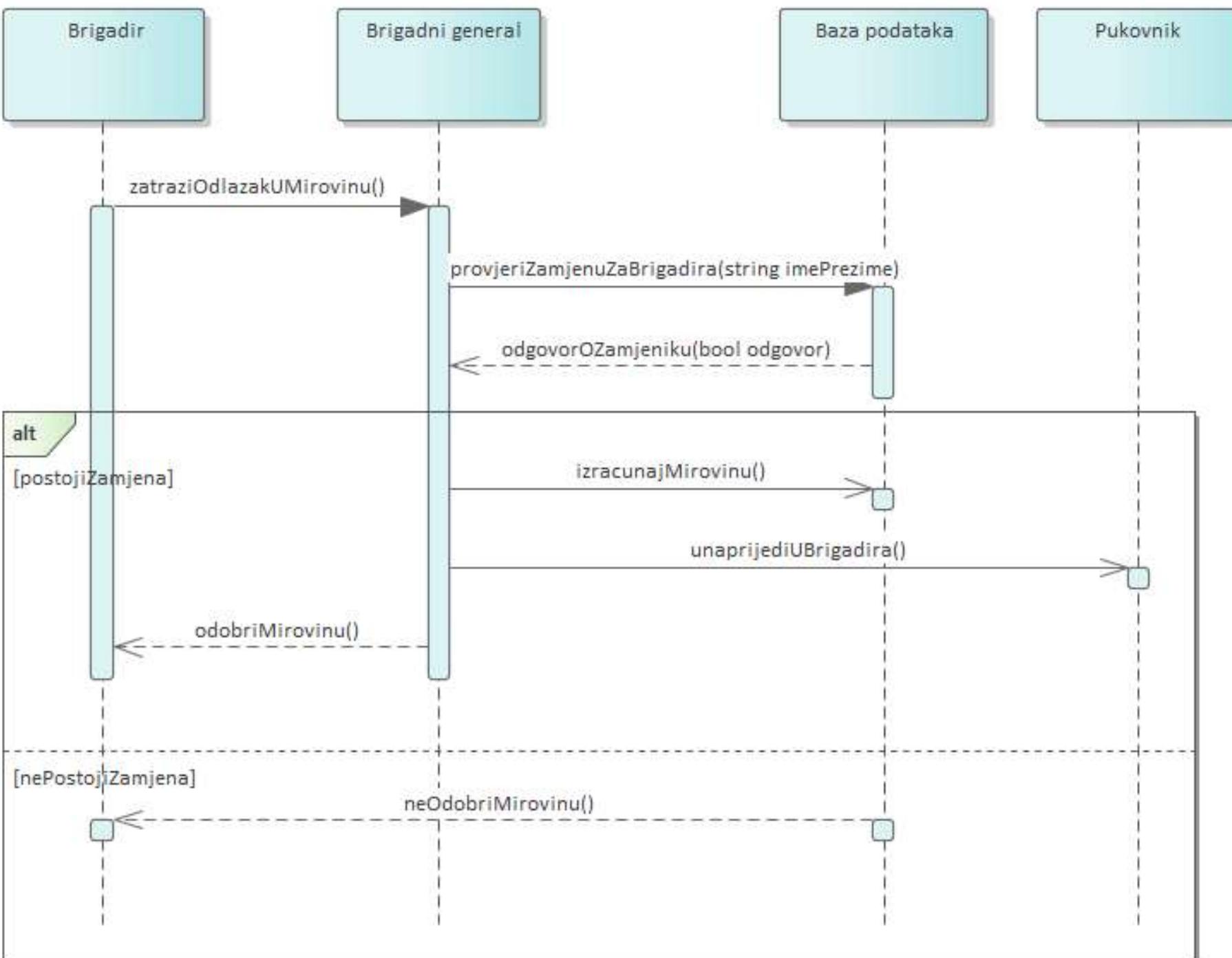
PRIMJERI MODELIRANJA

Dijagram slijeda – primjer 1

Modelirati dijagramom slijeda odlazak brigadira u mirovinu. Nakon zahtjeva, brigadni general mu dozvoljava odlazak u mirovinu ako ima prikladnu zamjenu (provjerava u vojnoj bazi podataka). Brigadiru se u tom slučaju izračunava iznos mirovine na temelju podataka u vojnoj bazi, a istovremeno general unapređuje nekog pukovnika u brigadira. Ako postoji zamjena, brigadirov zahtjev se prihvata.



sd Odlazak brigadira u mirovinu



Dijagram slijeda – primjer 2

Poslužitelj prima poruke od dva klijenta: klijenta A i klijenta B. Na početku, na poslužitelju se pokreće interni postupak kojim poslužitelj počinje osluškivati dolazak poruka. Taj interni postupak kao argument prima port (int) na kojem treba osluškivati. Nakon toga, klijent A šalje jednu ili više poruka poslužitelju. Svaka poruka ima svoje zaglavlje (string), tekst (string) i određenu duljinu (int). Za svaku poruku klijenta A, poslužitelj pokreće postupak obrade koji rezultira slanjem povratne poruke (rezultata) klijentu A. Klijent A čeka na odgovor. Dok obrađuje poruke klijenta A, poslužitelj ne može primiti poruku klijenta B. Klijent B šalje samo jednu poruku s jednim argumentom, izvještajem (string), i to pod uvjetom da klijent B ima pripremljeni izvještaj. Ako izvještaj nije pripremljen, poruka se ne šalje. Ako primi poruku od klijenta B, poslužitelj nastavlja dalje sa svojim radom i čeka dolazak novih poruka od klijenata te ne vraća odgovor klijentu B odmah. Klijent B također nastavlja s radom ne čekajući odgovor. U slučaju da je izvještaj pozitivno ocijenjen, poslužitelj će poslati povratnu poruku pod nazivom "izvještaj ok", a inače se ništa ne šalje. Poslužitelj se prekida pozivom internog postupka "prekini()" nakon čega poslužitelj više ne čeka na dolazak poruka od klijenata.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI AKTIVNOSTI

Dijagram aktivnosti

- Modeliranje poslovnog procesa
- Prikaz toka upravljanja ili podataka
- Prikaz niza vezanih aktivnosti
- U odnosu na dijagram obrazaca uporabe:
 - niz (povezanih) obrazaca uporabe
 - detalji obrasca uporabe

Elementi dijagrama aktivnosti (1)

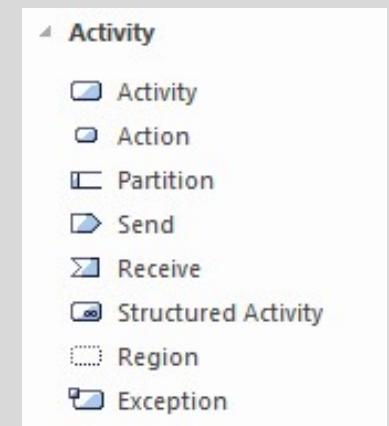
- Vrste elemenata:
 - Čvor (engl. *node*)
 - Prijelazi (engl. *activity edges*)
 - Particije (engl. *swimlane*)

Elementi dijagrama aktivnosti (2)

- Vrste elemenata:
 - Čvor (engl. *node*)
 - Čvor akcije (engl. *action node*)
 - Upravljački čvor (engl. *control node*)
 - Objektni čvor (engl. *object node*)
 - Prijelazi (engl. *activity edges*)
 - Upravljački tok (engl. *control flow*)
 - Tok objekta (engl. *object flow*)
 - Particije (engl. *swimlane*)
 - → prikaz podjele prema sudionicima („plivačke staze“)

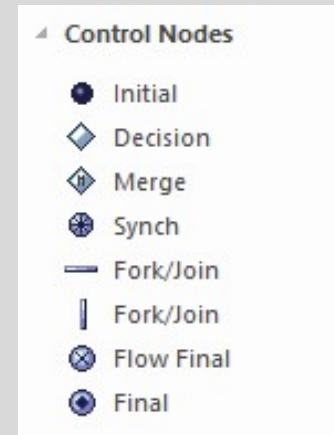
Čvorovi akcije

- Vrste:
 - Pozivanje/izvođenje operacija (engl. *call action*)
 - Slanje signala (engl. *send signal*)
 - Prijhvatanje događaja (engl. *accept event*)
 - Vremenski događaji (engl. *time event*)



Upravljački čvorovi

- Vrste:
 - Početni čvor (engl. *initial node*)
 - Završni čvor (engl. *final node*)
 - Završetak toka (engl. *flow final node*)
 - Čvor odluke/uvjetno grananje (engl. *decision node*)
 - Spajanje (engl. *merge*)
 - Grananje/račvanje (engl. *fork*)
 - Spajanje/skupljanje/sinkronizacija (engl. *join*)



Elementi dijagrama aktivnosti: početni i konačni čvor

- **Početni čvor** označen je crnim krugom, a **konačni čvor** označen je zaokruženim crnim krugom.

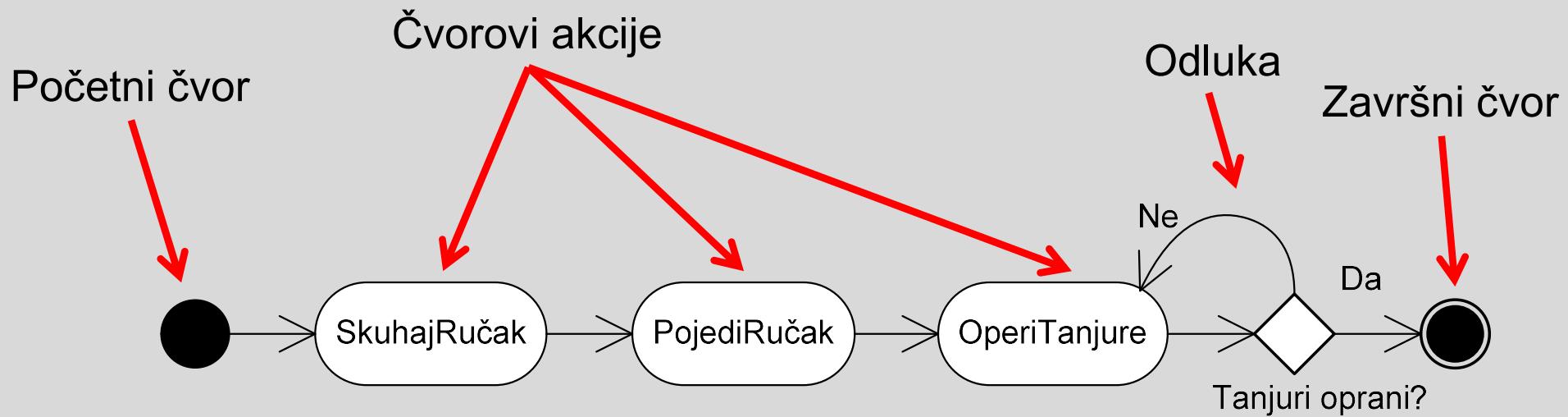


- Dijagram aktivnosti može imati više početnih i konačnih čvorova.

Elementi dijagrama aktivnosti:

aktivnosti

- Akcija se označena zaobljenim pravokutnikom.
 - Unutar nekog čvora akcije se odvija samo ta akcija – atomarna operacija



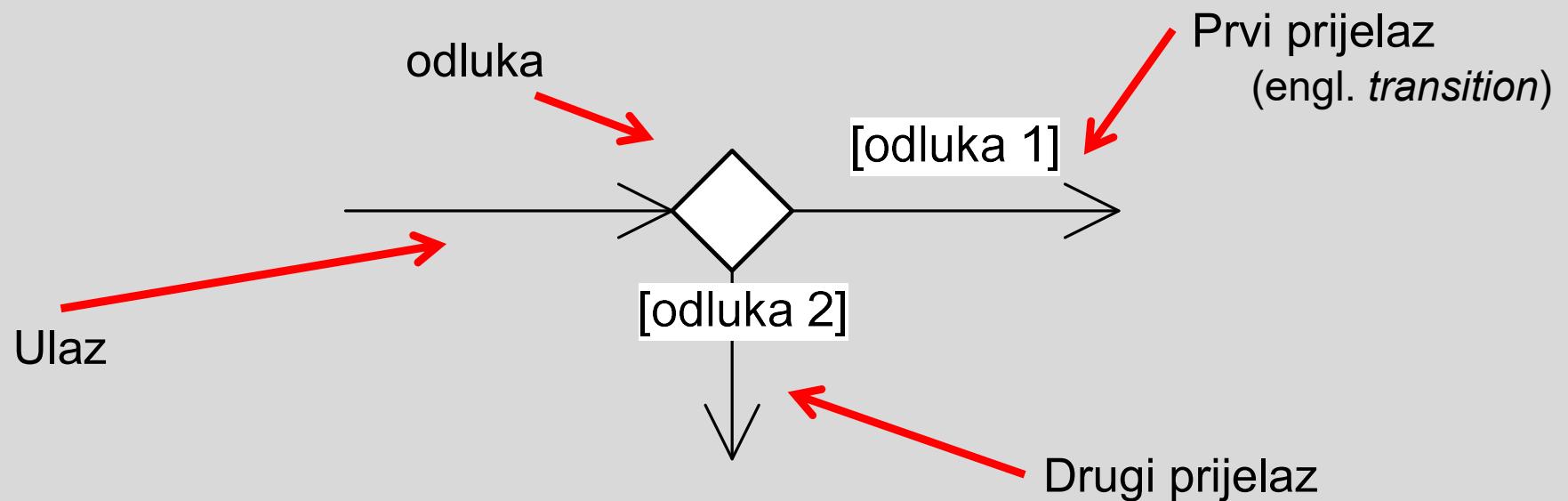
Elementi dijagrama aktivnosti: prijelaz između aktivnosti

- **Prijelaz između akcija označen je jednostruko usmjerenom strelicom** i može sadržavati:
 1. Može imati **naziv** (engl. *edge name*).
 2. **Uvjeti prijelaza**, ograničenja ili izraz koji treba zadovoljiti da bi se prijelaz ostvario (engl. *guard*, *guard expression*). Prikazuje se unutar uglatih zagrada.



Elementi dijagrama aktivnosti: odluka

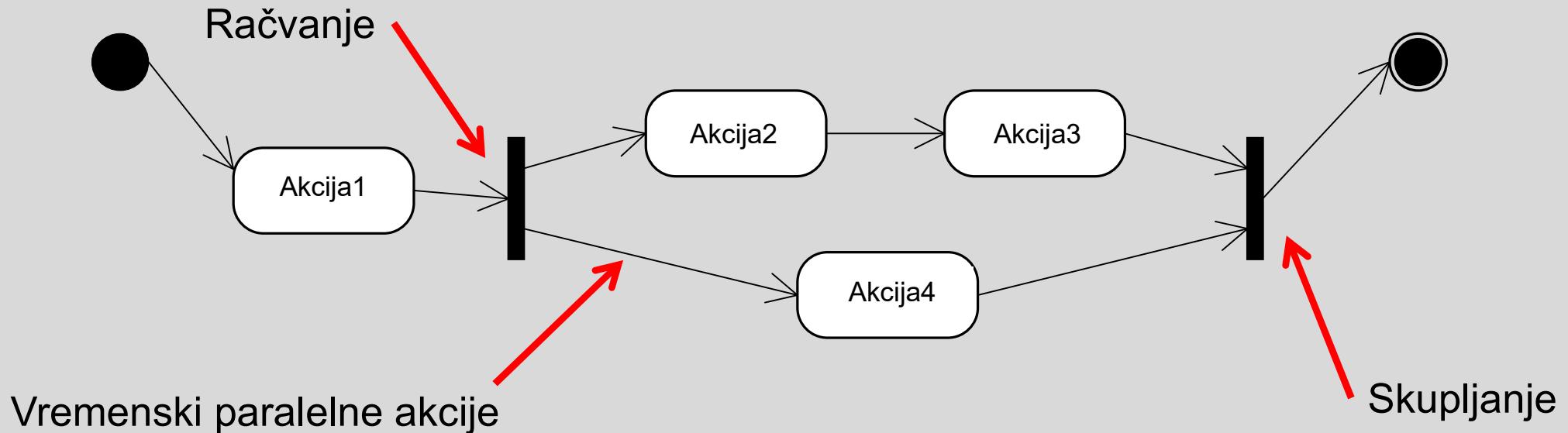
- **Odluka ili uvjetno grananje** prikazano je bijelim (“praznim”) dijamantnim oblikom.



- U jednu odluku može ući i izaći **proizvoljno mnogo akcija**. Svaka odluka ima **jedinstveni uvjet**.
- Statičko uvjetovanje grananja.

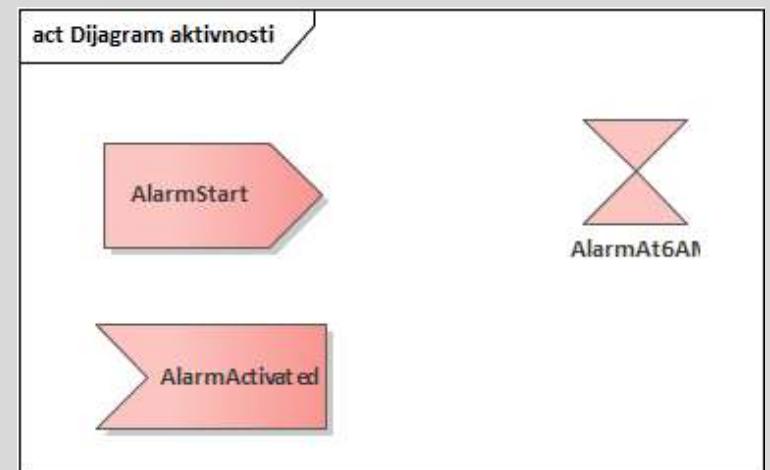
Elementi dijagrama aktivnosti: račvanje i skupljanje

- **Račvanje (*fork*) i skupljanje (*join*)** koristi se u slučaju vremenski paralelnog odvijanja akcija.
- Dijagram se račva nakon neke akcije i ponovno skuplja (**sinkronizira**) nakon prolaska kroz paralelne akcije.



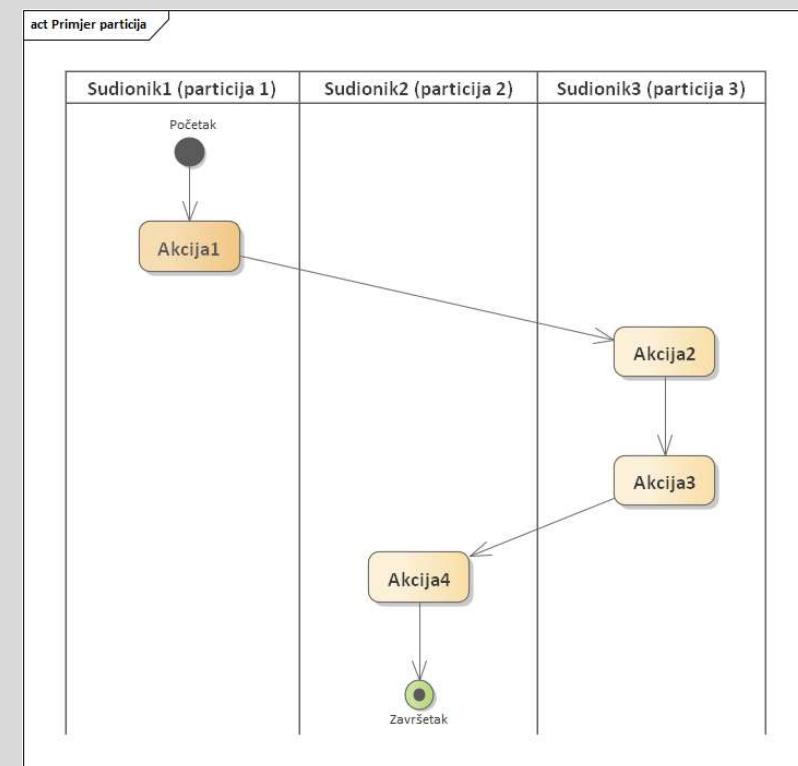
Elementi dijagrama aktivnosti: signali/događaji

- **Signali (engl. *signals*) i događaji (engl. *events*)**
 - koriste se da bi nadomjestili postojanje događaja kod dijagrama stanja
- Postoje tri vrste signala/događaja:
 1. **Slanje signala** (engl. *sending signal, generating signal*),
 2. **Prihvaćanje događaja** (engl. *receiving event, accepting event*)
 3. **Vremenski događaji** (engl. *time event*)



Plivačke staze

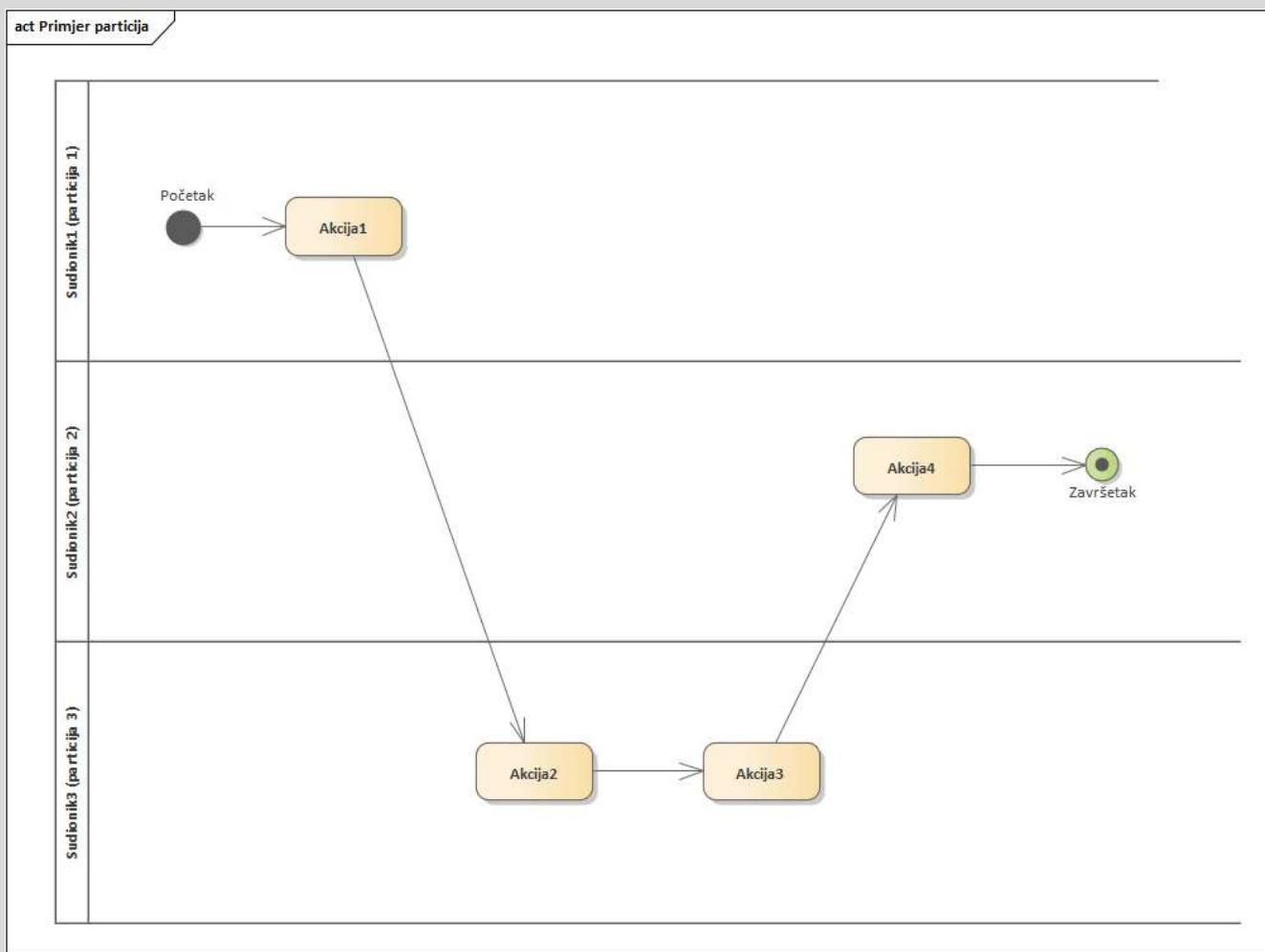
- „**Plivačke staze**“ (engl. *swimlanes*) grupiraju akcije koje izvode isti aktori.
 - Vertikalne i/ili horizontalne particije razgraničene linijama.
 - Pridonose boljoj organizaciji dijagrama kada se želi istaknuti koji od dijelova sustava što izvodi.



Plivačke staze odgovaraju sudionicima/aktorima u obrascima uporabe i dijagramima slijeda.

„Plivačke staze”

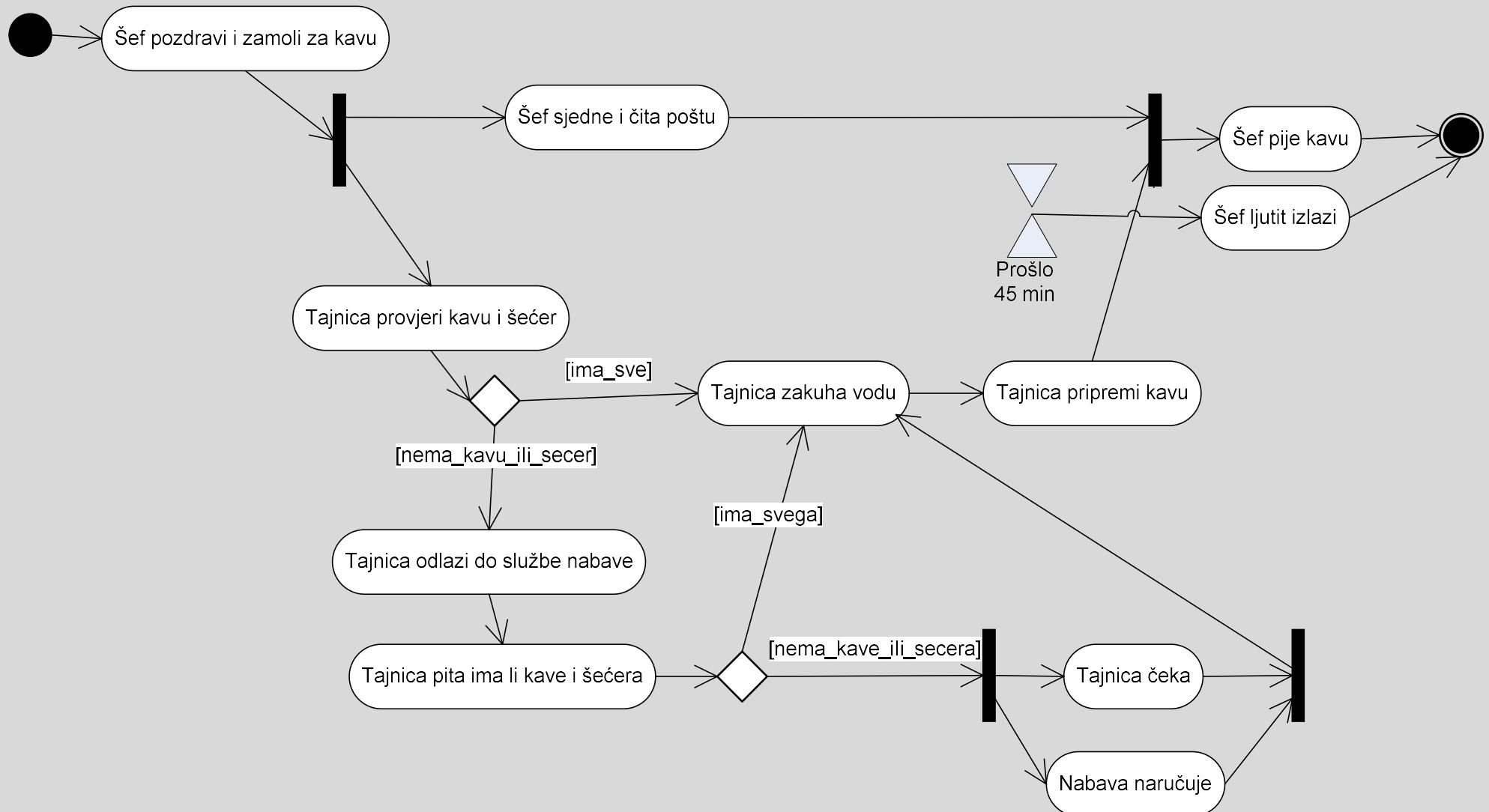
- Mogu biti i vodoravno položene.



Primjer: Kuhanje jutarnje kave u tvrtci

Modelirajte postupak kuhanja jutarnje kave u nekoj tvrtki dijagramom aktivnosti. Pretpostavite da u dotičnoj tvrtki tajnica svako jutra kuha šefu kavu. Slijed aktivnosti je sljedeći. Po dolasku, šef pozdravi tajnicu i zamoli ju za kavu. Nadalje, šef sjedne i čita pristiglu poštu. Za to vrijeme, tajnica najprije provjeri ima li dovoljno kave i šećera u uredu. Ako ima dovoljno kave i šećera, tada stavi vodu da zakuha. U slučaju da nema dovoljno kave ili šećera, tajnica ode do službe nabave poduzeća. Ako služba nabave ima kavu ili šećer, tada joj to daju i ona se vrati nazad u ured i zakuha vodu za kavu. U slučaju da služba nabave nema kavu ili šećer, tada oni naruče da im se ista što prije dostavi. Budući da tajnica zna da šef ne može bez jutarnje kave, ona ostaje kod službe nabave sve dok ne dobije kavu. Čim kavu dobije, tajnica se vraća nazad u ured i zakuha kavu. Ako u međuvremenu prođe više od 45 minuta otkako je šef naručio kavu, šef prestaje čitati poštu i izlazi lјutit iz ureda. Tajnica u svakom slučaju čim ima kavu i šećer i kad voda zakuha pripremi kavu, čak i u slučaju da je šef već izašao. Ako je šef još tamo, šef pije kavu.

Rješenje primjera



Napomena: U rješenje je moguće uključiti plivačke staze šefa, tajnice i nabave.

Laboratorijske vježbe

UML DIJAGRAMI AKTIVNOSTI U DOKUMENTU ZAHTJEVA

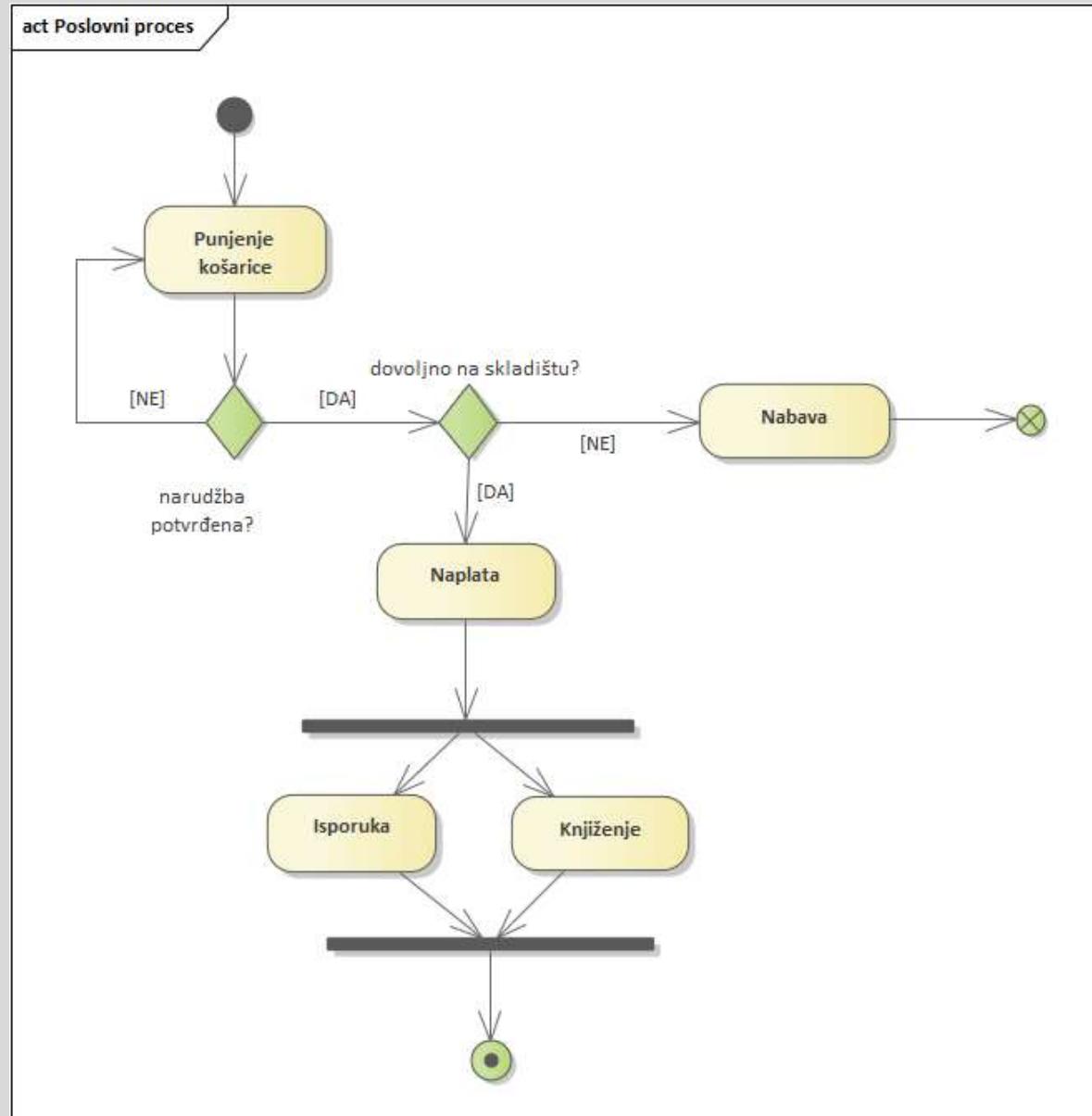
Dijagram aktivnosti

- Na više razina:
 - Poslovni proces - kontekst
 - Dodatni opis obrasca uporabe

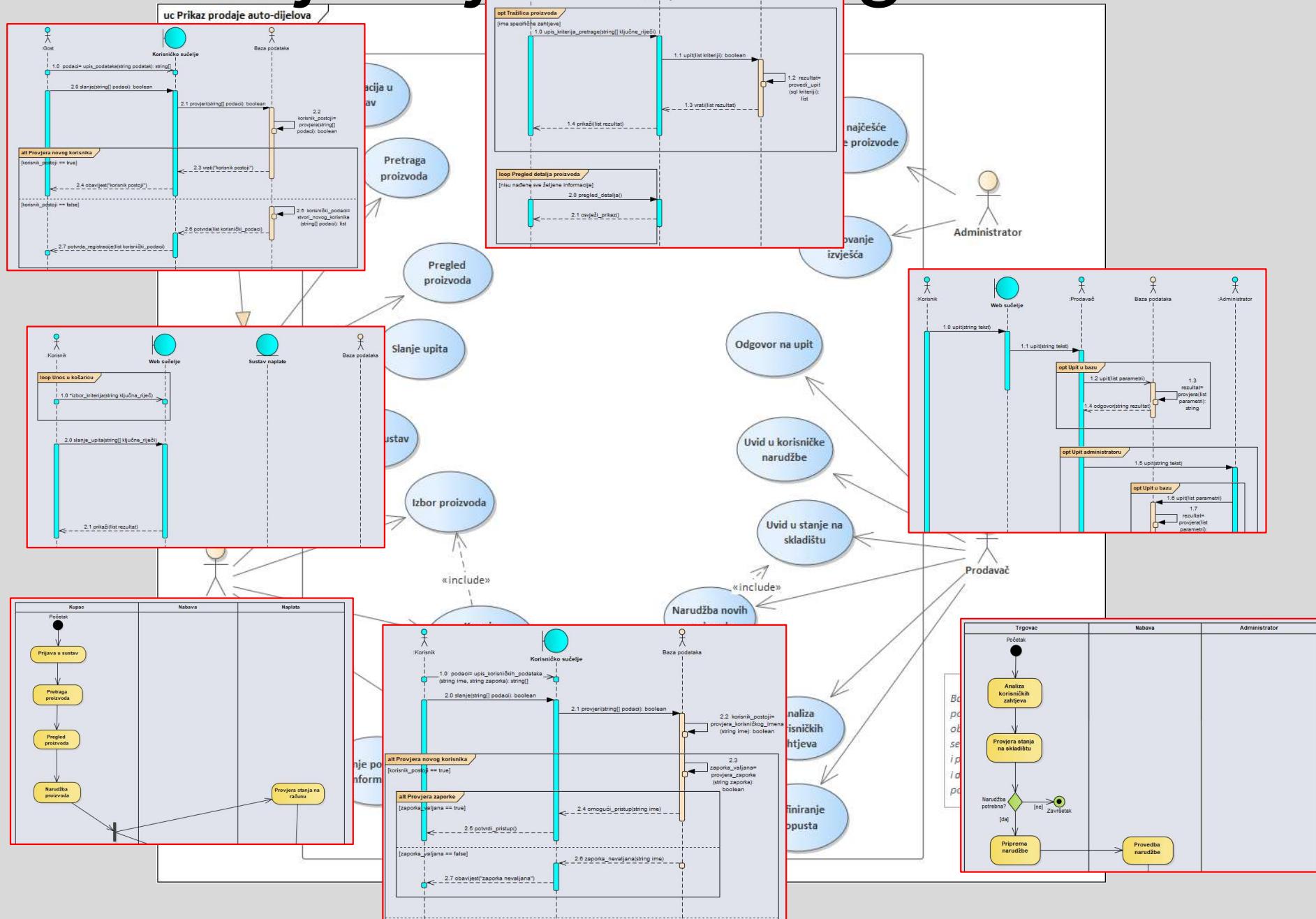
Gledište poslovnog procesa

- Kako se sustav koristi u širem poslovnom procesu → procesni model
- UML dijagram aktivnosti se koristi za prikaz poslovnog procesa

Poslovni kontekst prodaje autodijelova



Primjer cjeleokupnog sustava



Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

REFERENCE I LITERATURA

- Sveučilišna zbirka zadataka iz UML-a - A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014., dostupno u *Skriptarnici i knjižnici Fakulteta elektrotehnike i računarstva, te Nacionalnoj i sveučilišnoj knjižnici u Zagrebu*.
- Predavanja ovog predmeta
- Allen Holub's UML Quick Reference:
<http://www.holub.com/goodies/uml>
- Booch G., Jacobson I., Rumbaugh J. "UML Distilled".
- Nastavni materijali kolegija Oblikovanje programske potpore, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI KOMPONENTI

Dijagram komponenti

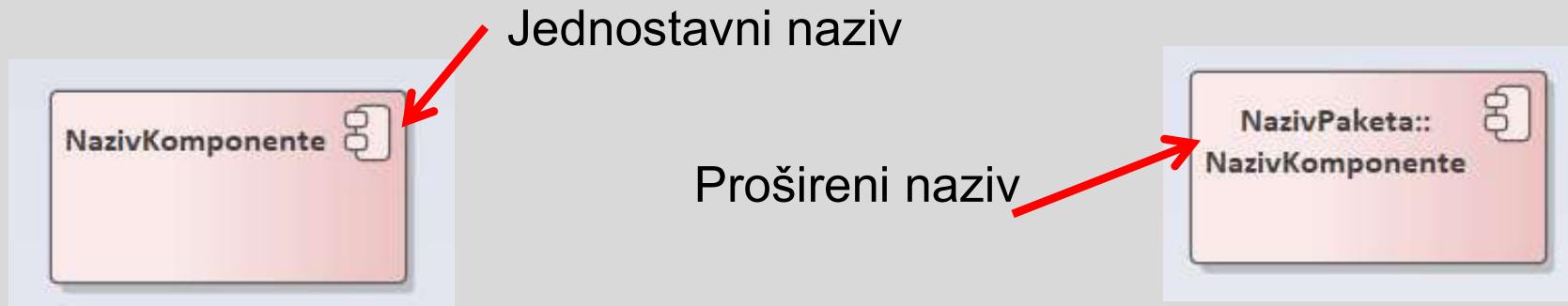
- Prikaz arhitekture programske potpore
 - gotovi dijelovi ili strukturne cjeline sustava i njihovi međusobni odnosi.
- **Strukturni UML dijagram, opisuje statične odnose:**
 - organizaciju i međuovisnost između implementacijskih komponenti programske potpore
- Zajedno s dijagramima razmještaja nazivaju se **fizički dijagrami:**
 - Pojedinci komponenti nalaze se u dijagramu razmještaja.

Komponente

- Komponenta \equiv zasebna cjelina programske potpore s vlastitim sučeljem
- Komponenta \equiv fizička i stvarna implementacija logičkih elemenata sustava
- \rightarrow Fizičke cjeline sustava
- \rightarrow Izvršne datoteke, programske knjižnice, tablice, datoteke i svi drugi dokumenti.
- Može sadržavati druge komponente \rightarrow prikaz unutarnje strukture

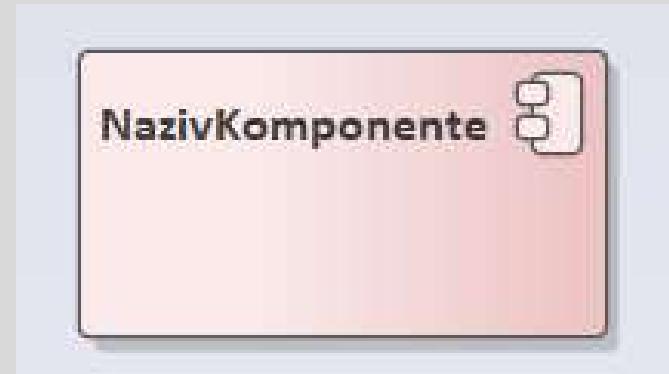
Svojstva komponenti

- Svaka komponenta ima jedinstven naziv ili ime koje je razlikuje od ostalih komponenti u dijagramu.
 - **Jednostavni naziv** (engl. *simple name*): niz znakova s određenim značenjem za korisnike dijagrama. U praksi uključuju i nastavak njihovih datoteka.
 - **Naziv puta** (engl. *path name*): naziv paketa kojemu komponenta pripada. Zapisuje se kao prefiks jednostavnom nazivu.

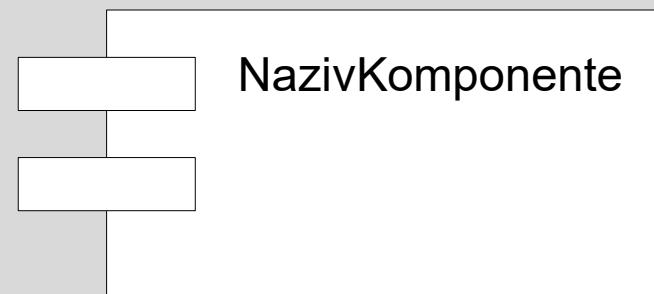


Notacija komponente

- Od UML standarda 2.0:

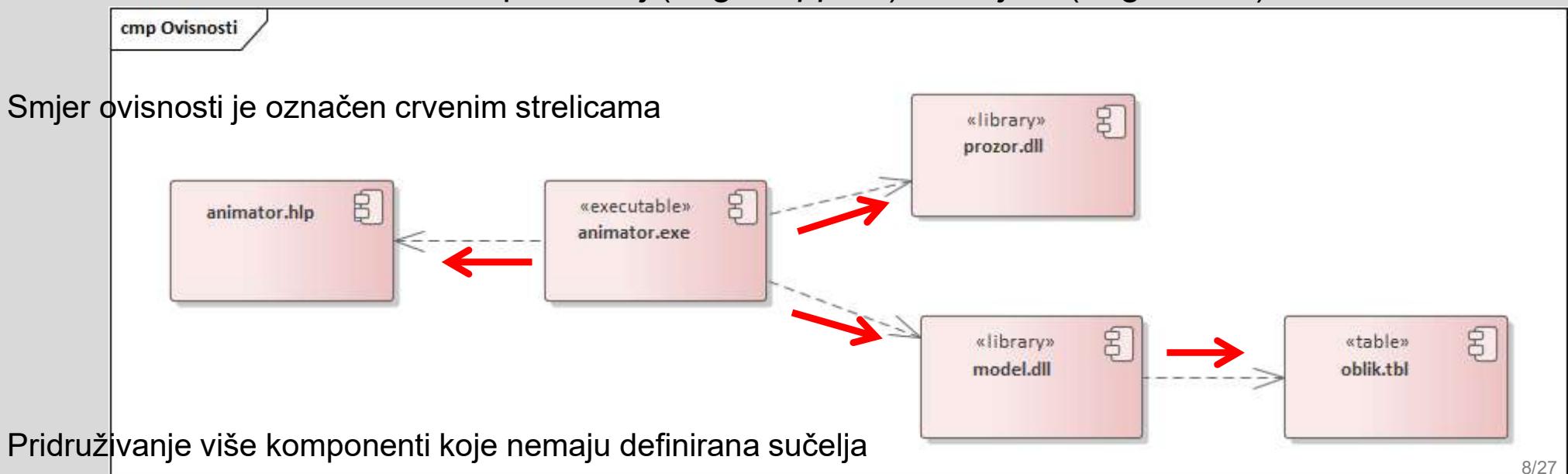


- U standardu 1.x:



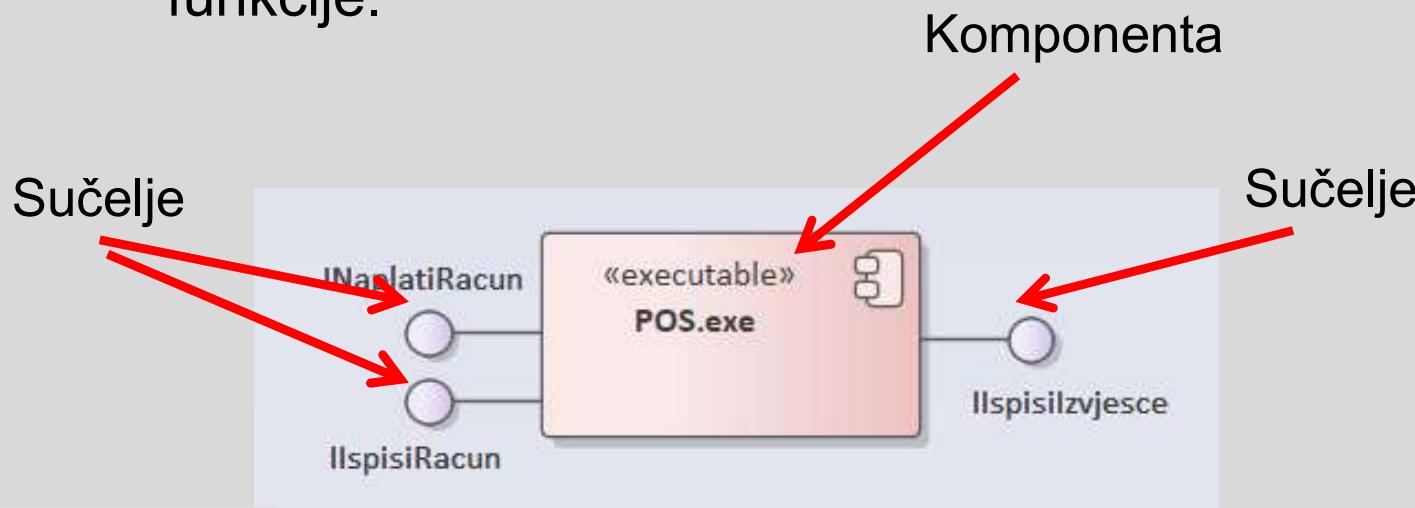
Pridruživanje komponenti

- **Pridruživanje** (engl. *association*) ili **veza** opisuje odnose između komponenti ili pojedinaca komponenti.
 - Komponente se pridružuju sučeljima ako su definirana.
 - Dijagram komponenti u pravilu koristi jednu vrstu veze: **ovisnost** (engl. *dependency*).
 - Ovisnost je uvijek jednosmjerna: “**B ovisi o A**” u smjeru strelice.
 - **A** se naziva isporučitelj (engl. *supplier*) i **B** klijent (engl. *client*).



Sučelja komponenti (1)

- Sučelje je kolekcija operacija koja određuju usluge komponente.
- Ponašanje komponente definirano je pomoću njezinih sučelja. Komponenta može biti zamijenjena drugom komponentom ako i samo ako su njihova sučelja identična. To je temeljena ideja ponovnog iskorištenja koda (engl. *software reuse*).
 - U UML dijagramima uporaba sučelja nije uvijek nužna. Komponente se mogu povezati neposredno. Ali zbog veće izražajnosti dijagrama potrebno je odrediti sučelja i njihove funkcije.

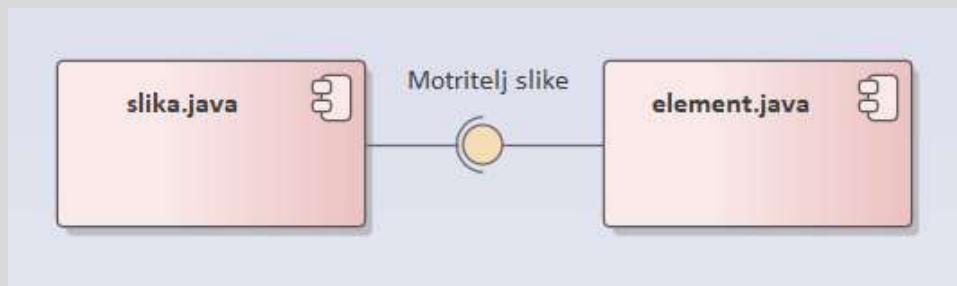


Sučelja komponenti (2)

- Sučelja komponenti mogu biti prikazana na dva načina:
 - **Ikonizirani oblik** (engl. *iconic form*). Najčešći način prikaza sučelja. Komponenta koja realizira sučelje povezana je sa sučeljem s izostavljenom vezom realizacije (engl. *elided realization relationship*).
 - **Prošireni oblik** (engl. *expanded form*). Prikaz sučelja pomoću veze realizacije. Izgled ove veza identičan je vezi realizacije u dijagramima razreda. Operacije sučelja **moraju biti prikazani** u dijagramu.

Sučelja komponenti (2)

- Ikonizirani oblik

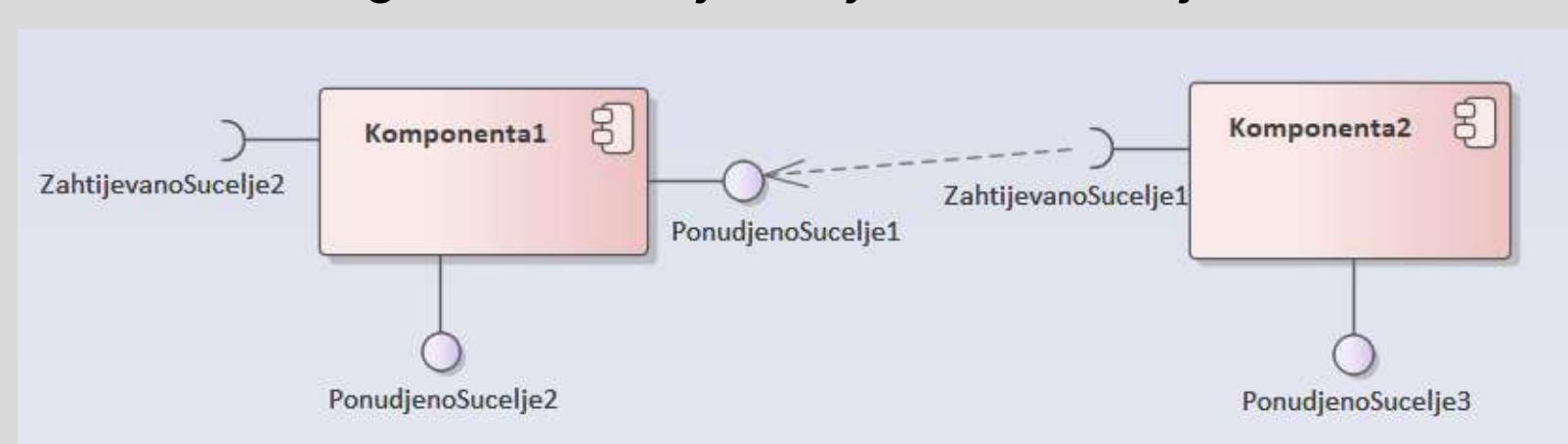


- Prošireni oblik



Sučelja komponenti (3)

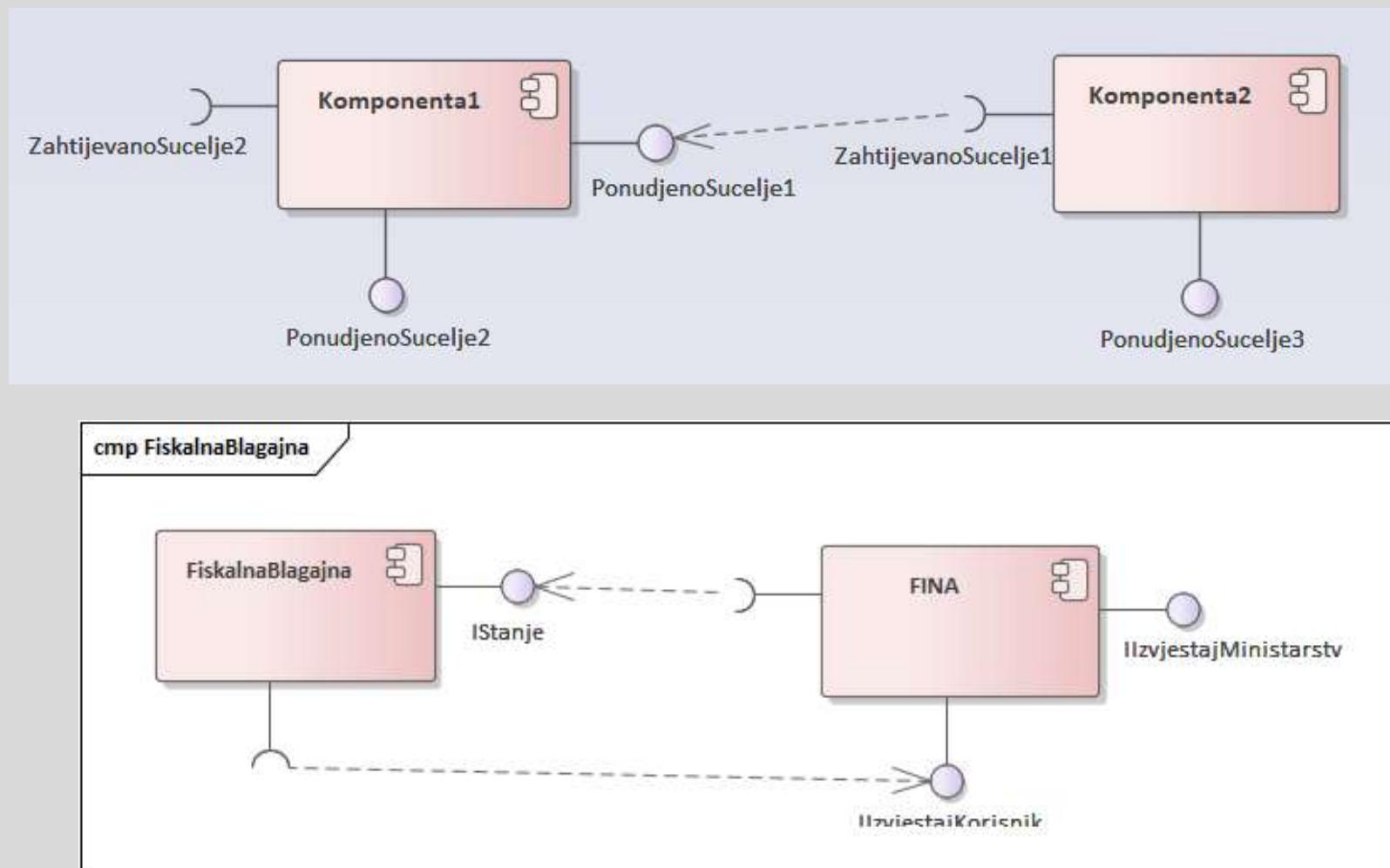
- **Ponuđeno sučelje** (engl. *export* ili *provided interface*) je sučelje koje neka komponenta realizira kao uslugu za druge komponente.
- **Zahtijevano sučelje** (engl. *import* ili *required interface*) je sučelje koje neka komponenta koristi.
 - Jedna komponenta može realizirati više ponuđenih sučelja i koristiti neograničeni broj zahtijevanih sučelja.



Višestruka ponuđena i zahtijevana sučelja komponenti

Sučelja komponenti (3)

- Primjeri – višestruka ponuđena i zahtijevana sučelja komponenti:



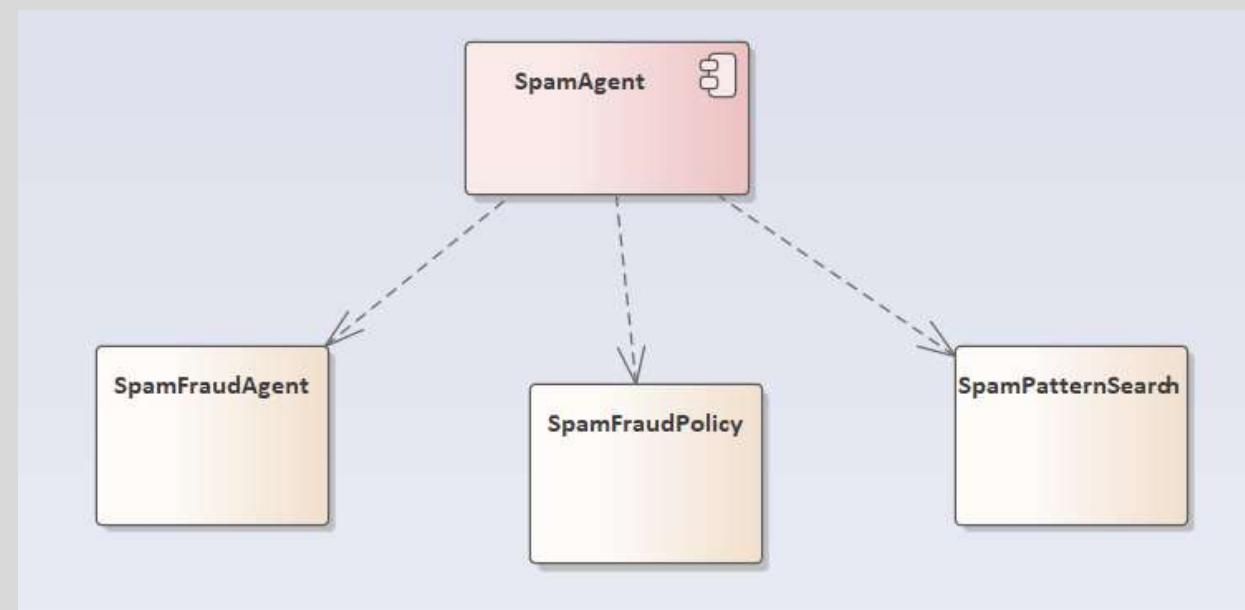
Razlike komponenti i razreda (1)

- Razlike komponenti i razreda (iz UML dijagrama razreda) su:
 - **Razina apstrakcije:** razredi predstavljaju logičku apstrakciju sustava, dok su komponente fizički (stvarni i opipljivi) artefakti.
 - **Pogled na sustav:** komponente i razredi pripadaju različitim pogledima na sustav s drugačijim razinama apstrakcije.
 - **Funkcionalnost:** razredi imaju vlastite atribute i operacije koje mogu neposredno izvršavati. Komponente imaju pristup samo onim operacijama koje se mogu dosegnuti kroz sučelja.

Odnos komponenti i razreda (2)

- U iznimnim slučajevima moguće je dijagramom komponenti prikazati razrede koji tvore komponentu.
 - Koristi se veza ovisnosti.

Primjer: DLL-datoteka spamagent.dll sastoji se od tri razreda: SpamAgent, SpamPolicy i SpamPatternSearch.



Vrste komponenti (1)

- UML definira tri vrste komponenti:
 1. **Komponente isporuke/razmještaja** (engl. *deployment components*)
 2. **Komponente radnog proizvoda** (engl. *work product component*)
 3. **Komponente izvođenja** (engl. *execution components*)
- Imaju isti UML simbol, ali razlikuju se po stereotipovima.

Vrste komponenti (2)

- **Komponente razmještaja/isporuke** (engl. *deployment components*) su one komponente koje su **nužne i dovoljne** za ispravan rad dizajniranog sustava.
- Primjeri:
 - Dinamički povezane biblioteke (engl. *dynamic linked libraries*, DLL) i izvršne datoteke (engl. *executables*, EXE).
 - Također: COM+, CORBA, Enterprise Java Bean, tablica baza podataka i druge izvršne datoteke.

Vrste komponenti (3)

- **Komponente radnog proizvoda** (engl. *work product component*) su one datoteke koje su **nastale tijekom razvoja sustava**.
- Primjeri:
 - Datoteke s izvornim kodom (engl. *source code*)
 - Datoteke s podacima (engl. *data files*)
- Komponente radnog proizvoda koriste se za realizaciju drugih komponenti.

Vrste komponenti (4)

- **Komponente izvođenja** (engl. *execution components*) nastaju kao **posljedica izvođenja i rada sustava**.
- Primjer:
 - COM+ objekti jer nastaju izvođenjem DLL datoteka.
 - Program Debug Database (PDB) datoteke nastale tijekom izrade .NET projekata.
 - Vlastite posredne (engl. *intermediate*) datoteke bilo kojeg formata.

Stereotipovi komponenti

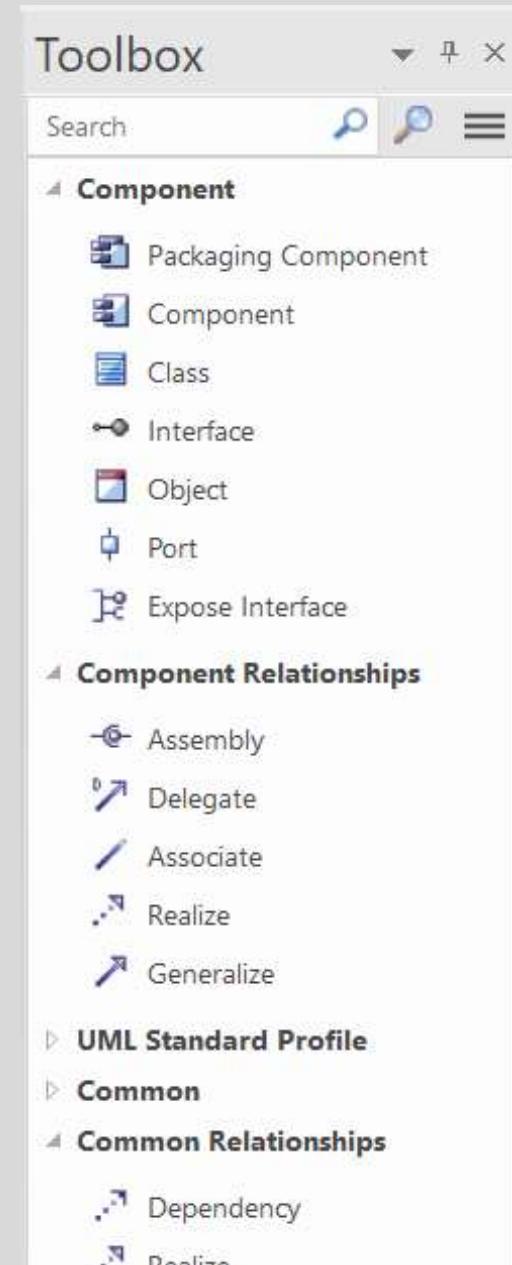
- Primjeri stereotipova za komponente:
 - «**executable**» - Komponenta može biti izvršena u čvoru
 - «**library**» - Komponenta je statička ili dinamička objekta biblioteka
 - «**table**» - Komponenta je tablica baze podataka
 - «**file**» - Komponenta je dokument s izvornim kodom ili podacima
 - «**document**» - Komponenta je dokument općeg značenja ili sadržaja
- UML ne definira specifične ikone za stereotipove, ali potiče se korištenje u praksi standardiziranih slikovnih simbola za izvršne datoteke, DLL datoteke, tablice baze podataka, itd.

Dijagram komponenti - Općenito

- <https://www.uml-diagrams.org/component-diagrams.html>

Dijagram komponenti u EA 15

- Komponenta
- Sučelje
- Sklapanje (engl. *assembly*)
- Realizacija
- Ovisnost



Paketi: Organizacija komponenti u dijagramu

- **Komponente se hijerarhijski organiziraju i grupiraju u pakete** slično kao što se organiziraju razredi.
- UML v1.x dopušta organizaciju komponenti **pomoću pridruživanja (veze) ovisnosti**.

Primjer: Modeliranje komponenti sustava

Nakon instalacije, Windows aplikacija za računalnu animaciju „Animator“ sastoji se od izvršne datoteke 'animator.exe' koja koristi DLL datoteke 'prozor.dll', 'model.dll', 'crtaj.dll' i 'prikazi.dll'.

Konfiguracija aplikacije nalazi se u formatiranoj datoteci 'animator.ini', a sustav pomoći u binarnoj datoteci 'animator.hlp'.

Izvršna datoteka 'animator.exe' je u inačici 3.1.4.

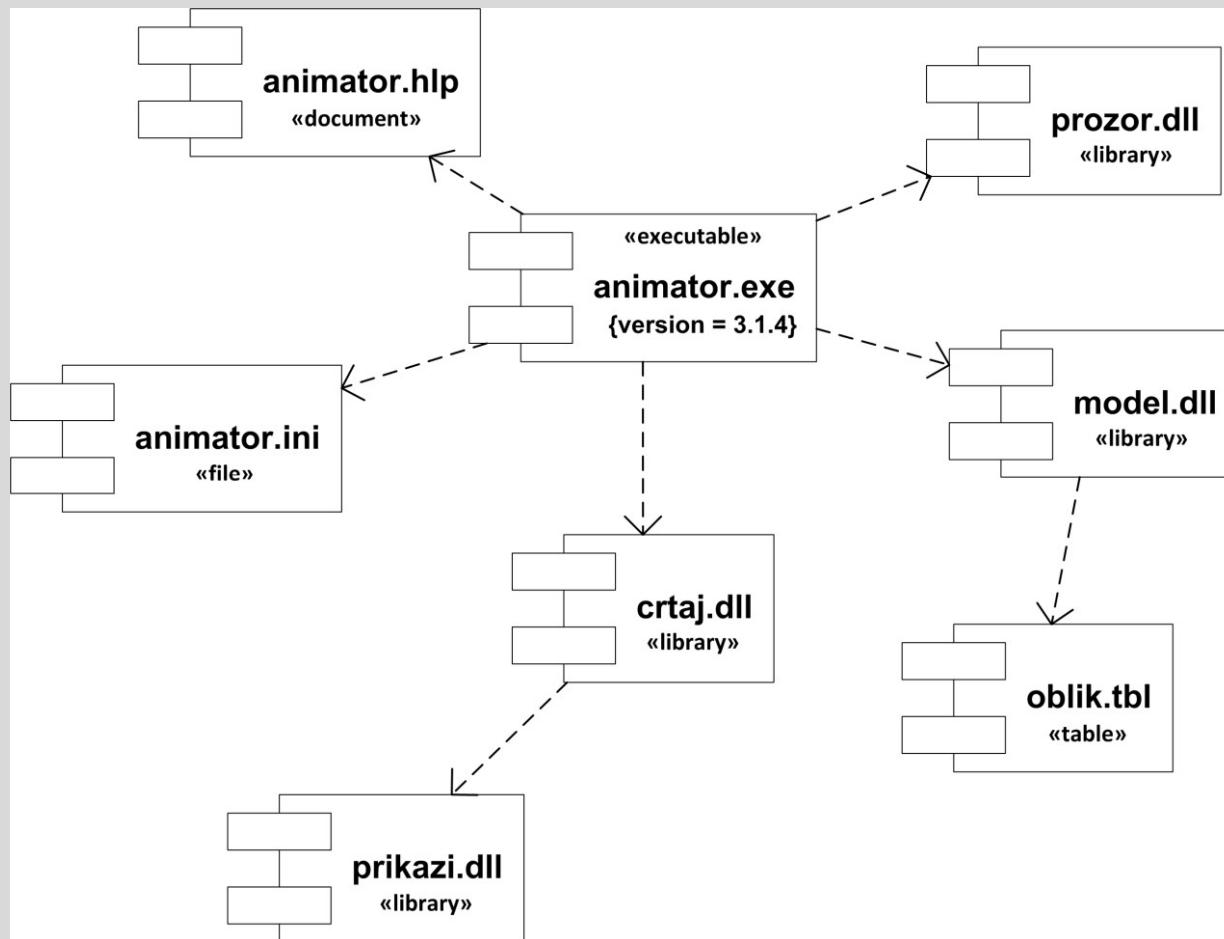
Biblioteka 'model.dll' pohranjuje podatke u tablicu baze podataka pod nazivom 'Oblik', koja se sastoji od datoteke 'oblik.tbl'.

Biblioteka 'crtaj.dll' koristi 'prikazi.dll'.

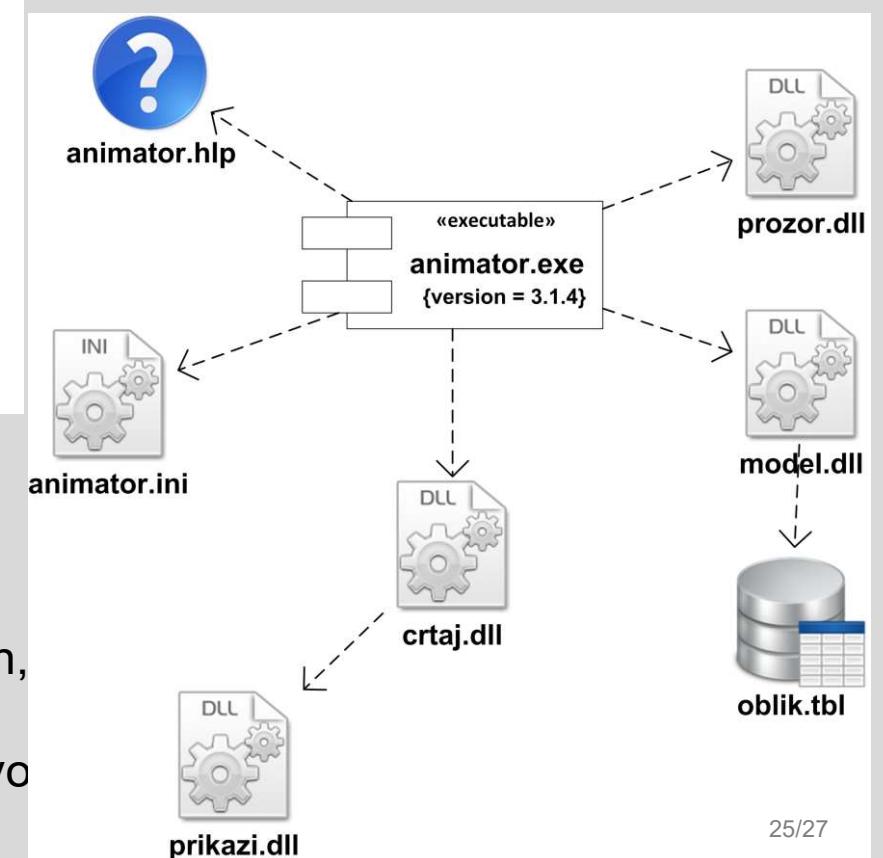
Prikažite navedene komponente u dijagramu.

Rješenja primjera

Koriste se stereotipovi komponenti i standardni simboli za prikaz datoteka. Jedina veza u dijagramu je ovisnost.



Zbog jasnoće dijagrama često se koriste normirane ikone umjesto simbola UML-komponenti. Ako je dijagram složen, s puno komponenti i veza, upotreba ikona pridonosi jednostavnijem i bržem razumijevanju dijagrama, pogotovo kod osoba koje nisu stručne u UML-u.



Laboratorijske vježbe

UML DIJAGRAMI KOMPONENTI U DOKUMENTU OSTVARENJA

Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

REFERENCE I LITERATURA

- Sveučilišna zbirka zadataka iz UML-a - A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014., dostupno u *Skriptarnici i knjižnici Fakulteta elektrotehnike i računarstva, te Nacionalnoj i sveučilišnoj knjižnici u Zagrebu*.
- Predavanja ovog predmeta
- Allen Holub's UML Quick Reference:
<http://www.holub.com/goodies/uml>
- ArgoUML manual:
<http://argouml.tigris.org>
- Booch G., Jacobson I., Rumbaugh J. "UML Distilled".
- Prezentacije predavanja i drugi dokumenti kolegija Oblikovanje programske potpore, dodiplomski studij, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI RAZMJEŠTAJA

Dijagram razmještaja

- **Dijagram razmještaja, dijagram isporuke, dijagram rasporeda** (engl. *deployment diagrams*)
- prikazuje sklopolje i programsku potporu potrebnu za implementaciju sustava u stvarnom radnom okruženju.
 - **Strukturni UML dijagram, opisuje statične odnose s fizičkog aspekta implementacije.** Isto kao dijagrami komponenti.
- Elementi koji tvore dijagrame razmještaja su:
 1. Čvorovi (engl. *nodes*)
 2. Komponente (engl. *components*)
 3. Njihova međusobna pridruživanja ili veze (engl. *associations*)
 - Osim navedenih elemenata u dijagramima razmještaja mogu se naći paketi i komentari.

Elementi dijagrama razmještaja

- Razlike između čvorova i komponenti:
 - Čvorovi predstavljaju **fizički vid sustava**, a komponente predstavljaju **logički vid sustava**.
 - Komponente sudjeluju u izvršavanju sustava, a čvorovi izvršavaju komponente. Čvorovi su računalni resursi koji omogućuju pokretanje i izvršavanje komponenti.
- **Čvor:**
 - Sklopovlje sustava kao što su razni poslužitelji.
 - Jedan čvor može simbolizirati više računala kao što je grozd poslužitelja.
- **Komponenta:**
 - Izvršne **datoteke**, stolne, mobilne i mrežne **aplikacije** koje čine opisani sustav.



Svojstva čvorova

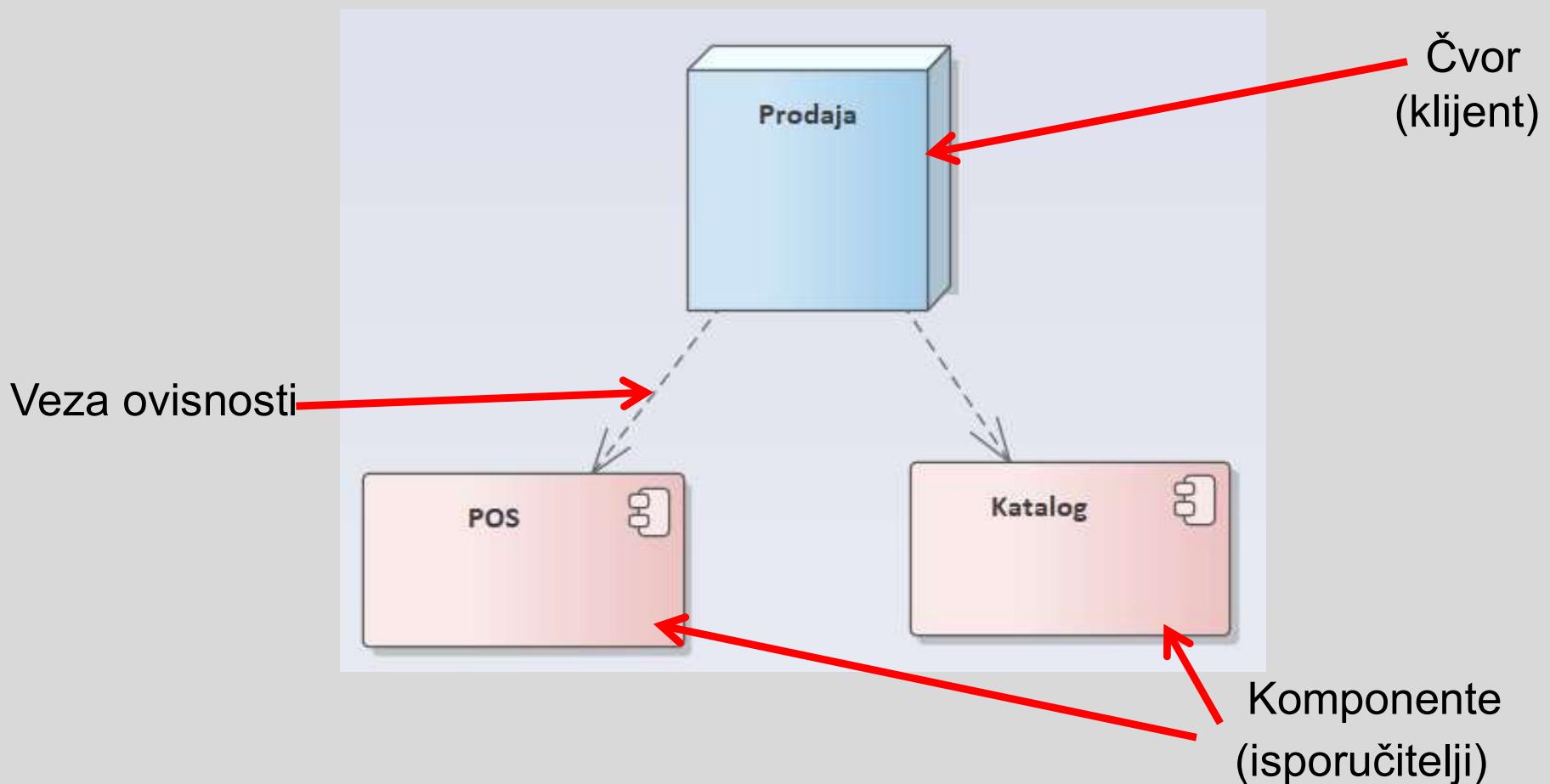
- Svaki čvor ima jedinstven naziv ili ime koje je razlikuje od ostalih čvorova u dijagramu.
 - **Jednostavan naziv** (engl. *simple name*) je samo niz znakova s određenim značenjem za korisnike dijagrama.
 - **Naziv puta** (engl. *path name*) je naziv paketa kojemu čvor pripada. Naziv putanje uvijek prethodi (prefiks) imenu čvora. Npr. “server::backup”.
- Imena čvorova slijede ista pravila kao imena komponenti:
 - **Prošireni naziv** (engl. *extended name*) je oblik imena čvora s nazivom putanje. Takvi čvorovi pripadaju drugom paketu i nazivaju se **prošireni čvorovi** (engl. *extended nodes*).
 - Čvorovi s jednostavnim nazivom zovu se **jednostavni čvorovi** (engl. *simple nodes*).

Veze čvorova

- Čvorovi u dijagramima razmještaja mogu biti povezani s:
 1. Jednosmjernom vezom
 2. Dvosmjernom vezom (češće od jednosmjernih)
 3. Vezom ovisnosti
- Veza predstavlja komunikaciju između čvorova.
- Čvorovi i komponente su međusobno ovisni.
 - Ne uvijek, ali to je rijetko.
 - Neka komponenta može biti ovisna drugoj komponenti ili čvoru, i obrnuto (čvor može ovisiti o komponenti).
 - Smjer ovisnosti usmjeren je od klijentu (engl. *client*) prema isporučitelju (engl. *supplier*). Pri tome kažemo da klijent ovisi o isporučitelju.

Primjer: Ovisnost komponenti podsustava za naplatu

Neki podsustav za naplatu sastoji se od dvije komponente: 1) mjesto prodaje (engl. *Point Of Sale, POS*) i 2) katalog koji sadržava popis svih proizvoda koji se mogu prodati i njihovih jediničnih cijena. Podsustav ovisi o komponentama.

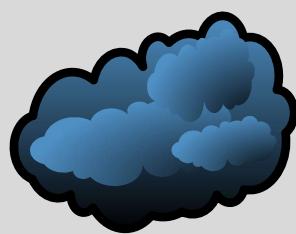


Proširenja dijagrama razmještaja (1)

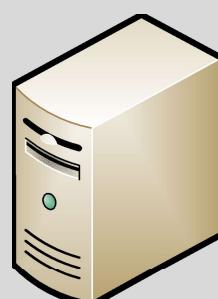
- Dijagrami razmještaja mogu se proširiti uvođenjem stereotipova i zamjenom simbola.
- **Dozvoljeni stereotipovi čvorova su:**
 - «**executionEnvironment**» - čvor koji obrađuje podatke i izvršava komponente (ili «**processor**»)
 - «**device**» - čvor koji ne može obrađivati podatke, ali predstavlja sklopoljje, neki uređaj ili sučelje prema fizičkom svijetu.

Proširenja dijagrama razmještaja (2)

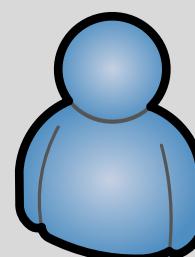
- Osim uvođenjem stereotipova dijagrame razmještaja moguće je proširiti korištenjem slika ili ikona za čvorove specifične namjene.
 - Umjesto standardnih UML simbola čvora.
 - Namjena je učinkovitije (brzo i jednoznačno) razumijevanje namjene čvorova.



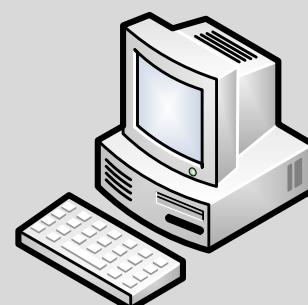
Telekomunikacijska
mreža



Poslužitelj



Korisnik



Klijent



datoteka.dll



baza.tbl

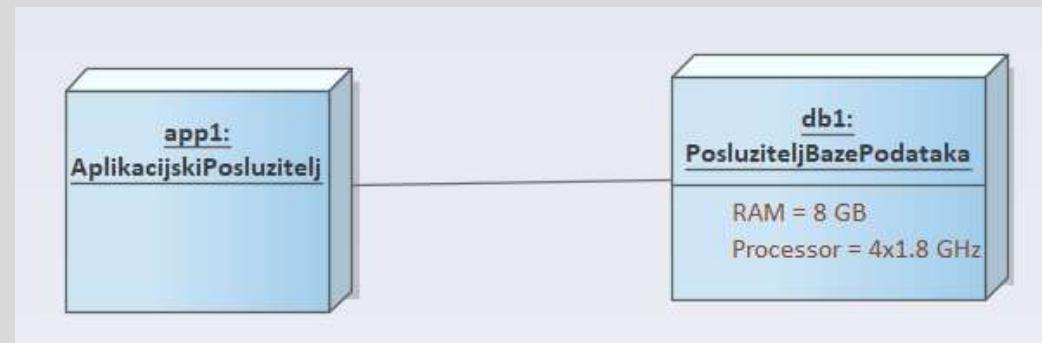


pomoc.hlp

- Učestalo korištenje istog simbola za procesore i uređaje modeliranog sustava različite namjene pridonosi nejasnoći dijagrama.

Instance čvorova i komponenti

- Dijagram razmještaja može prikazivati instance (pojedince) čvorova i komponenti.
 - Dijagram razmještaja postaje sličan dijagramu objekata.
 - Pojedince raspoznajemo po podcrtanim nazivima.
- U jednom dijagramu razmještaja nije dozvoljeno miješati definicije i pojedince čvorova ili komponenti
 - Prikazuju se samo definicije ili samo pojedinci.

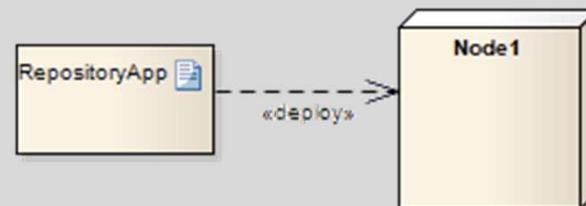


- Pojedince čvorova je korisno označiti najvažnijim podacima iz specifikacije računala i radne okoline, primjerice „Brzina procesora=3GHz, Radna memorija=16GB, Čvrsti disk=2TB, Verzija=3.1.4“.

Dijagram razmještaja u EA 15

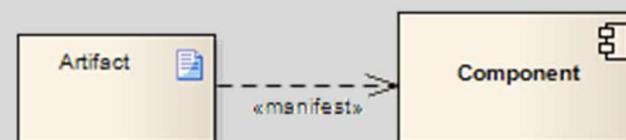
- Node ≡ čvor

- Device
 - Execution Environment



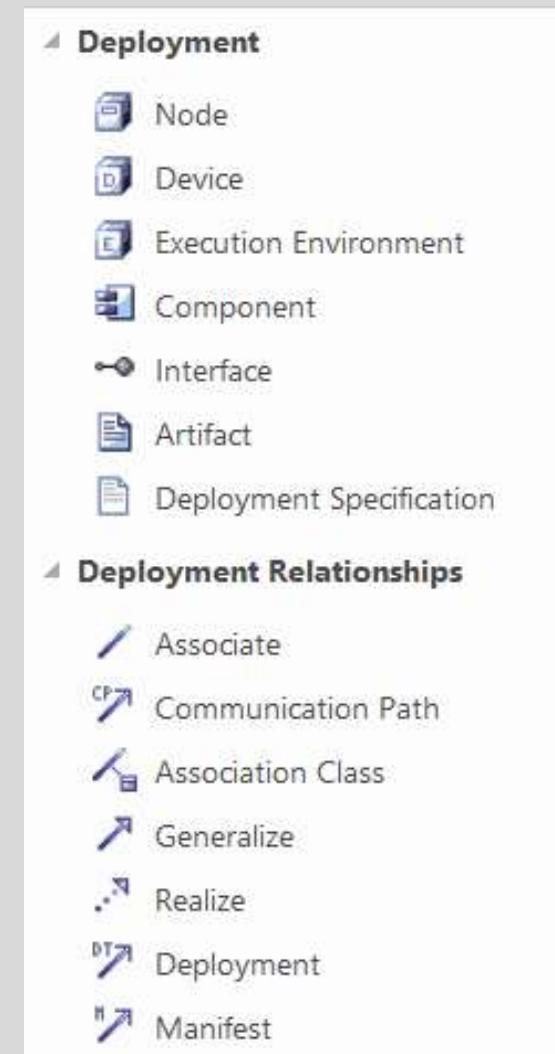
- Deployment

→ smještaj artefakta u čvoru

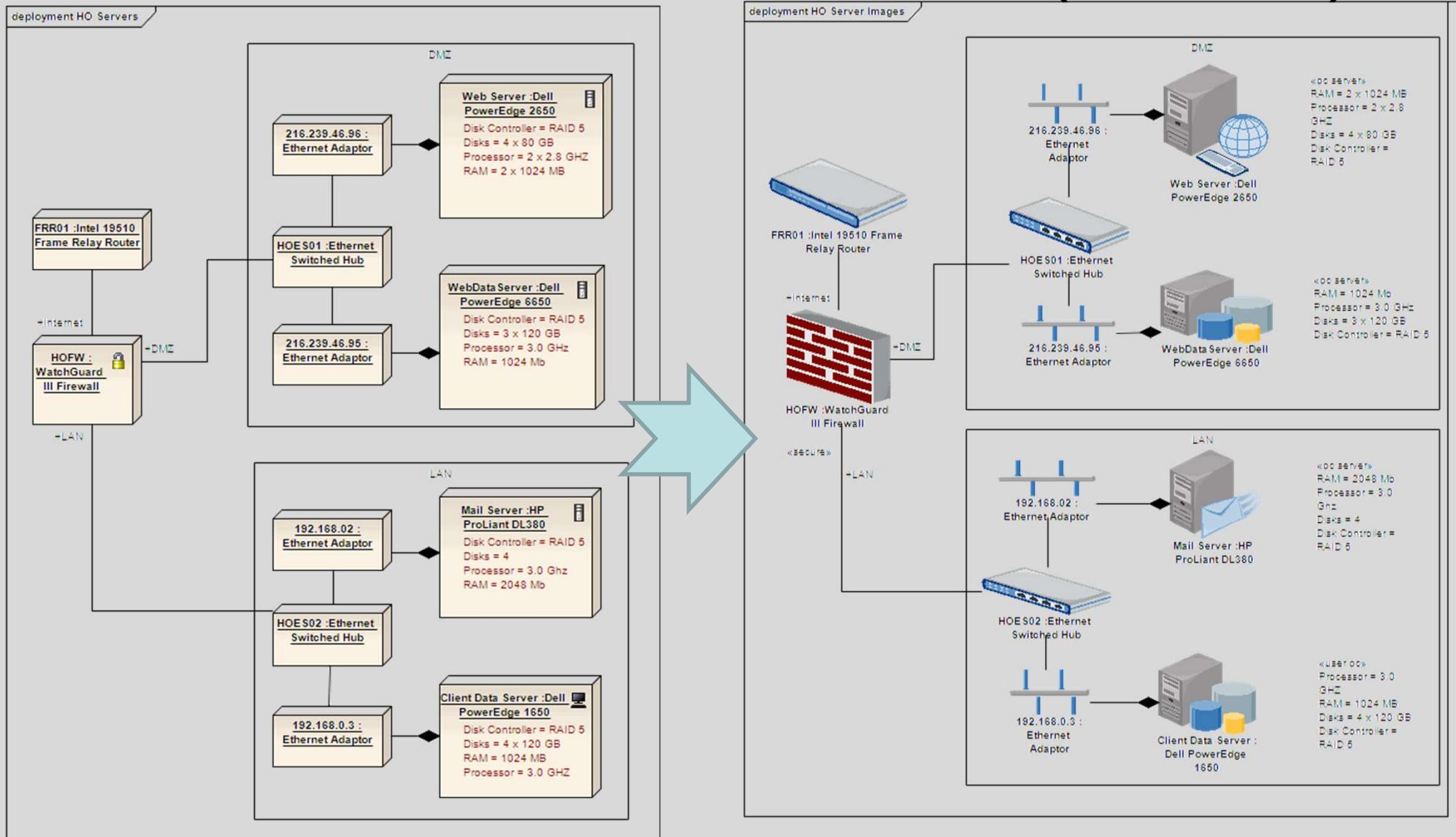


- Manifest

→ sadržaj artefakta ostvaruje element modela



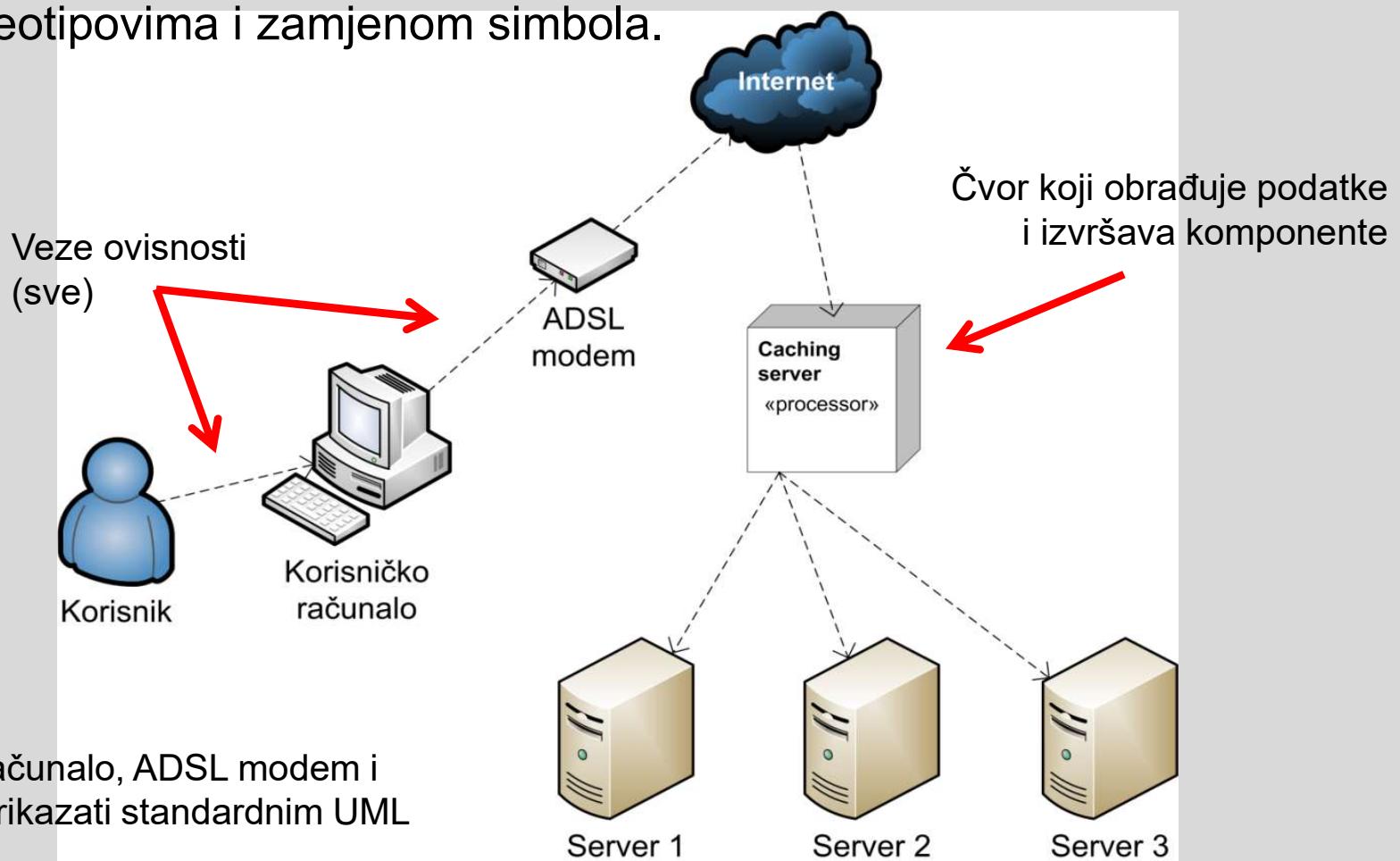
Uvoz datoteke simbola (EA 15)



[izvor: https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/deploymentdiagram.html]

Primjer: Raspodijeljeni informatički sustav

Neki raspodijeljeni višeslojni informatički sustav sastoji se od korisničke aplikacije koja se preko Interneta povezuje na poslužiteljsku aplikaciju. Korisnička aplikacija izvršava se na korisničkom računalu i spaja se na Internet preko ADSL-modema. Poslužiteljska aplikacija izvršava se na grozdu tri poslužitelja kojima upravlja poslužitelj privremene memorije (engl. *caching server*). Potrebno je izraditi dijagrame razmještaja opisanog sustava proširenjem stereotipovima i zamjenom simbola.



Laboratorijske vježbe

UML DIJAGRAMI RAZMJEŠTAJA U DOKUMENTU OSTVARENJA

Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

REFERENCE I LITERATURA

- Sveučilišna zbirka zadataka iz UML-a - A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014., dostupno u *Skriptarnici i knjižnici Fakulteta elektrotehnike i računarstva, te Nacionalnoj i sveučilišnoj knjižnici u Zagrebu*.
- Predavanja ovog predmeta
- Allen Holub's UML Quick Reference:
<http://www.holub.com/goodies/uml>
- ArgoUML manual:
<http://argouml.tigris.org>
- Booch G., Jacobson I., Rumbaugh J. "UML Distilled".
- Prezentacije predavanja i drugi dokumenti kolegija Oblikovanje programske potpore, dodiplomski studij, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

UML DIJAGRAMI RAZREDA

Napomena

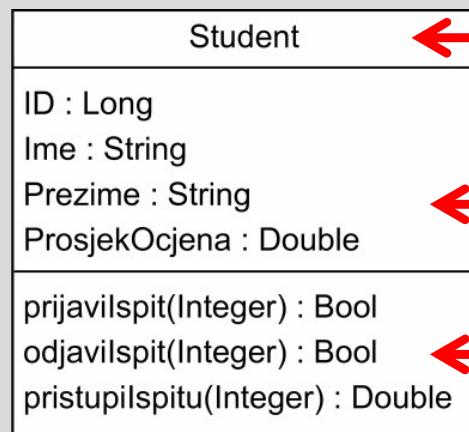
- Prikazani primjeri su rađeni u alatu ArgoUML i dijelom u alatu Enterprise Architect 15
- Svrha primjera je prikaz koncepata neovisno o upotrijebljenom alatu

Karakteristike dijagrama razreda (1)

- **Dijagram razreda** (engl. *class diagram*) opisuje vrste objekata unutar nekog sustava i njihove međusobne odnose.
 - opisuje razrede (klase) i njihove međusobne veze.
- Strukturalni UML dijagram, opisuje statičke odnose.
- **Dva osnovna tipa odnosa** između razreda:
 1. Pridruživanje (veza ili asocijacija)
 2. Podtip
- **Pridruživanje dijelimo na:**
 - **Jednosmjerno, dvosmjerno, refleksivno i agregacija.**
 - **Kompozicija je podvrsta agregacije.**

Karakteristike dijagrama razreda (2)

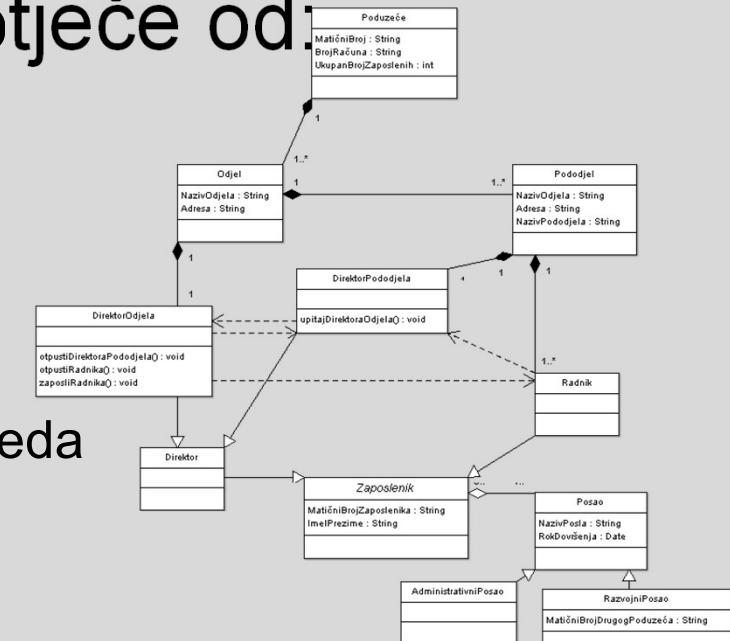
- Najčešći termin koji koristimo potječe od:
 - **Class dijagram**
- Hrvatski termini:
 - **Dijagram razreda ili dijagram klasa**



Jedinstveni naziv razreda

Skup atributa

Skup operacija



- Prikazuju razrede, atribute i operacije razreda, njihova svojstva i ograničenja, sučelja, pridruživanja, vlastite tipove podataka, enumeracije, pakete i komentare.

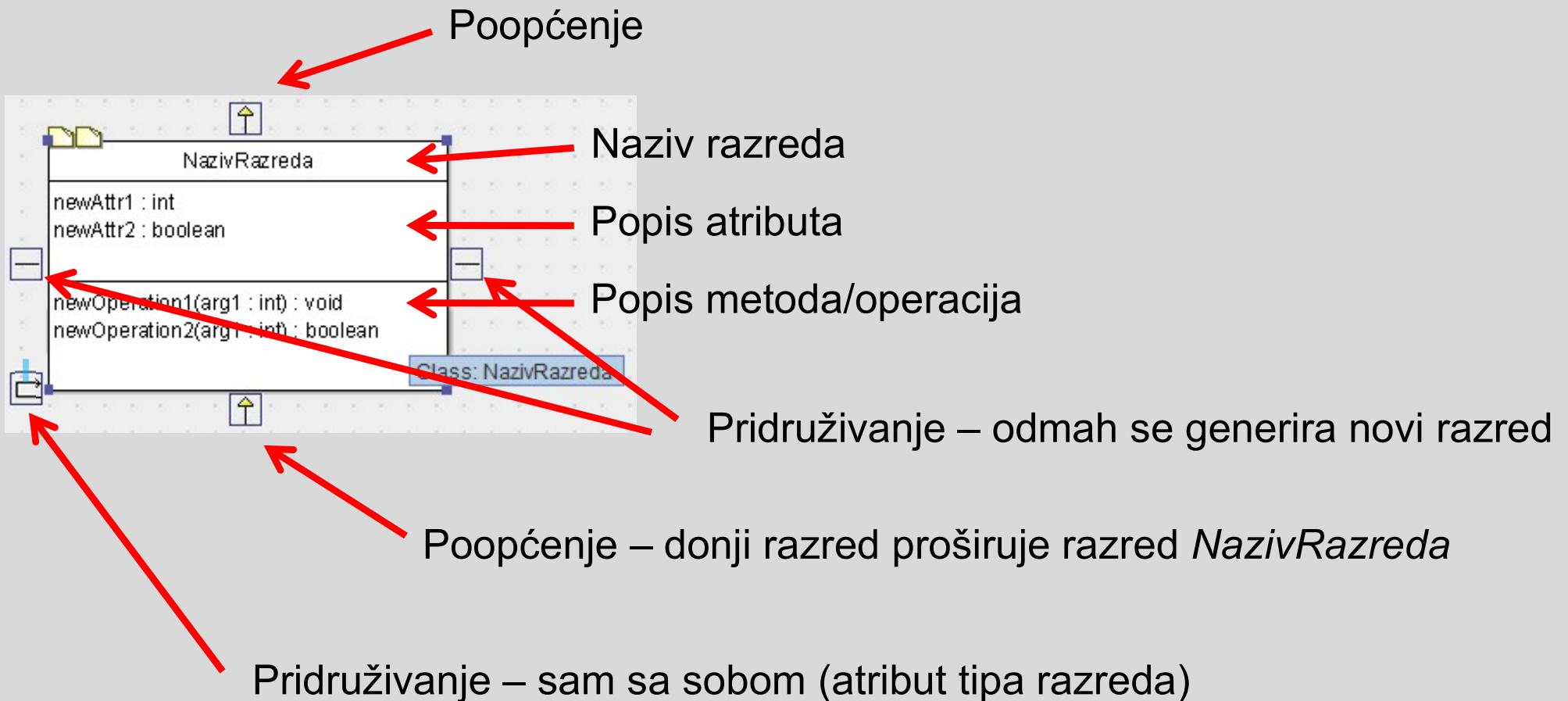
Načini uporabe (1)

- Općenito, dijagrami razreda mogu se koristiti na **tri različita načina** (pogleda, perspektive ili razine):
 - 1. Koncept**
 - Razmatranje neke domene. Dijagram ne mora biti povezan s programskom potporom. Neovisan o programskom jeziku.
 - 2. Specifikacija**
 - Sučelja sustava.
 - 3. Implementacija**
 - Izrada programske potpore. U primjeni najčešća perspektiva.
- Vrijedi za sve UML dijagrame, ali najočitije je s dijagramima razreda

Načini uporabe (2)

- Razine uporabe nisu formalni dio UML specifikacije, ali su jako korisne u postupku izrade modela.
- Granice između razina nisu strogo definirane. Nema velikih razlika između koncepta i specifikacije. Između specifikacije i implementacije postoji veća razlika.
- Možemo se poslužiti stereotipovima <>> u označivanju razreda, npr. <<implementation>> ili <<type>> za implementaciju i koncept.

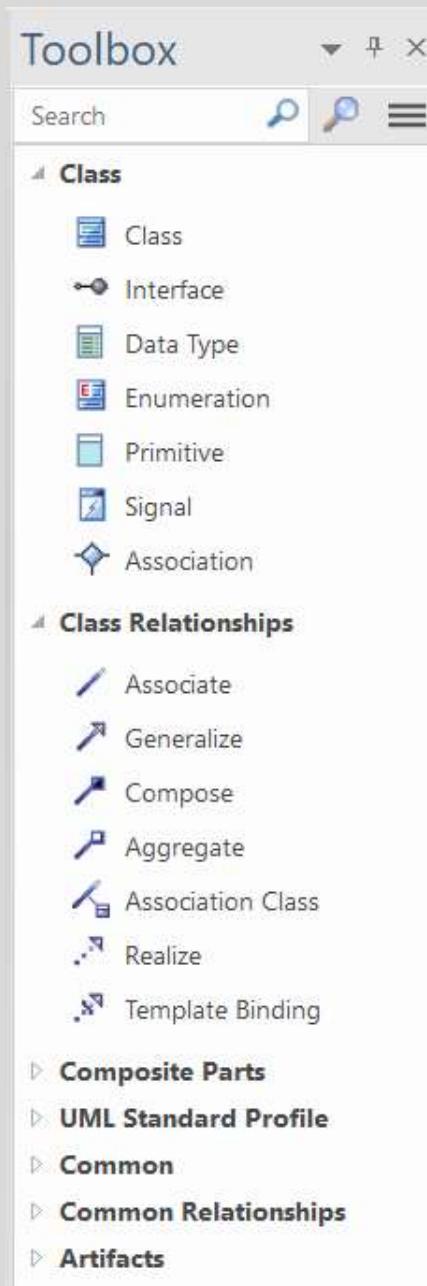
Simbol razreda u uređivaču ArgoUML



Napomena: Izvorni kôd u ArgoUML dobiven je iz kartice Source Code. Kôd je koristan za ilustraciju dijagrama, ali potrebno ga je proširiti i provjeriti njegovu ispravnost u nekom od jezičnih procesora.

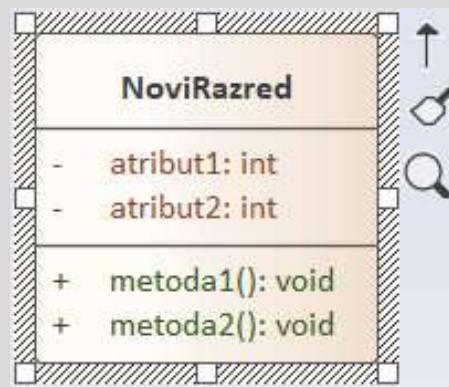
Uporaba razreda u alatu EA 15

- Razredi
- Odnosi



Uporaba razreda u alatu EA 15

- Razred



- Svojstva

The screenshot shows the EA 15 software interface with the 'Features' dialog open for the 'NoviRazred' class. The dialog has tabs for Attributes, Operations, Receptions, Parts / Properties, and Interaction Points. The Attributes tab is selected, displaying the following table:

Name	Type	Scope	Stereotype	Alias	Initial Value
atribut1	int	Private			
atribut2	int	Private			
New Attribute...					

Razred

- **Razred ili klasa** (engl. *class*) je osnovni tvorbeni element UML class dijagrama.

1. Objekt

- Predstavlja entitet iz stvarnog svijeta ili neki koncept.
- Apstrakcija nečega što ima dobro definirane granice i smisao u sustavu.

2. Razred

- Opis skupine objekata sa sličnim svojstvima.
- Svaki objekt je pojedinac (instanca) jednog razreda.

Atributi

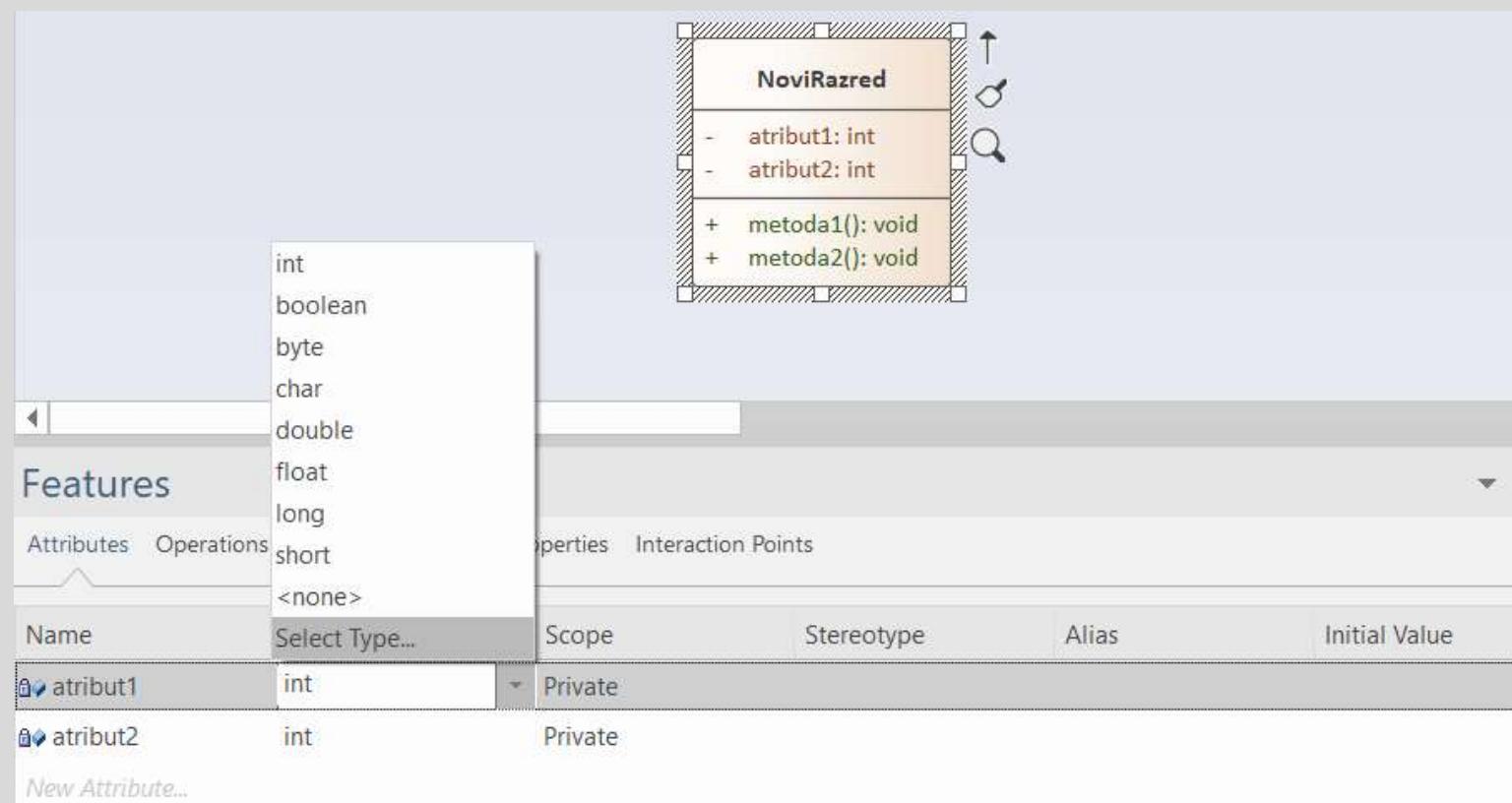
- **Atributi** (engl. *attributes*) razreda imaju sljedeća svojstva:
 - **Razina vidljivosti** (engl. *visibility*)
 - **Naziv** (engl. *name*)
 - **Vrsta ili tip** (engl. *type*)
 - **Početnu vrijednost** (engl. *initial value*)
- Dodatno:
 - **Promjenjivost** (engl. *changeability*)
 - **Modifikator** (engl. *modifier*)

Stupanj vidljivosti atributa

- Moguće su **četiri vrijednosti**:
 - **Public** (simbol: +)
 - Atribut je dostupan svim razredima i paketima.
 - **Private** (simbol: -)
 - Atribut je dostupan samo unutar istog razreda.
 - **Protected** (simbol: #)
 - Atribut je dostupan unutar istog razreda i izvedenih razreda.
 - **Package** (simbol: ~)
 - Atribut je dostupan svim razredima istog paketa.
- Mogu se primijeniti i na operacije razreda

Vrsta ili tip atributa

- Ugrađeni



- Vlastiti

Promjenjivost atributa (primjer ArgoUML)

- **addOnly**
 - Vrijednost atributa može se samo povećavati.
- **changeable**
 - Vrijednost atributa može se nesmetano mijenjati. Podrazumijevana (*default*) postavka.
- **static**
 - Modifikator, vrijednost atributa ne mijenja se (konstanta) i ne ovisi o životu objekta.
 - Ključne riječi u programskim jezicima *static* ili *const*.

Promjenjivost atributa (primjer ArgoUML)

- **frozen**
 - Vrijednost atributa (ili asocijacije) može se promijeniti samo jednom tijekom života (engl. *lifetime*) pripadajućeg objekta.
- **read-only**
 - Vrijednost atributa ne može se mijenjati izvan objekta kojemu pripada.
 - Nije isto što i *frozen*.

Operacije

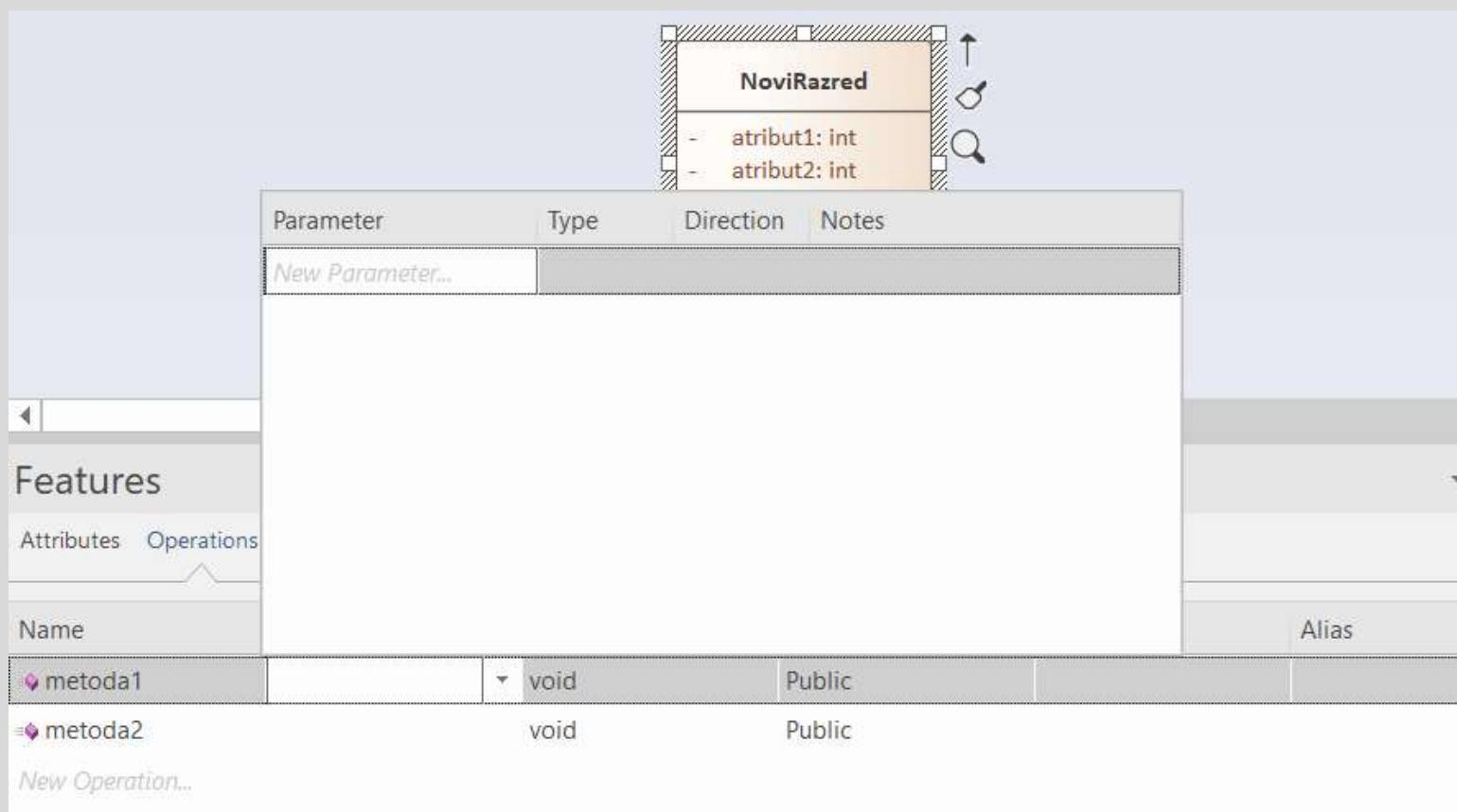
- **Operacije** (engl. *operations*) su procesi koje razred može izvršiti.
 - Drugim riječima, to su vlastite metode i funkcije razreda.
- Za njih možemo odrediti (primjer ArgoUML):
 - **Vidljivost** (*public*, *package*, *protected*, *private*)
 - Modifikatore (**static**, **abstract**, *leaf*, *root*, *query*)
 - Istodobnost (*sequential*, *guarded*, *concurrent*)
 - **Parametre ili argumente**

Operacije

- **Operacije** (engl. *operations*) su procesi koje razred može izvršiti.
 - Drugim riječima, to su vlastite metode i funkcije razreda.
- Za njih možemo odrediti (primjer ArgoUML):
 - **Vidljivost** (*public*, *package*, *protected*, *private*)
 - Modifikatore (*static*, *abstract*)
 - **Parametre ili argumente**

Operacije

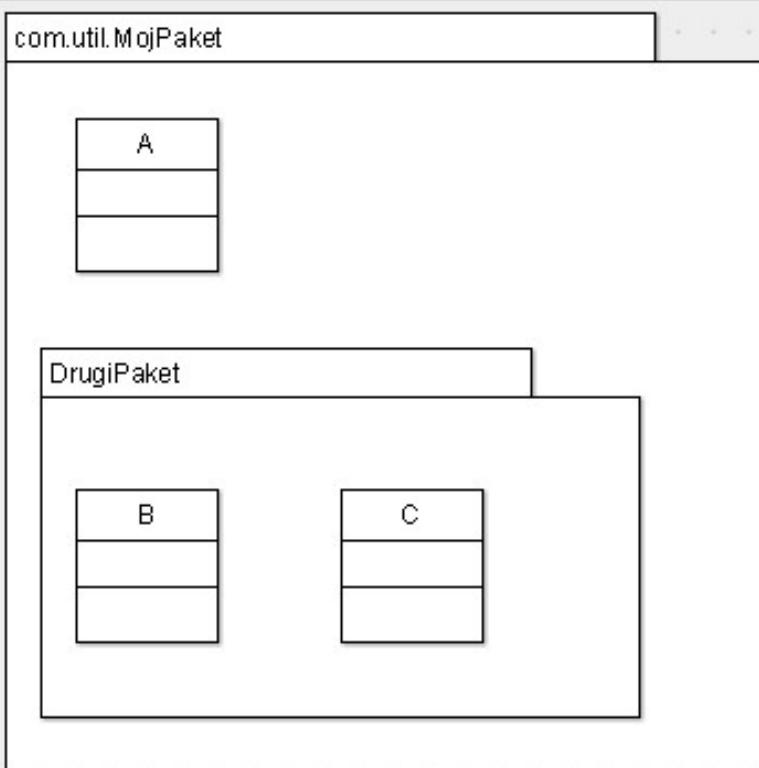
- Primjer EA 15:



Paket

- **UML paket** (engl. *package*) je skup različitih objekata.
- Svrha paketa je omogućiti hijerarhijsku organizaciju elemenata u UML dijagramu.
- Mogu sadržavati druge pakete, objekte, razrede, komponente, UC-ove, itd.
- U programskom kodu interpretira se kao namespace u C++ i C#, package u Javi, ...

Primjer paketa



Java

```
package com.util.MojPaket;  
  
public class A {}
```

C++

```
namespace com {  
    namespace util {  
        namespace MojPaket {  
            namespace DrugiPaket {  
  
                class B {};  
  
            } /* End of namespace  
                  com.util.MojPaket::DrugiPaket */  
        } /* End of namespace com.util.MojPaket */  
    } /* End of namespace com.util */  
} /* End of namespace com */
```

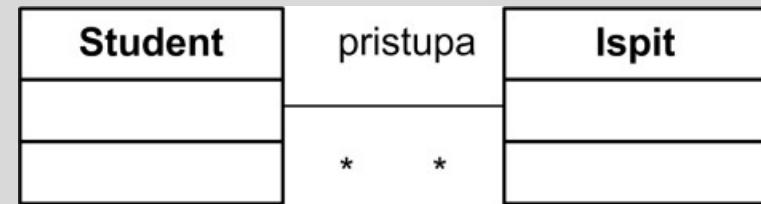
Java

```
package com.util.MojPaket.DrugiPaket;  
  
public class B {}
```

Pridruživanje

- **Pridruživanje** (engl. *association*) ili **veza** opisuje odnose između pojedinaca (instanci) razreda.

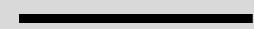
– “Student pristupa ispitu”



- Svaka veza ima dva vrha. Svaki vrh je pridružen (dodiruje) jedan od razreda u vezi.
- Vrh može imati vlastito ime:
 - **Naziv uloge** (engl. *role name*).
 - Vrhovi se još nazivaju **uloge** ili **role** (engl. *association role*).

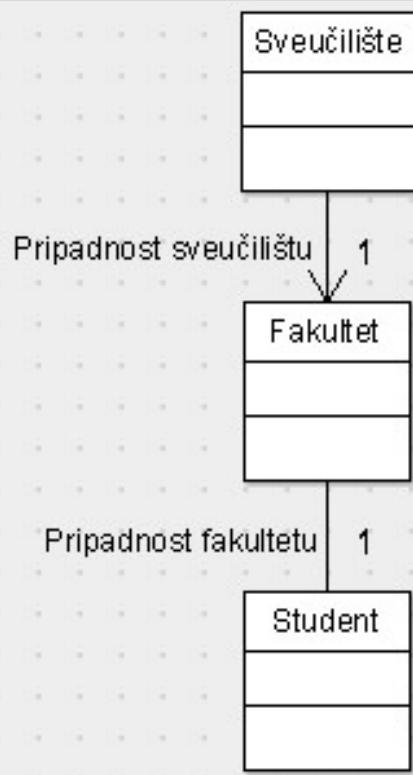
Smjer pridruživanja

- Ako je vrh neke asocijације označen strelicom onda je definiran njezin **smjer** (engl. *navigability*)
- Podjela veza po smjeru:
 - Jednosmjerna (unidirekcialna)
 - Smjer je definiran na samo jednom vrhu.
 - Dvosmjerna (bidirekcialna)
 - Smjer je definiran na oba vrha.
- Ako smjer nije definiran smatra se da je veza nepoznata (nedefinirana) ili bidirekcialna.



Primjeri pridruživanja

C++



Razred Sveučilište

```
class Fakultet;
class Sveučilište {
public:
    Fakultet *Pripadnost sveučilištu;
};
```

Razred Fakultet

```
class Student;
class Fakultet {
public:
    Student *Pripadnost fakultetu;
};
```

Razred Student

```
class Fakultet;
class Student {
public:
    Fakultet *Pripadnost fakultetu;
};
```

Java

Razred Sveučilište

```
public class Sveučilište {
    public Fakultet Pripadnost sveučilištu;
}
```

Razred Fakultet

```
public class Fakultet {
    public Student Pripadnost fakultetu;
}
```

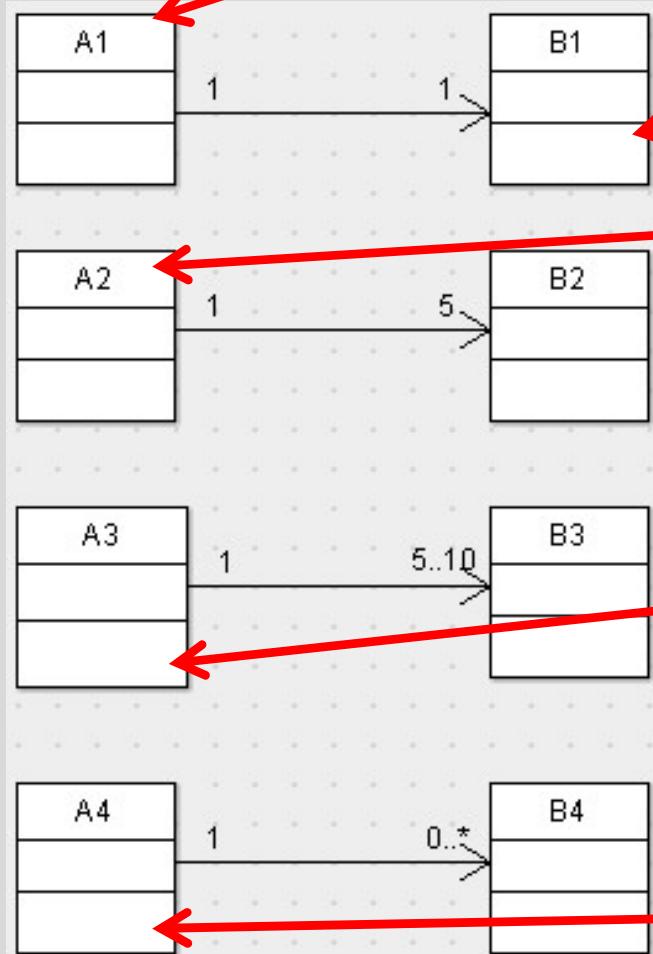
Razred Student

```
public class Student {
    public Fakultet Pripadnost fakultetu;
}
```

Višestrukost pridruživanja

- Vrh može određivati **višestrukost veze** (engl. *multiplicity*).
 - Govori nam koliko objekata može sudjelovati u određenom odnosu.
- Dozvoljene vrijednosti na **bilo kojoj strani** pridruživanja:
 - 1 = točno 1 pojedinac (podrazumijevana vrijednost)
 - n_1 = bilo koji točno određen broj, npr. 0, 1, 3, 5, 15
 - $n_1..n_2$ = između n_1 i n_2
 - $n_1..^*$ ili $n_1..n$ = između n_1 i više pojedinaca, neograničeno
 - $0..^*$ ili * ili n = više pojedinaca, neograničeno

Primjeri višestrukosti asocijacija



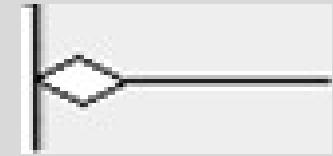
```
public class A1 {  
    public B1 myB1;  
}
```

```
import java.util.Vector;  
public class A2 {  
    public Vector myB2;  
}
```

```
import java.util.Vector;  
public class A3 {  
    public Vector myB3;  
}
```

```
import java.util.Vector;  
public class A4 {  
    public Vector myB4;  
}
```

Agregacija



- **Vrsta pridruživanja** koja pokazuje da jedan razred sadrži druge, tj. da je dio drugog razreda.
 - Razred je agregiran (sadržan) u drugom razredu → oblik odnosa cjelina-dio (tj. podskup-nadskup) → veza DIO-OD (engl. PART-OF).



C++

```
#ifndef Fakultet_h
#define Fakultet_h

#include <vector>
#include "Student.h"
```

Java

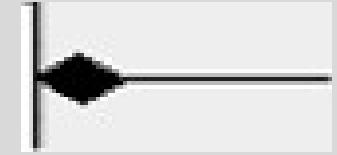
```
import java.util.Vector;

public class Fakultet {
    public Vector myStudent;
}
```

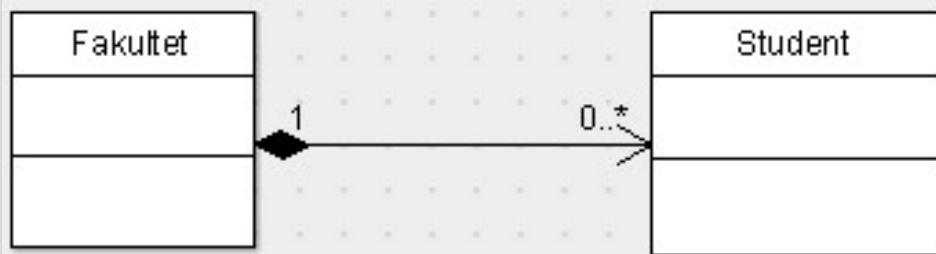
```
class Fakultet {
    public:
        std::vector< Student*> myStudent;
};

#endif // Fakultet_h
```

Kompozicija



- **Vrsta pridruživanja** slična agregaciji, ali kod uništavanja objekta (tj. pojedinca) uništavaju se i pojedinci razreda koji su dio tog objekta.



C++

```
#ifndef Fakultet_h  
#define Fakultet_h  
  
#include <vector>  
#include "Student.h"
```

Java

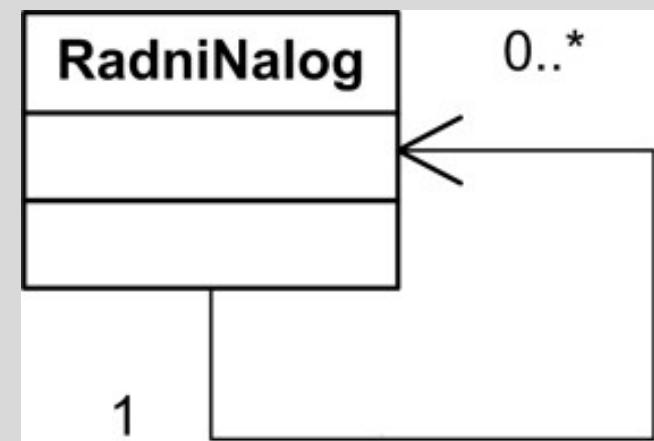
```
import java.util.Vector;  
  
public class Fakultet {  
  
    public Vector myStudent;  
}
```

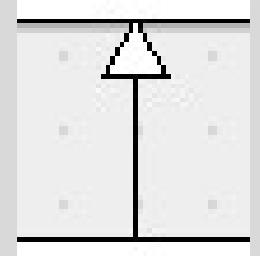
```
class Fakultet {  
    public:  
        std::vector< Student > myStudent;  
};  
  
#endif // Fakultet_h
```

Refleksivno pridruživanje

- Više pojedinaca istog razreda ponekad ovise jedan o drugome ili međusobno komuniciraju.
- Ova vrsta ovisnosti realizira se pomoću refleksivnog pridruživanja, agregacije ili kompozicije.

Moguća je jednosmjerna i dvosmjerna veza, agregacija, kompozicija, ovisnost, ali nije moguća veza poopćenja.





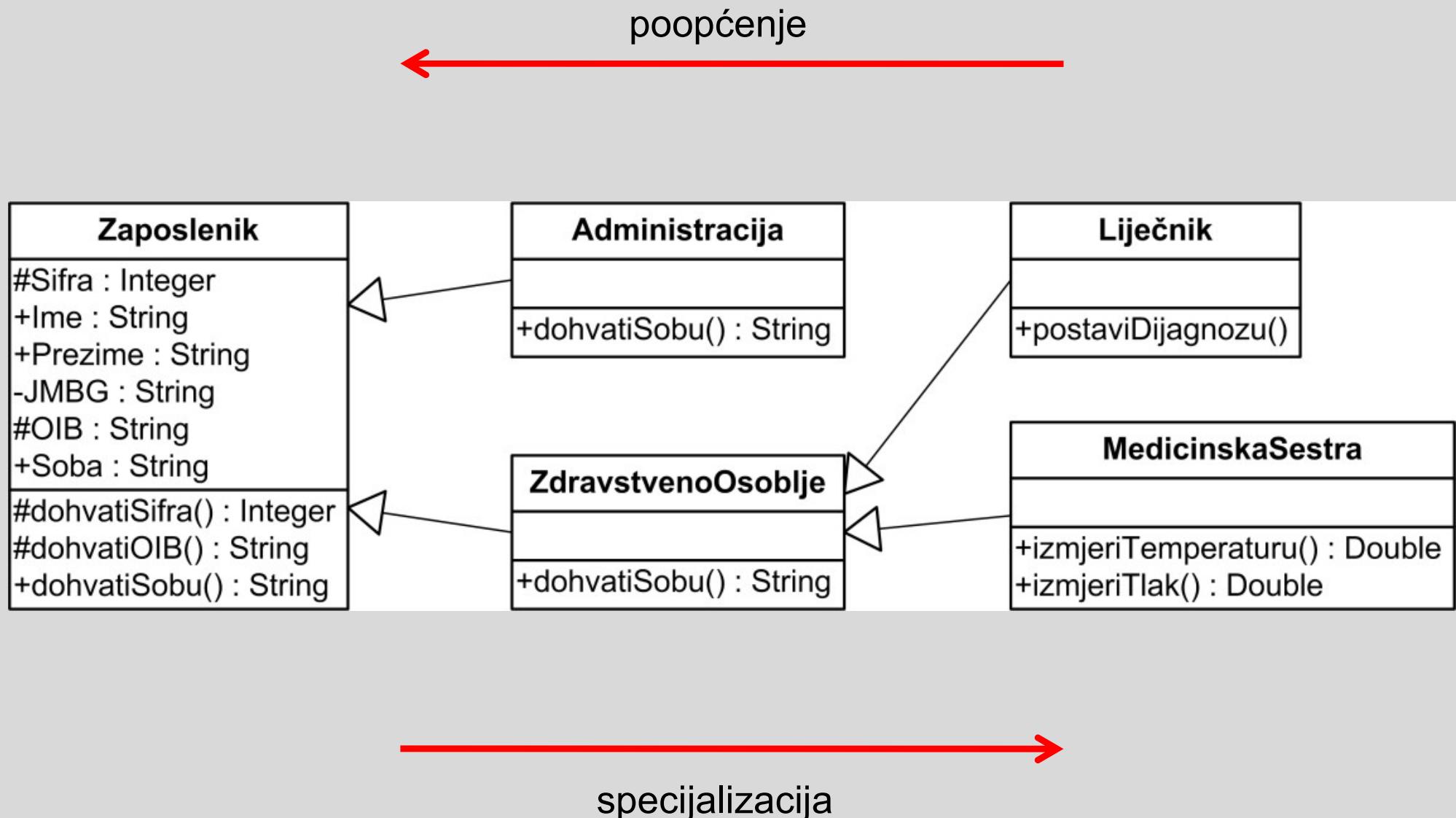
Nasljeđivanje (1)

- **Nasljeđivanje** (engl. *inheritance*) je koncept UML-a u kojemu objekt koji se nasljeđuje je proširen u objektu koji ga nasljeđuje.
- Oblik UML odnosa - podtip.
 - Nasljedna veza između razreda.
 - Jeden razred je *roditelj* (nadrazred) drugome razredu (*dijete* ili *podrazred*).
 - Odnos roditelj-dijete → pravilo JE, podskup-skup, (engl. IS-A).

Nasljeđivanje (2)

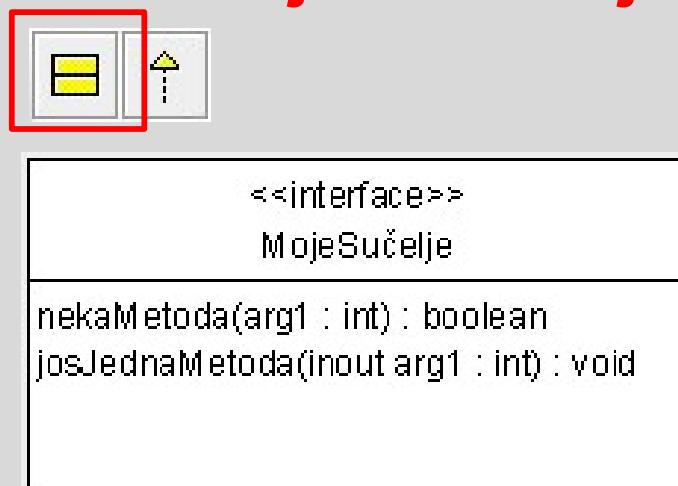
- **Poopćenje** – omogućuje stvaranje nadrazreda koja objedinjuje strukturu i ponašanje zajedničko za nekoliko razreda.
- **Specijalizacija** – omogućuje stvaranje podrazreda koja predstavlja dodavanje novih elemenata.
 - Podrazred uvijek ima više ili jednak broj svojstava u odnosu na nadrazred
 - Podrazred nasljeđuje od nadrazreda atribute, relacije i operacije.
 - Podrazred može biti proširen atributima, operacijama ili relacijama.
 - Podrazred može imati svoju implementaciju operacija koje je naslijedio.

Primjer nasljeđivanja



Sučelje

- **Sučelje** (engl. *interface*) je skup operacija koja specificira usluge nekog razreda.
 - Sučelje definira skup operacijskih specifikacija (tj. njihovih potpisa), ali nikada skup njihovih implementacija.
 - **Sučelje je razred, ali bez atributa i sve operacije imaju samo tijelo, bez implementacije.**



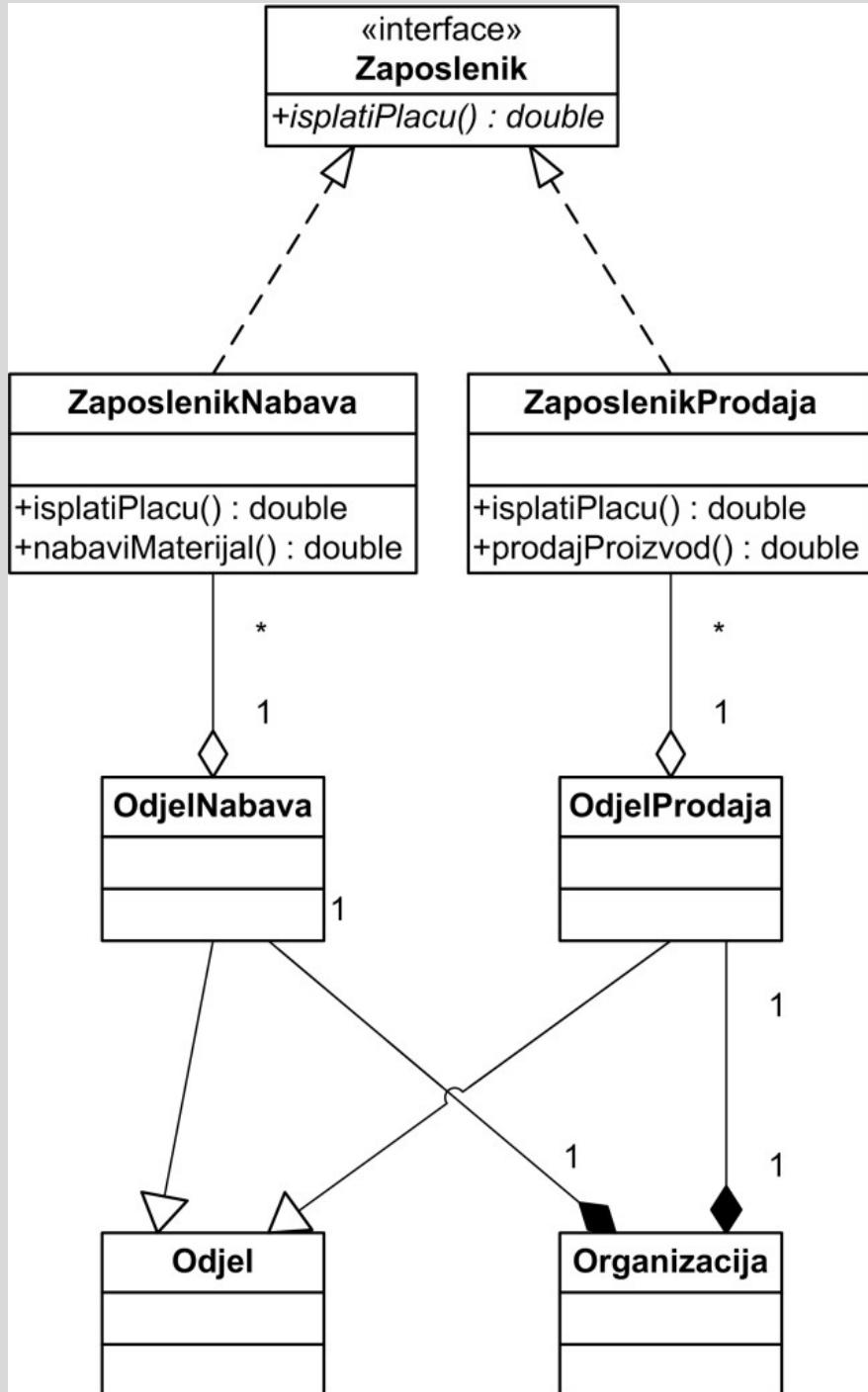
```
public interface MojeSučelje {  
  
    public boolean nekaMetoda(int arg1);  
  
    public void josJednaMetoda(int arg1);  
}
```

Realizacija sučelja



- **Realizacija** (engl. *realization*) je veza UML-a koja označava ostvarenje sučelja.
- **Razred realizira ili ostvaruje sučelje.**
 - Veza realizacije (strelica) je usmjerenja od razreda prema sučelju.
- Realizacija je slična nasljeđivanju, ali u realizaciji nasljeđuju se samo definicije operacija, bez njihove implementacije. S nasljeđivanjem specifičniji razred dobiva sve atribute i operacije općenitijeg ili nadređenog razreda.

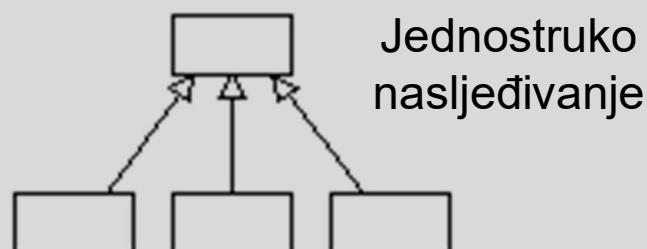
Primjer sučelja i realizacije



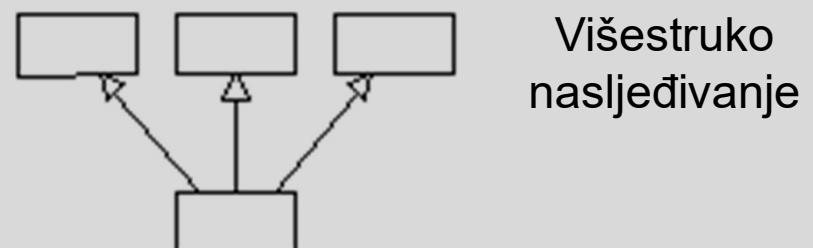
U nekoj organizaciji postoje odjeli Nabave i Prodaje. Odjeli imaju vlastite zaposlenike, koji rade samo u tom odjelu. Svi zaposlenici dobivaju isplate mjesecnih plaća, ali zaposlenici nabave i prodaje imaju drugačiji algoritam izračuna iznosa plaće. Zaposlenici prodaje mogu prodati proizvode organizacije, a zaposlenici Nabave kupiti ulazni materijal potreban za proizvodnju. Operacije nabave i prodaje vraćaju *double* vrijednosti. Zaposlenike je potrebno definirati koristeći zajedničko sučelje.

Višestrukost nasljeđivanja i realizacije

- Višestruko nasljeđivanje (engl. *multiple inheritance*) je zabranjeno u nekim OO programskim jezicima (npr. Java i C#). U C++ je dozvoljeno.
- Višestruka realizacija je uvijek dopuštena.
 - Omogućuje opće višestruko nasljeđivanje (engl. *general multiple inheritance*) i u Javi tako da je moguće implementirati više razreda bez mijenjanja njihove definicije, što je u konačnici slično učinku višestrukog nasljeđivanja.



Jednostruko
nasljeđivanje



Višestruko
nasljeđivanje

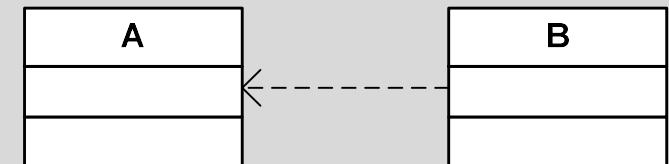
Identificiranje vrste odnosa

- Kako odrediti koja vrsta odnosa između dva razreda je ispravna: **pridruživanje, agregacija, kompozicija ili nasljeđivanje?**
- **Agregacija:** ako jedan razred obuhvaća ili sadržava drugog (povezani su odnosom cjelina-dio).
- **Kompozicija:** ako su razredi u odnosu cjelina-dio, ali nakon uništavanja pojedinaca cjeline moraju se uništiti i pojedinci koji čine dio cjeline.
- **Nasljeđivanje:** ako su razredi u odnosu roditelj-dijete.
- **Pridruživanje:** ako razredi nisu u odnosima cjelina-dio i roditelj-dijete.

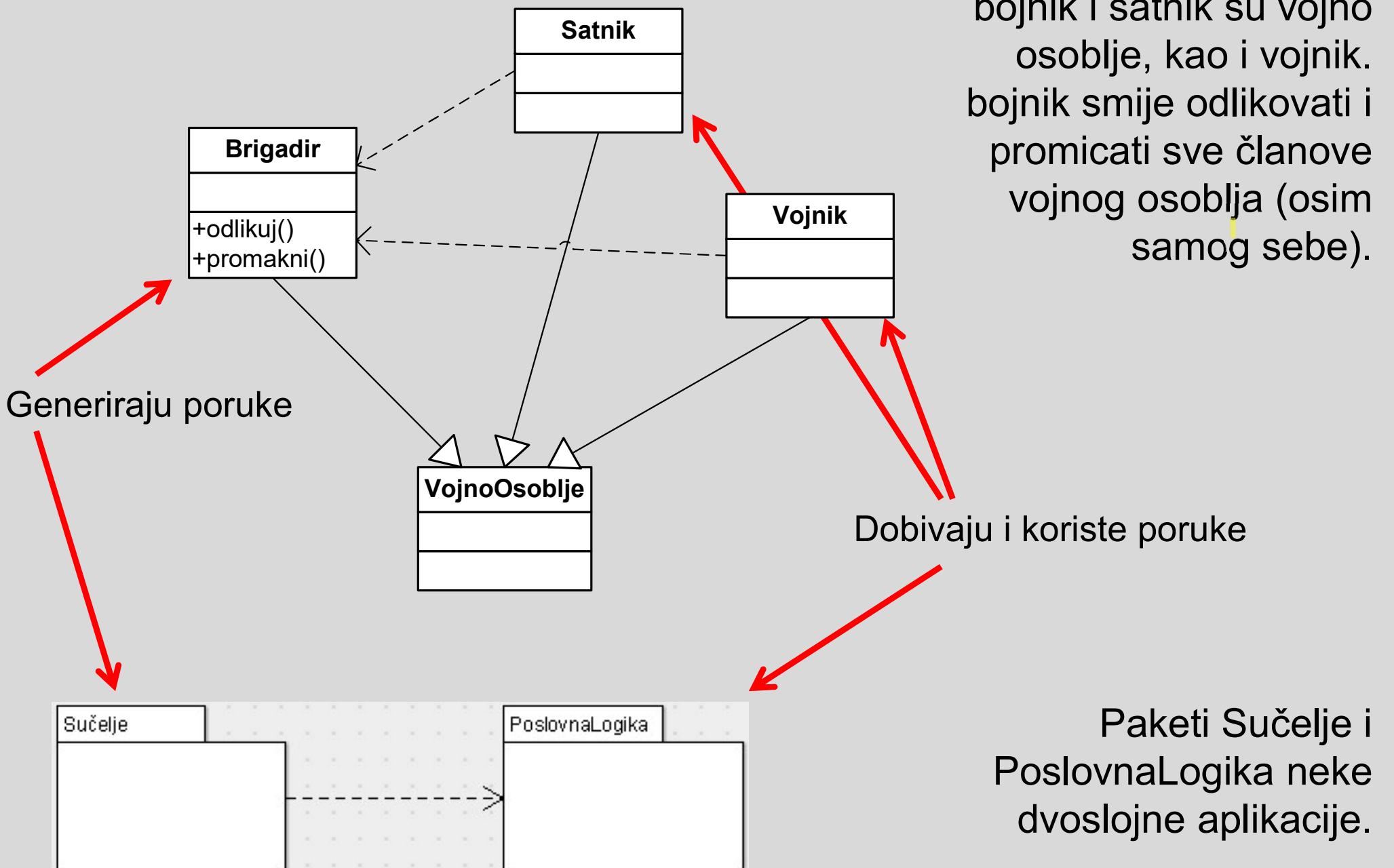
Ovisnost



- **Ovisnost** (engl. *dependency*) pokazuje da jedan razred ili paket dijagrama ovisi o drugome.
 - Semantička relacija između dvije stvari u kojoj promjena u jednoj (neovisnoj stvari) može utjecati na semantiku druge (ovisne stvari).
- Ovisnost je uvijek jednosmjerana.
- Čitamo: “**B ovisi o A**” u smjeru strelice.
 - **A** se naziva isporučitelj (engl. *supplier*) i **B** se naziva klijent (engl. *client*).
- ArgoUML ne preslikava svojstvo ovisnosti u programski kôd.



Primjeri ovisnosti



Odbrojčavanje

- **Odbrojčavanje** (engl. *enumeration*) je oblik tipa podatka koji sadržava uređene parove imenovanih identifikatora i njima pridruženih vrijednosti.
 - Te vrijednosti nazivaju se odbrojčani literali.
- **Koriste se za opis diskretnih vrijednosti.**

«enumeration»
KomisijaOcjena
+JednoglasnoPoložio = 1
+Položio = 2
+NijePoložio = 0

Komentari

- Unatoč formalnoj izražajnosti UML dijagrama razreda ponekada su potrebni i **komentari**.
 - Ne upotrebljavaju se uvijek. Koriste se za dodatni opis svrhe nekog razreda, atributa, veza, operacija i drugih elemenata dijagrama.
 - U komentarima je poželjno biti **jasan i sažet**, te **obuhvatiti sve bitne** aspekte UML elementa koji se opisuje, a koji nisu nedvosmisleno jasni iz samog dijagrama.
 - Komentari mogu biti povezani s konkretnim elementom dijagrama, ili se mogu odnositi na cijeli dijagram. Specifični komentari povezani su s elementom neoznačenom vezom, a komentari o cijelom dijagramu nemaju takvih veze i nalaze se na rubu crteža obično u jednom od uglova dijagrama.

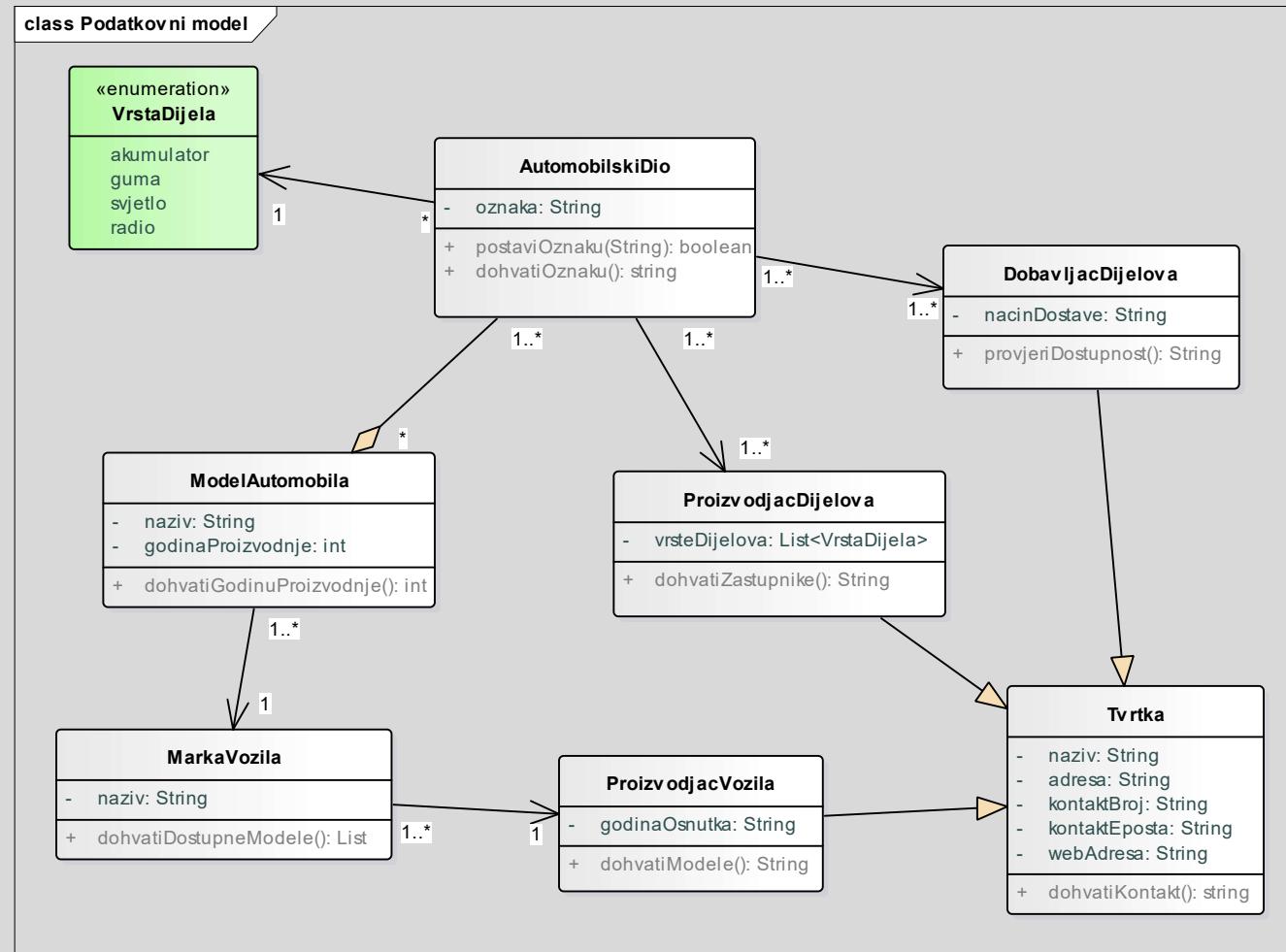
Laboratorijske vježbe

UML DIJAGRAMI RAZREDA U DOKUMENTU ZAHTJEVA

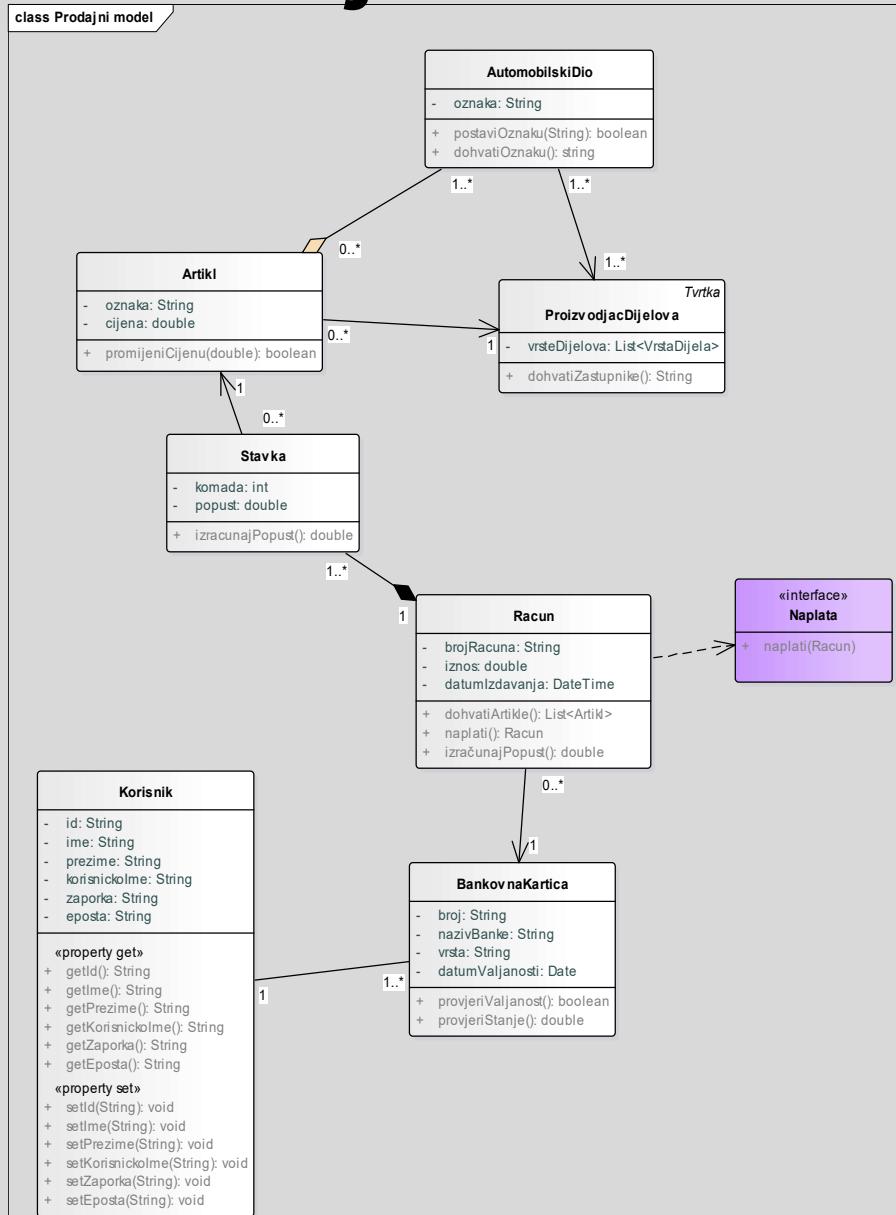
Dijagram razreda

- Opis arhitekture:
 - Opis domene
 - Opis podatkovnog modela
- Poslije, u dokumentu ostvarenja:
 - opis ostvarenja

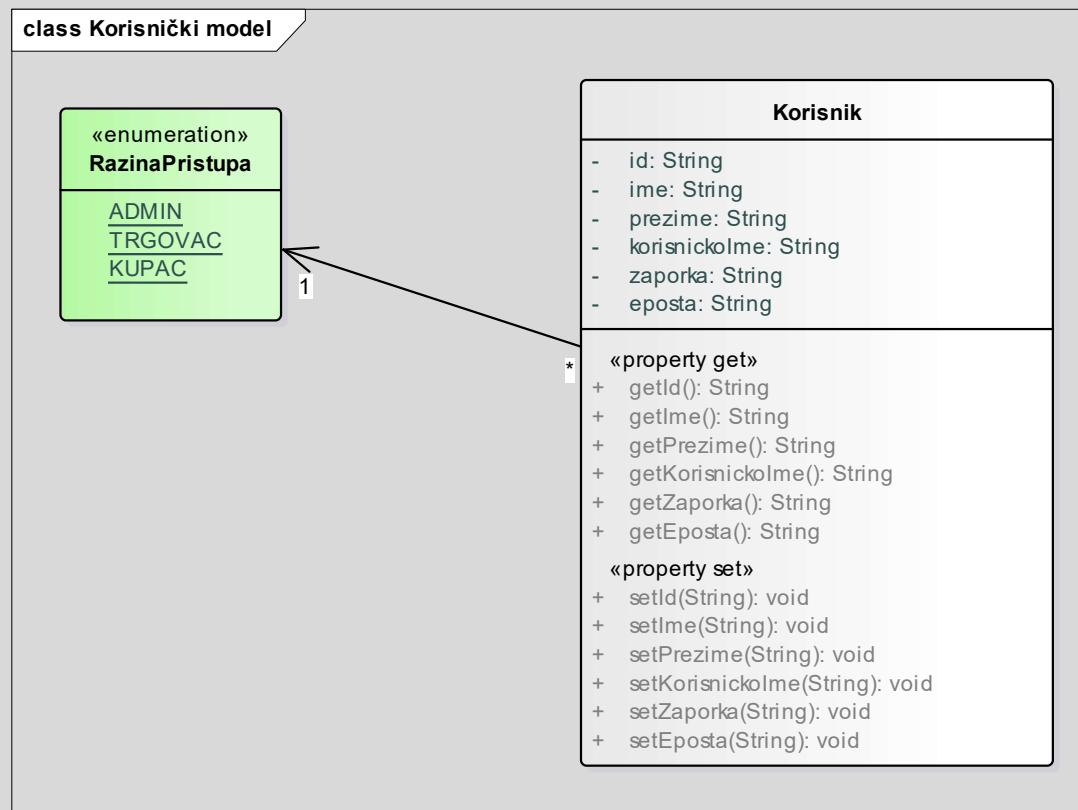
Podatkovni model prodaje autodijelova



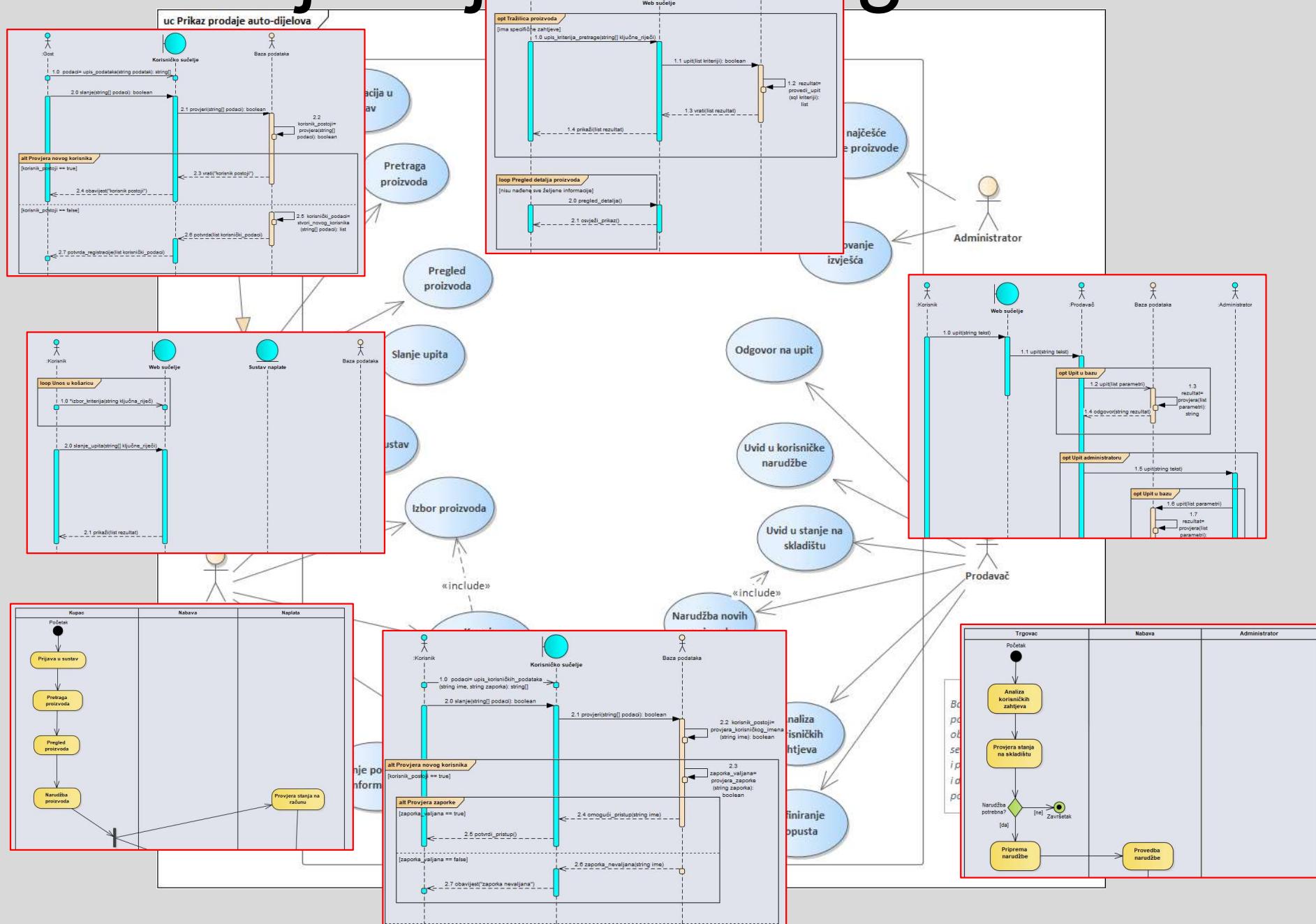
Podatkovni model prodaje autodijelova



Podatkovni model prodaje autodijelova



Primjer cjeleokupnog sustava



Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

Nekoliko savjeta

- Dijagrami razreda su dio gotovo svih objektno-orientiranih (OO) paradigmi i **koriste se veoma često.**
- **Mogu biti bogati informacijama i stoga teško čitljivi**, stoga nekoliko savjeta:
 - Ne koristite odmah sve moguće notacije. Počnite s jednostavnim dijagramom i postepeno dodajte detalje.
 - Razlikujte različite poglede na sustav.
 - U praksi ne morate modelirati baš sve. Koncentrirajte se na bitne segmente. Bolje je imati nekoliko kvalitetnih dijagrama po dijelovima sustava, nego zastarjeli dijagram cijelog sustava.

Različita gledišta na dijagram

- Prema razini detalja → u različitim razvojnim fazama projekta:
 - konceptualna
 - specifikacijska
 - implementacijska

Različita gledišta na dijagram

- **koncepcionalna:**
 - opis stvari iz stvarnog svijeta
 - prikaz koncepata iz domene sustava → odgovaraju razredima
 - **jezično neovisni prikaz**

Različita gledišta na dijagram

- **specifikacijska:**
 - apstrakcije i komponente sa specifikacijom i sučeljima
 - nema čvrste obveze prema stvarnoj implementaciji
 - **pogled na razini sučelja**, ne implementacije

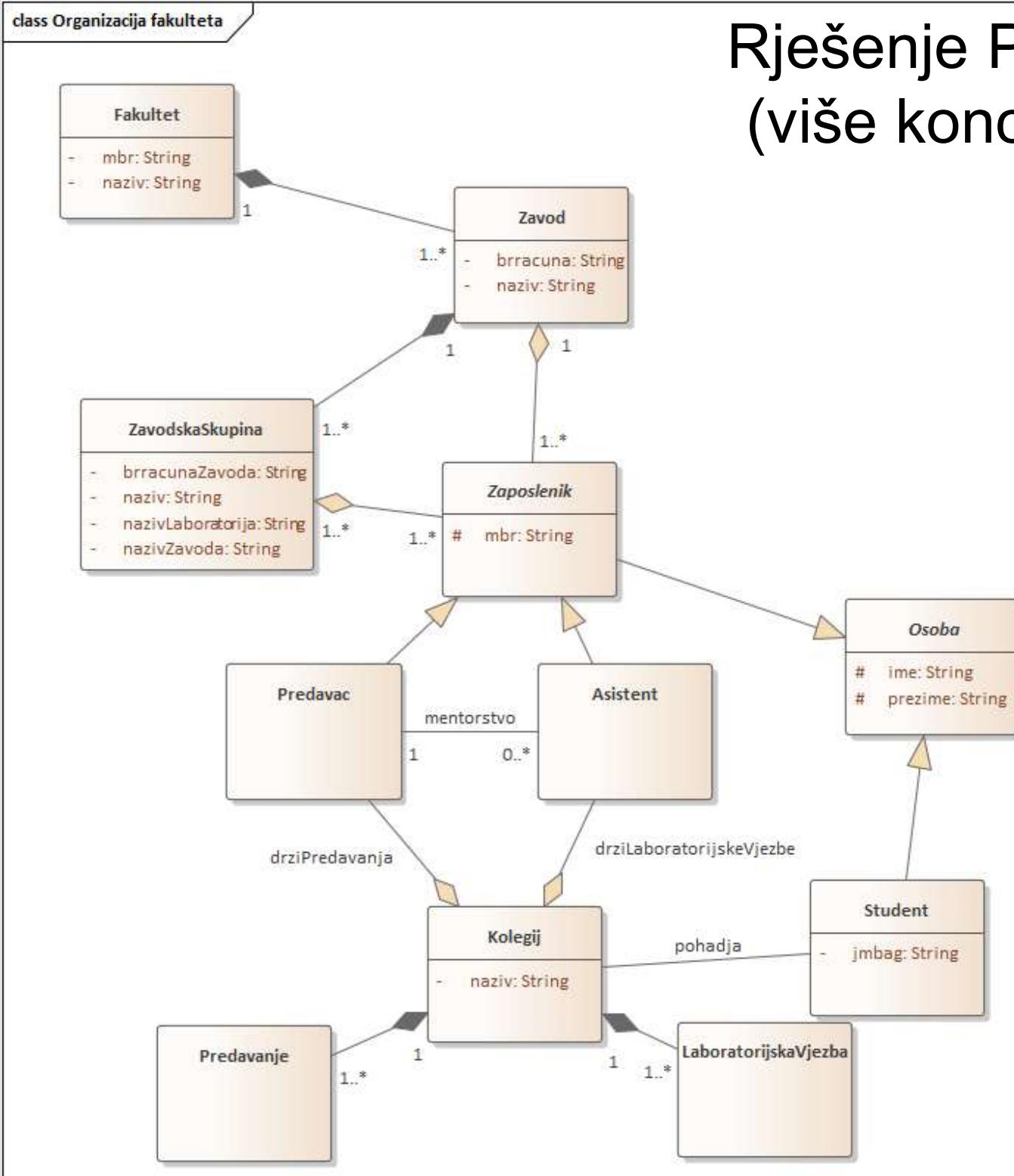
Različita gledišta na dijagram

- **implementacijska:**
 - opis na razini konkretnе tehnologije i jezika
 - **pogled na razini implementacije**

Primjer 1: Modeliranje organizacije fakulteta

Neki fakultet sastoji se od jednog ili više zavoda, a svaki zavod od jedne ili više zavodskih skupina. Zavodsku skupinu čine zaposlenici. Zaposlenici mogu raditi i u nekoliko zavodskih skupina, ali ne mogu raditi na više zavoda. Postoje dva konkretna tipa zaposlenika: predavači i asistenti. Svaki predavač ima barem jedan kolegij koji predaje, a svaki asistent drži laboratorijske vježbe iz barem jednog kolegija. Svaki kolegij može imati jednog ili više predavača i asistenata. Asistent ima jednog predavača u funkciji mentora, a predavač može imati više asistenata. Svaki kolegij se sastoji od više predavanja i više laboratorijskih vježbi i ima svoj naziv (String). Uklanjanjem kolegija ukidaju se predavanja i laboratorijske vježbe, ali naravno, ne otpuštaju se zaposlenici koji kolegij drže. Student je zasebna kategorija u organizaciji fakulteta i u ovom modelu prepostavite samo da sluša jedan ili više kolegija. I student i zaposlenik su osobe. Svaka osoba ima svoje ime i prezime. Dodatno, svaki zaposlenik ima svoj matični broj zaposlenika (String), a svaki student svoj JMBAG (String). Fakultet ima svoj matični broj (String) i naziv (String). Zavod ima svoj naziv (String) i broj računa (String). Naziv i broj računa zavoda nasljeđuju i zavodske skupine, s tim da one osim toga imaju i svoj naziv skupine te dodatno, naziv glavnog laboratorija (String).

Rješenje Primjera #1 (više konceptualno)



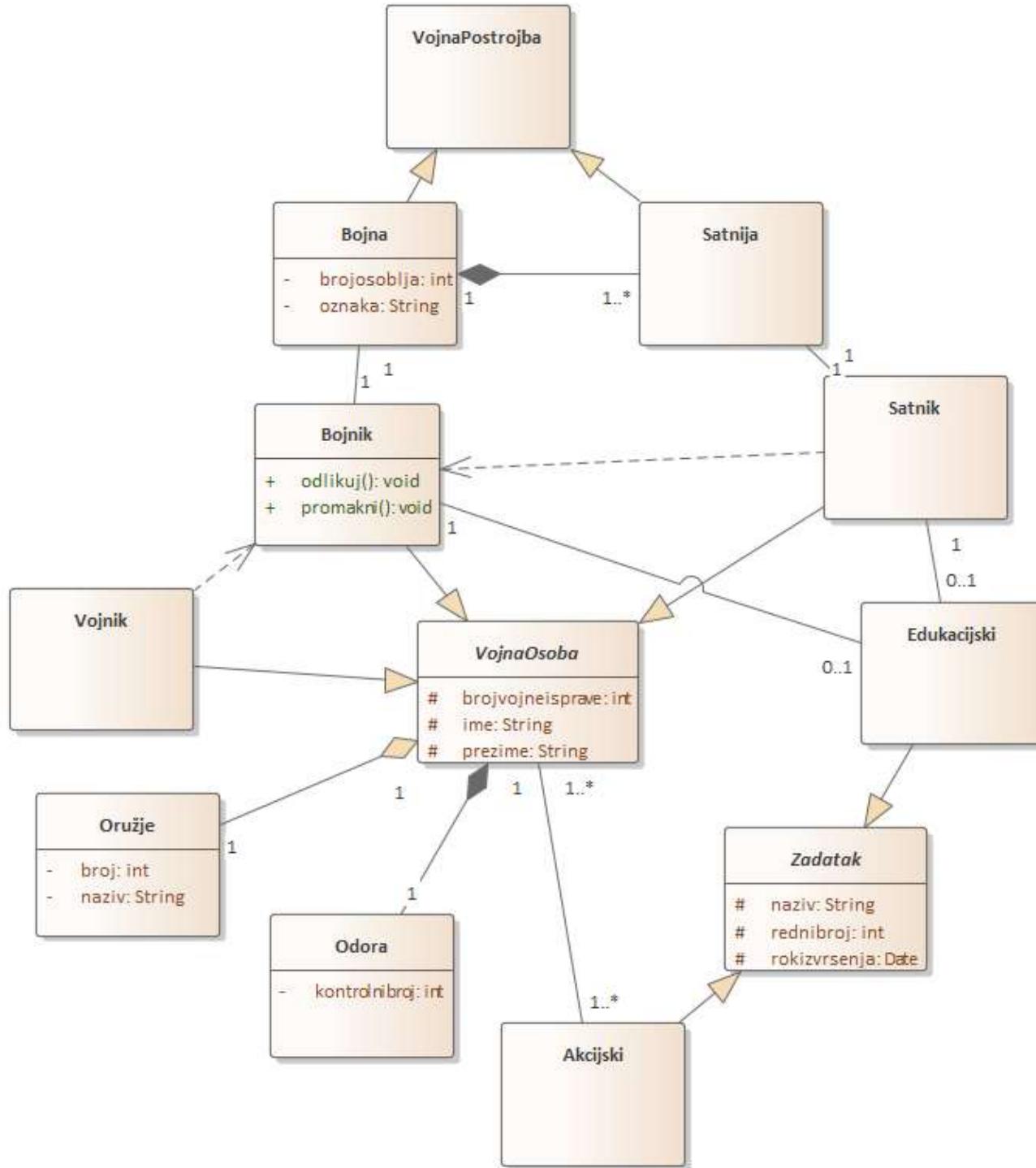
Komentar rješenja Primjera #1

- Veza *pohadja* može biti agregacija između razreda *Student* i *Kolegij*.
- Veza *drziLaboratorijskeVjezbe* može biti jednosmjerna umjesto agregacije, kao i veza *drziPredavanja*
- Nije potrebno napraviti dodatnu vezu između razreda *Asistent* i *LaboratorijskeVjezbe* jer ovakav sustav omogućuje povezivanje ta dva entiteta.

Primjer 2: Modeliranje organizacije vojne postrojbe

Prepostavite da neka vojna postrojba može biti bojna ili satnija. Svaka bojna sadrži jednu ili više satnija. Na čelu bojne nalazi se bojnik, a na čelu satnije satnik. Bojnik i satnik su vojno osoblje, kao i vojnik. Bojnik smije odlikovati i promicati sve članove vojnog osoblja (osim samog sebe). Svaki član vojnog osoblja ima svoje zadatke. Zadatak ima svoj redni broj (int), naziv (String) i rok izvršenja (Date). Postoje dva tipa zadataka: edukacijski i akcijski. Edukacijske zadatke smiju obavljati samo bojnik i satnik. Oni mogu imati najviše jedan edukacijski zadatak. Svaki edukacijski zadatak drži samo jedan bojnik ili satnik, ali jedan edukacijski zadatak može istodobno imati bojnika i satnika. Svaki član vojnog osoblja može imati jedan ili više akcijskih poslova, a jedan akcijski posao može obavljati više različitih članova vojnog osoblja. Svaki član vojnog osoblja nosi po jedan komad oružja i vojnu odoru. Vojna odora je prilagođena svakom pojedinom članu vojnog osoblja i ako iz bilo kojeg razloga član vojnog osoblja napusti vojnu postrojbu, vojna odora se uništava. Oružje nosi svaki član vojnog osoblja, ali ono ostaje na raspolaganju čak i ako pojedinac napusti vojnu postrojbu. Svaki član vojnog osoblja ima svoje ime i prezime (String) i broj vojne isprave (int). Svaka bojna ima svoju oznaku (String) i broj vojnog osoblja (int). Svaki komad oružja ima svoj broj (int) i naziv (String). Svaka odora ima svoj kontrolni broj (int).

Rje



Komentar rješenja Primjera #2

- Veza razreda Bojnik i Zadatak, te Satnik i Zadatak?

REFERENCE I LITERATURA

- Sveučilišna zbirka zadataka iz UML-a - A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014
- Allen Holub's UML Quick Reference:
<http://www.holub.com/goodies/uml>
- Booch G., Jacobson I., Rumbaugh J. "UML Distilled"

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

OBJEKTNO ORIJENTIRANI RAZVOJ PROGRAMA

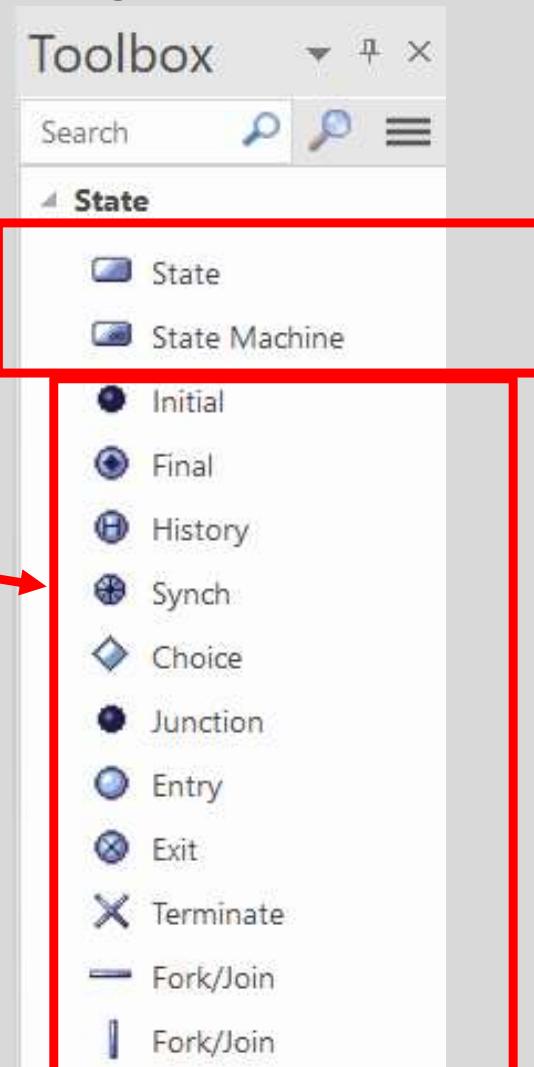
UML DIJAGRAMI STANJA

UML dijagram stanja

- engl. UML State Machine Diagram
- Prikaz sustava ili objekata u sustavu u smislu stanja i prijelaza između stanja u kojem se mogu naći
 - uvjetovano vanjskim događajima
- Promjene stanja pojedinog objekta tijekom njegovog životnog vijeka
- Ne prikazuje aktore niti vanjsko sučelje prema korisnicima.
- Ponašajni dinamički UML dijagram.

Stanja i pseudostanja

- Stanja
- Pseudostanja



Stanje

- Stanje objekta \equiv vrijednost jednog ili više atributa objekta
 - jedno početno stanje, 0 ili više krajnjih stanja
- Aktivnosti unutar stanja
 - *entry* – akcija pri ulasku u stanje
 - *do* – aktivnost koja se izvodi sve dok je stanje aktivno
 - *exit* – akcija pri izlasku iz stanja
 - dodatni događaji i akcije koje su poticanje njima



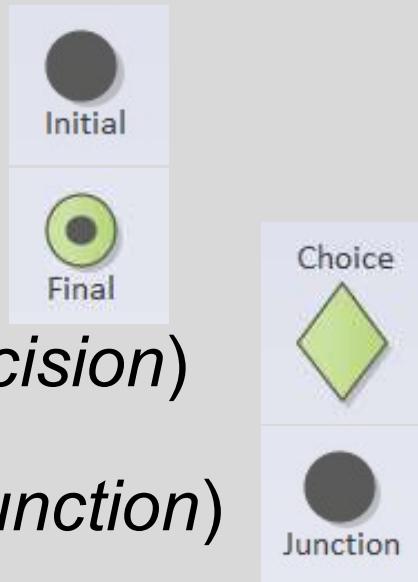
Prijelazi

- Sintaksa: [uvjet] događaj/akcija
- Dozvoljene promjene stanja iz trenutnog u novo stanje (može biti to isto stanje).
- Potaknuti događajima (interakcija, vrijeme), uz zadovoljenje uvjeta (ako postoji) → „KADA se nešto dogodi, AKO je zadovoljen uvjet”



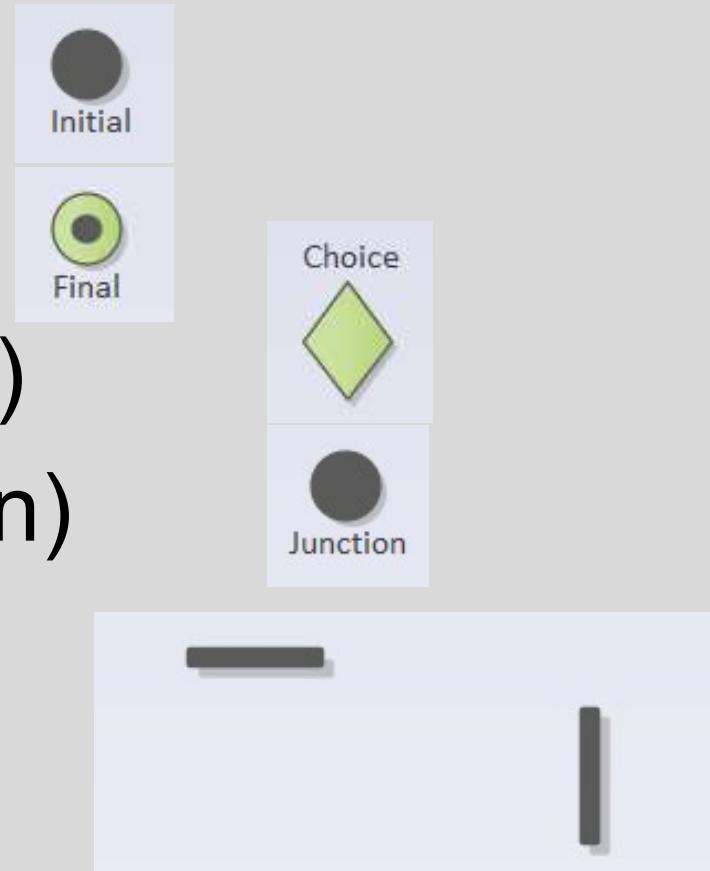
Pseudostanja

- Početno stanje
- Završno stanje
- Odluka (engl. *decision*)
- Spajanje (engl. *junction*)
- Račvanje (engl. *fork*)
- Skupljanje (engl. *join*)



Pseudostanja

- Početno stanje
- Završno stanje
- Odluka (engl. decision)
- Spajanje (engl. junction)
- Račvanje (engl. fork)
- Skupljanje (engl. join)

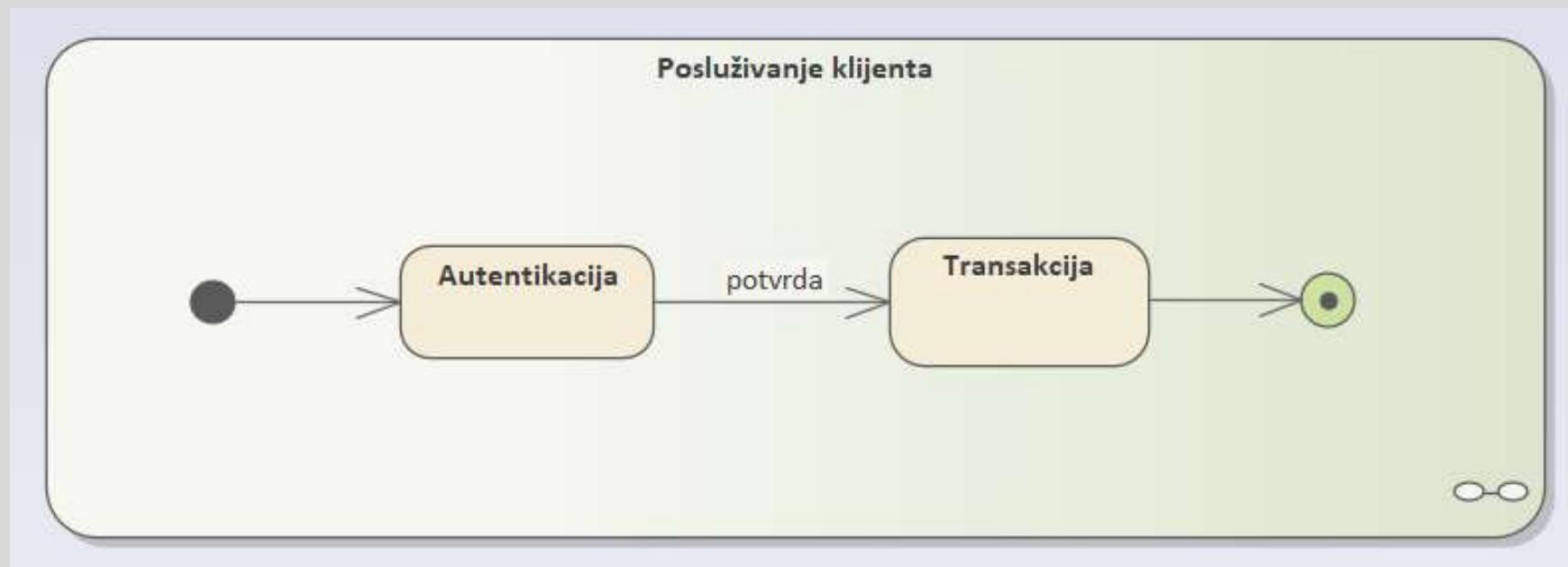


Složena stanja

- engl. *composite state*
 - jednostavno
 - ortogonalno
- engl. *submachine state*
 - cijeli stroj stanja unutar stanja („ugniježdeni“)

Jednostavno složeno stanje

- Sadrži podstanja unutar jednog područja:



Razlika u odnosu na dijagram aktivnosti

- Oba su ponašajni dijagrami, ali s različitim naglascima
- Imaju donekle sličnu notaciju
- Dijagram aktivnosti → tok funkcija (aktivnosti) koje nisu izričito poticane izvana
- Dijagram stanja → stanja poticana događajima

Laboratorijske vježbe

UML DIJAGRAMI STANJA U DOKUMENTU OSTVARENJA

U odnosu na dijagram aktivnosti

- **Dijagram aktivnosti:**
 - Kupnja proizvoda
 - Narudžba novih proizvoda
- **Dijagram stanja:**
 - Stanje košarice
 - Promjene web sučelja pri kupnji
 - Stanje narudžbenice
 - Promjene web sučelja pri narudžbi

Projektna dokumentacija

- moraju biti zastupljeni svi UML dijagrami o kojima se govorilo na predavanjima:
 - jedan ili više dijagrama obrazaca uporabe kojima se prikazuje cijeli sustav
 - najmanje dva dijagrama aktivnosti (ili jedan dijagram stanja) i koliko god je potrebno dijagrama slijeda za opis svih obrazaca uporabe
 - jedan ili više dijagrama razreda
 - jedan ili više dijagrama komponenti
 - dijagram razmještaja

REFERENCE I LITERATURA

- A. Jović, M. Horvat, I. Grudenić, "UML-dijagrami, zbirka primjera i riješenih zadataka", 2014., dostupno u Skriptarnici i knjižnici Fakulteta elektrotehnike i računarstva, te Nacionalnoj i sveučilišnoj knjižnici u Zagrebu
- <https://www.uml-diagrams.org>

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19970

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Ak. god. 2021./22.
Ljetni semestar

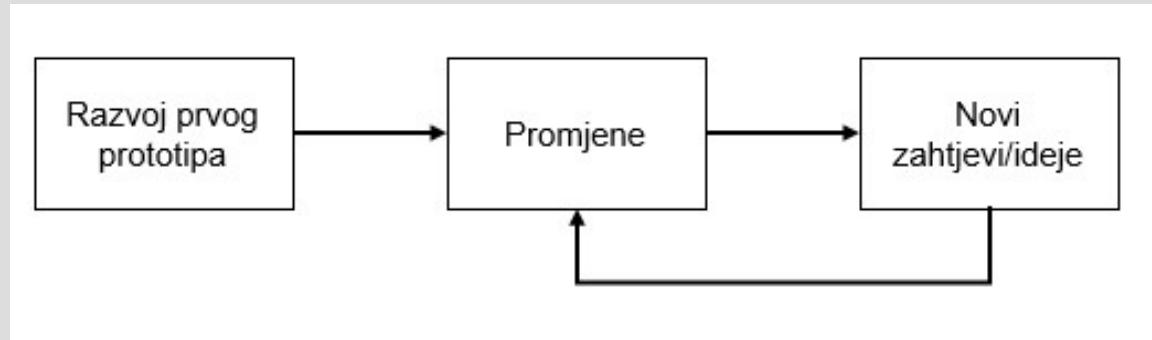
Procesi i modeli procesa programskog inženjerstva

Objektno orijentirani razvoj
programa

Modeli procesa programskog inženjerstva

- dobre prakse pristupa izradi programske potpore
- iskustva razvojnih timova kao skup pozitivnih zaključaka o važnosti i redoslijedu izvođenja projektnih aktivnosti

ad-hoc ili oportunistički pristup



ad-hoc ili oportunistički pristup

- Nema razrade zahtjeva i oblikovanja prije same implementacije.
- Dolazi do brže istrošenosti.
- S obzirom na neplanski razvoj, cilj nikad nije dovoljno jasan.
- Proces ispitivanja ni bilo koji oblik potvrde očekivane kvalitete proizvoda nisu unaprijed ni izričito izdvojeni
- trošak razvoja i održavanja je vrlo visok.

Generički modeli programskog inženjerstva

- **vodopadni model** (engl. *waterfall model*)
 - temeljne aktivnosti procesa izrade programske potpore smatraju se nezavisnim fazama razvoja
- **evolucijski model** (engl. *evolutionary model*)
 - sustav se razvija kroz niz inačica ili inkremenata od kod kojih svaki sljedeći inkrement dodaje neku novu funkcionalnost u onu na koju se nastavlja
- **komponentni model** (engl. *component-based model*)
 - motivacija je u prepostavci ponovne iskoristivosti postojećih komponenata.

Zasebni modeli programskog inženjerstva

- **Modelno-usmjereni razvoj**
 - temeljen na oblikovanju uporabom modela (engl. *model-based design, model-based software engineering*)
 - → **unificirani proces** (engl. *Unified Process, UP*)
- **agilni razvoj** (engl. *agile development*)

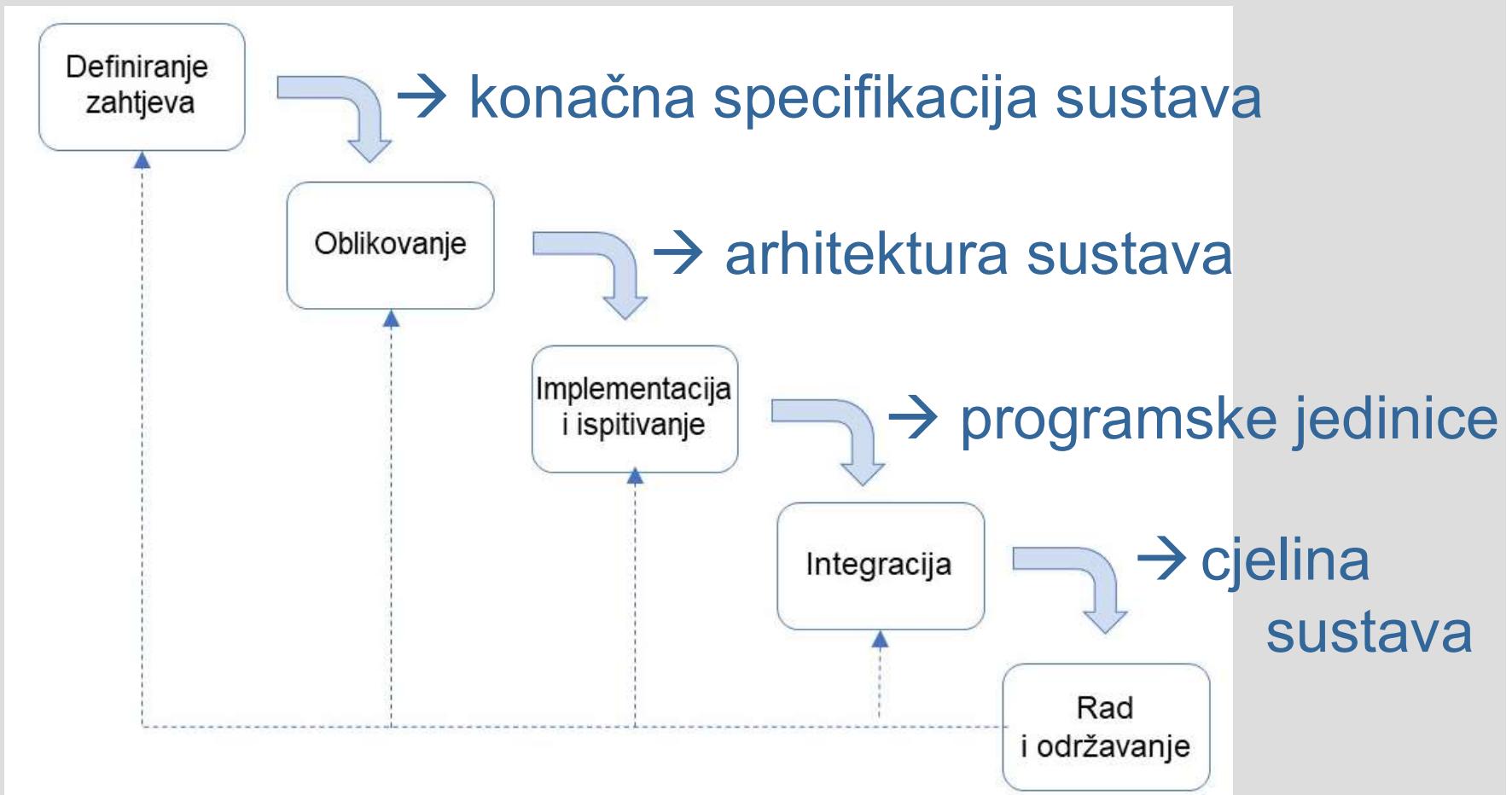
Procesi i modeli procesa programskog inženjerstva

VODOPADNI MODEL

Vodopadni model programskog inženjerstva

- procesne faze su jasno odvojene i u načelu nezavisne:
 - izlučivanja zahtjeva
 - oblikovanja
 - implementacije
 - integracije
 - održavanja sustava.
- dobro definiran plan rada za svaku od njih
- Sljedeća faza započinje tek kada je prethodna faza završena.

Vodopadni model programskog inženjerstva



Vodopadni model programskog inženjerstva

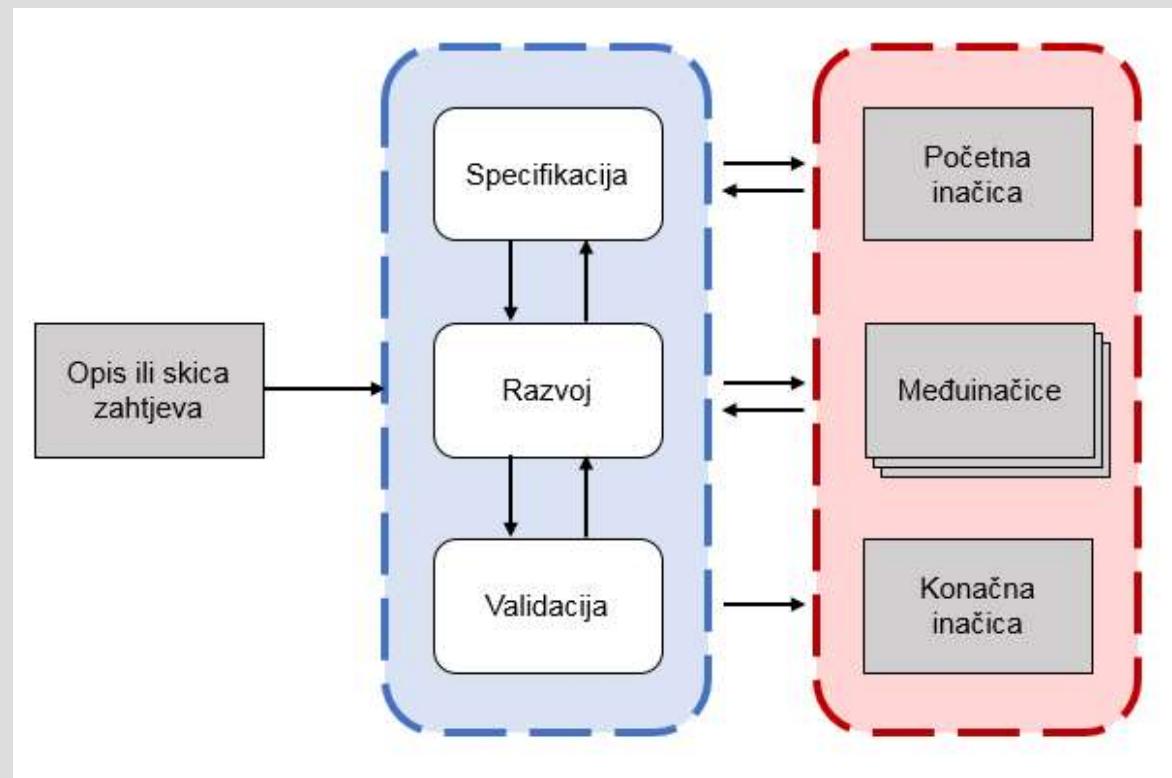
- strogost modela
- nezavisnost pojedinih faza
- ako su razumljivi zahtjevi s malom vjerojatnošću promjene
- razvoj na nekoliko odvojenih mesta
- ako postoji formalna specifikacija sustava

Procesi i modeli procesa programskog inženjerstva

EVOLUCIJSKI MODEL

Evolucijski model

- ispreplitanje faza:
(ili projektnih aktivnosti?)



Evolucijski model

- Dva tipa:
 - istraživački razvoj i oblikovanje (engl. *exploratory development*)
 - metoda odbacivanja prototipa (engl. *throw-away prototyping*)
- „Slabije“ definirani zahtjevi:
 - razvoj započinje u dijelovima sustava za koje su zahtjevi dobro definirani
 - izrada prototipova sustava koji se ne koriste u konačnici

Evolucijski model

- Prednosti:
 - brži nego vodopadni model
 - osnovica agilnog pristupa
 - brzi razvoj inkremenata
 - inkrementalni razvoj specifikacije (*)

Evolucijski model

- Nedostaci:
 - inkrementalni razvoj specifikacije (*)
 - proces razvoja i oblikovanja nije jasno vidljiv
 - zbog čestih izmjena narušeni:
 - struktura sustava
 - kvantifikacija napretka
 - dokumentiranje izdanja/inačica

Procesi i modeli procesa programskog inženjerstva

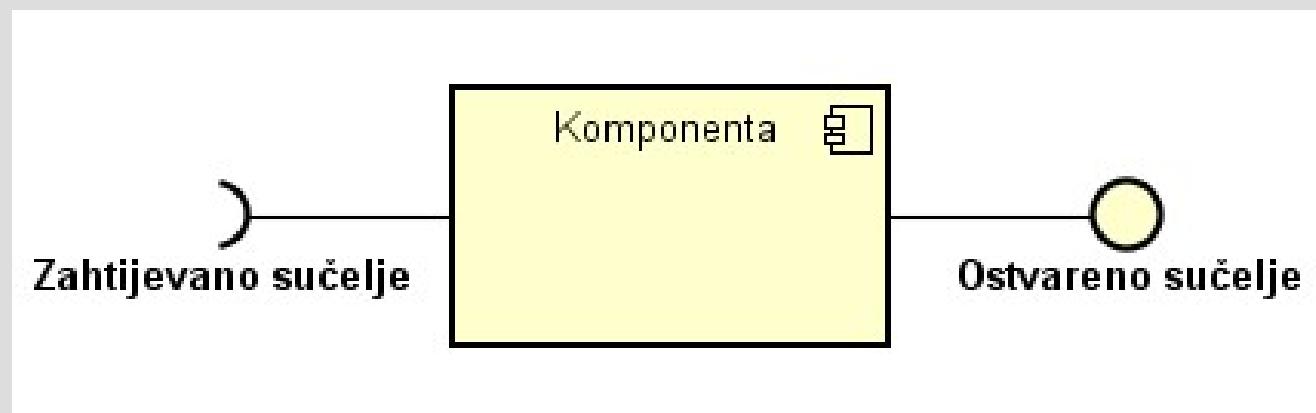
KOMPONENTNO-USMJERENI MODEL

Komponentno-usmjereni model

- Komponenta je nezavisna jedinica programske potpore s određenom funkcionalnosti koja se može kombinirati s drugim nezavisnim jedinicama u svrhu izrade cjelovitog sustava programske potpore.
- Komponenta kao „crna kutija“:
 - lokacija i tehnologija ostvarenja nebitni
 - identifikacija sučeljem

Komponentno-usmjereni model

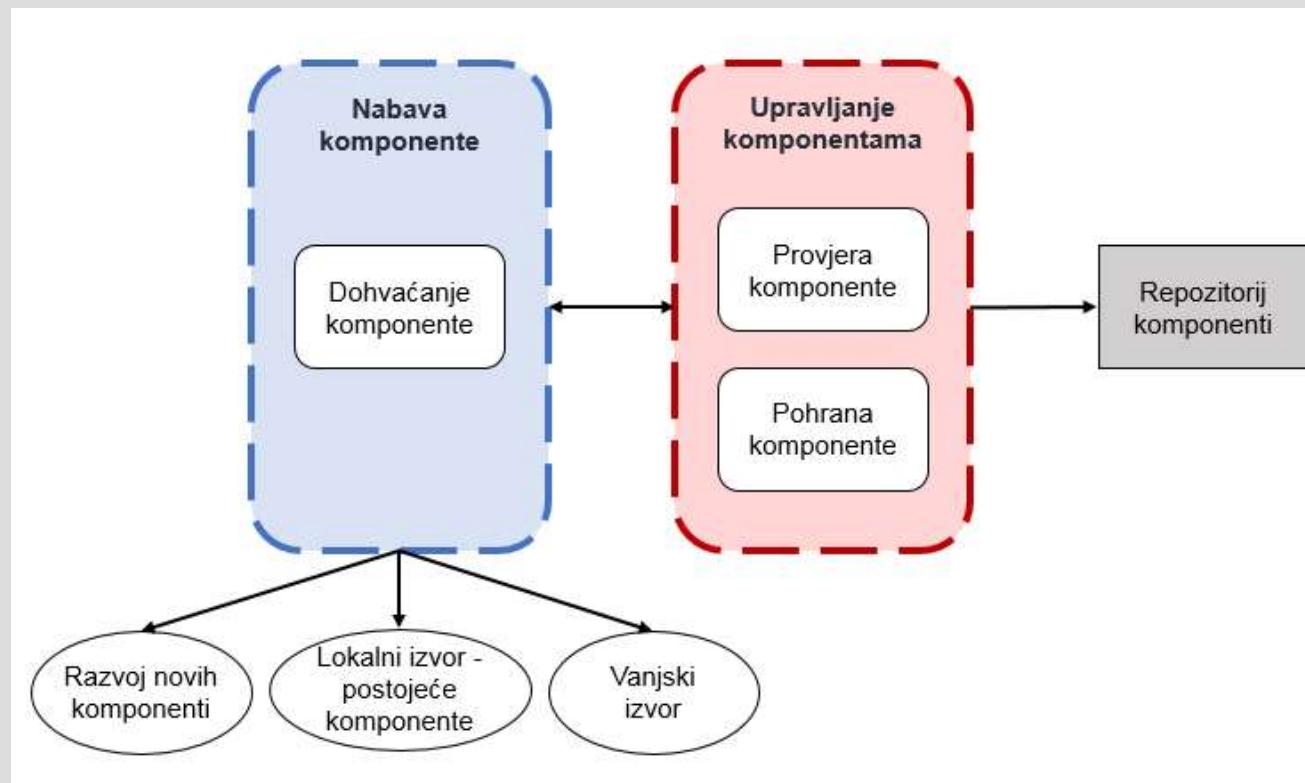
- Dva su temeljna tipa sučelja komponente:
 - **ponuđeno** ili izvezeno sučelje (engl. *exported interface*)
 - **zahtijevano** ili uvezeno sučelje (engl. *imported interface*)



Komponentno-usmjereni razvoj

- Dva različita stajališta:
 - razvoj za ponovnu iskoristivost (engl. *development for reuse*)
 - razvoj s korištenjem postojećih komponenti (engl. *development with reuse*)

Komponentno-usmjereni model



Komponentno-usmjereni model

- Zadana norma komponente:
 - definicija sučelja
 - uputa za korištenje
 - uputa za isporuku

Komponentno-usmjereni model

- ponovno iskorištavanje postojećeg kôda
se događa često i neformalno
- razlučivosti jedinica programske potpore
(u praksi):
 - objekti
 - komponente aplikacije
 - gotova aplikacija

Komponentno-usmjereni model

- Razvoj temeljen na ponovnoj iskoristivosti komponenti:



Prednosti komponentno-usmјerenog modela

- povećana je pouzdanost komponente
- olakšana je, bolja i jasnija procjena troška
- specijalizirani razvoj: bolje poklapanje sa zadanim normama izrade komponente
- značajno ubrzanje u razvoju sustava

Mane komponentno- usmjerenog modela

- nedostupan izvorni programski kôd, troškovi održavanja komponente rastu ako postane nekompatibilna s promjenama u sustavu
- korištenje alata programske potpore koji ne predviđaju ili ne podržavaju koncept ponovne uporabnosti komponenti
- visoki troškovi održavanja knjižnice komponenata mogu biti visoki
- napor pronalaženja, razumijevanja i prilagodbe komponenata

Procesi i modeli procesa programskog inženjerstva

UNIFICIRANI PROCES (UP)

Unificirani proces

- Unificirani proces (engl. *Unified Process*, UP) je metodologija razvoja programske potpore koja se temelji na oblikovanju pomoću modela (engl. *Model Based Design*, MBD), odnosno iterativnom razvoju, obrascima uporabe i usmjerenjem na arhitekturu sustava.

Unificirani proces

- Faze životnog ciklusa:
 - početak (engl. *inception*)
 - elaboracija (engl. *elaboration*)
 - izgradnja (engl. *construction*)
 - prijenos proizvoda korisnicima (engl. *transition*)

Unificirani proces

- Faze životnog ciklusa:
 - početak (engl. *inception*)
→ vizija ili ciljevi životnog ciklusa (engl. *lifecycle objectives*)
 - elaboracija (engl. *elaboration*)
→ temeljna arhitektura (engl. *lifecycle architecture*)
 - izgradnja (engl. *construction*)
→ početna sposobnost (engl. *initial operational capability*)
 - prijenos proizvoda korisnicima (engl. *transition*)
→ izdavanje izvršne inačice programa ili proizvoda (engl. *release*)

Procesi i modeli procesa programskog inženjerstva

AGILNI PRISTUP RAZVOJU PROGRAMSKE POTPORE

Agilni pristup

- Agilni pristup razvoju programske potpore podrazumijeva skupinu metoda za razvoj programske potpore kojima je zajednički iterativni razvoj uz male inkremente i brz odziv na korisničke zahtjeve. Ovaj model razvoja programske potpore koristi se za razvoj manjih i srednjih projekata u stalnoj interakciji s klijentima putem stalnog predočavanja novih poboljšanja, uz relativno slabo dokumentiranje.

Manifesto for Agile Software Development

- *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

Manifesto for Agile Software Development

- ***Individuals and interactions*** over processes and tools
- ***Working software*** over comprehensive documentation
- ***Customer collaboration*** over contract negotiation
- ***Responding to change*** over following a plan
- *That is, while there is value in the items on the right, we value the items on the left more.“*

12 temeljnih načela agilnog razvoja

- Najvažnije nam je zadovoljstvo naručitelja, koje postižemo ranom i neprekinutom isporukom programskog proizvoda koji donosi vrijednost.
- Spremno prihvaćamo promjene zahtjeva, čak i u kasnoj fazi razvoja. Agilni procesi koriste promjene da naručitelju stvore kompetitivnu prednost.
- Često isporučujemo upotrebljiv programski proizvod, u razmacima od nekoliko tjedana do nekoliko mjeseci, nastojeći da razmak bude čim kraći.
- Poslovni ljudi i razvojni inženjeri moraju svakodnevno zajedno raditi, tijekom cijelokupnog trajanja projekta.

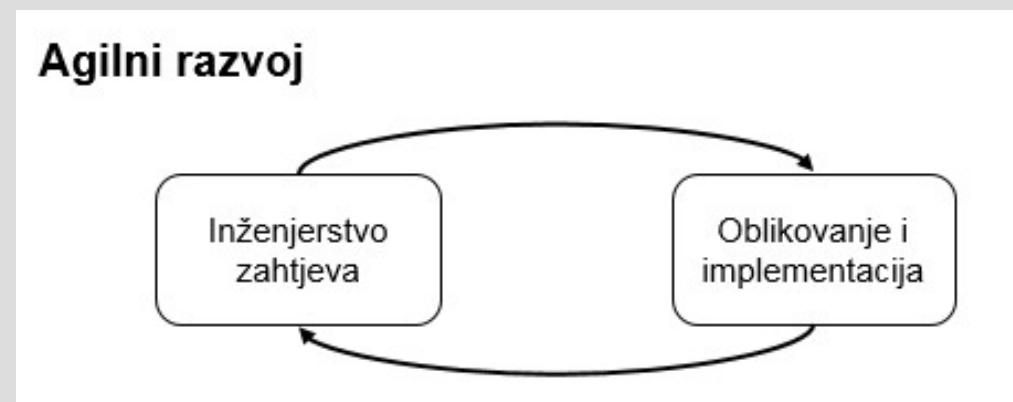
12 temeljnih načela agilnog razvoja

- Projekte ostvarujemo oslanjajući se na motivirane pojedince. Pružamo im okruženje i podršku koja im je potrebna, i prepuštamo im posao s povjerenjem.
- Razgovor uživo je najučinkovitiji način prijenosa informacija razvojnog timu i unutar tima.
- Upotrebljiv programski proizvod je osnovno mjerilo napretka.
- Agilni procesi potiču i podržavaju održivi razvoj. Pokrovitelji, razvojni inženjeri i korisnici trebali bi moći neograničeno dugo zadržati jednak tempo rada.
- Neprekinuti naglasak na tehničkoj izvrsnosti i dobrom oblikovanju pospješuju agilnost.

12 temeljnih načela agilnog razvoja

- Jednostavnost – vještina povećanja količine posla koji ne treba raditi – je od suštinske važnosti.
- Najbolje arhitekture, projektne zahtjeve i oblikovanje programske potpore stvaraju samo-organizirajući timovi.
- Tim u redovitim razmacima razmatra načine kako da postane učinkovitiji i zatim usklađuje i prilagođava svoje ponašanje u tom smjeru.

Model procesa agilnog razvoja



Metode/modeli agilnog razvoja

- Ekstremno programiranje (XP)
- Scrum
- “Čisti” razvoj (engl. *Lean development*)
- Kanban i Scrumban
- Disciplinirana agilna isporuka (DAD)
- Scrum na velikoj skali (LeSS)
- Adaptivni razvoj programske potpore (ASD)
- Crystal clear i ostale metode *crystal*
- Metoda dinamičnog razvoja sustava (DSDM)
- Razvoj vođen karakteristikama (FDD)
- Agile Unified Process (AUP)

Zaključno o agilnom razvoju

- naručitelj/korisnik spremjan usko surađivati s razvojnim timom
- visok stupanj povezanosti članova razvojnog tima

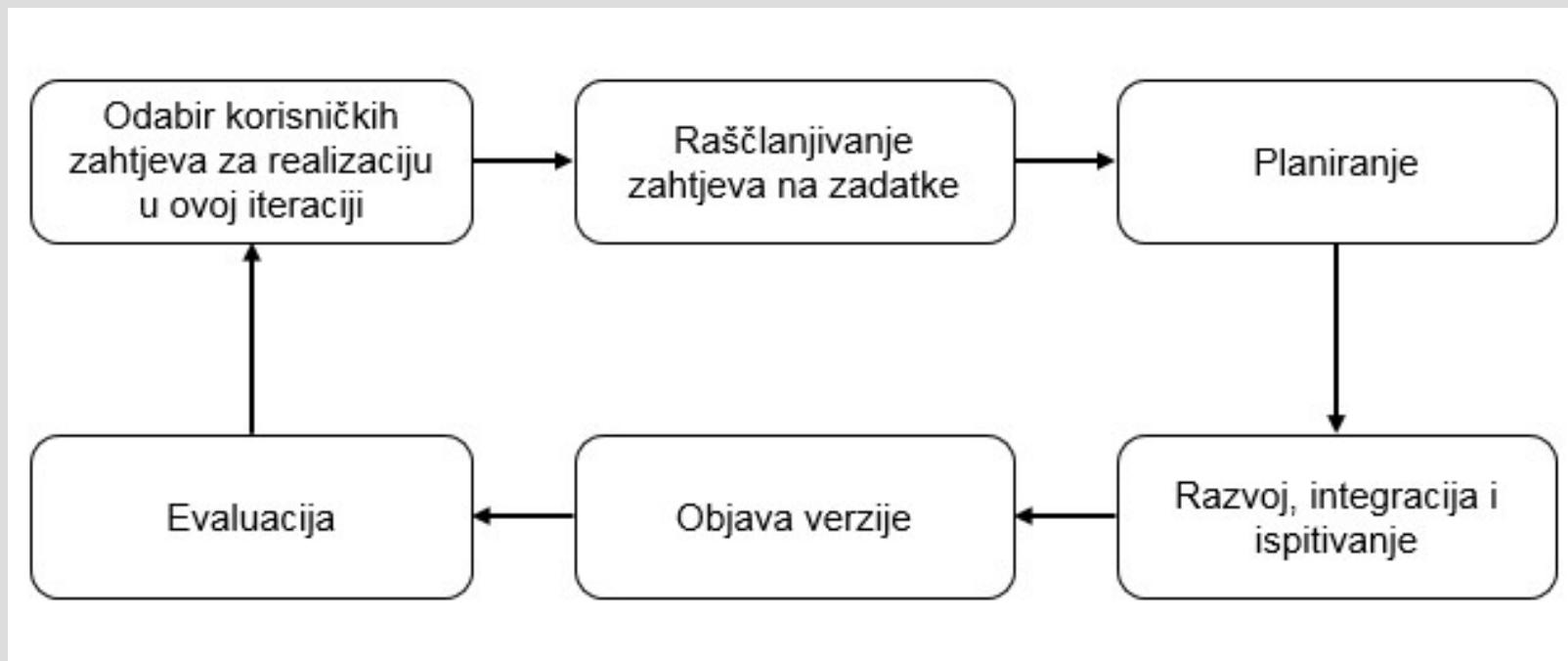
Za i protiv agilnog razvoja

- razvoj „velikih“ programskih sustava koji zahtijevaju suradnju između više razvojnih timova ili pak razvoj programske potpore s posebnim zahtjevima (sigurnost, pouzdanost, odziv u stvarnom vremenu)
- razvoj programske potpore koja će se koristiti kroz dugi vremenski period: manjak formalne i opsežne dokumentacije

Ekstremno programiranje (XP)

- jedina mjera napretka je funkcionalni programski proizvod
- razvoju, oblikovanju i brzoj isporuci funkcionalnih dijelova, uz stalno jedinično ispitivanje i često integracijsko ispitivanje
- stalno poboljšavanje koda i sudjelovanje korisnika/naručitelja u razvojnog timu
- programiranje u paru (*engl. pairwise programming*)

Ekstremno programiranje (XP)



Čisti razvoj programske potpore

- (engl. *Lean development, Lean*)
- cilj razviti u što kraćem vremenu sustav koji će maksimalno zadovoljiti korisnike uz minimalni nepotrebni angažman

Čisti razvoj programske potpore

- Principi oblikovanja:
 - eliminacija svega suvišnoga
 - naglašeno učenje
 - odlaganje donošenja odluke
 - brza isporuka
 - uvažavanje tima
 - ugradnja cjelovitosti u sustav
 - optimizacija cjeline

Scrum

- radni okvir agilnog razvoja programske potpore koji je strukturiran tako da podrži razvoj složenih proizvoda
- znanje dolazi iz iskustva i odlučivanja temeljenog na poznatome
- Timovi:
 - uloge, događaji, artefakti, pravila

Uloge u Scrumu

- vlasnik proizvoda (engl. *product owner*)
- Scrum vodja (engl. *Scrum master*)
- razvojni tim (engl. *development team member*)

Događaji u Scrumu

- (engl. *Scrum events*), osnovna jedinica razvoja **sprint**:
 - sastanak planiranja sprinta (engl. *sprint planning*)
 - sprint i dnevni sastanak Scrum tima (ili dnevni Scrum, engl. *daily Scrum, stand-up*)
 - revizija (ili recenzija) sprinta (engl. *sprint review*)
 - restrospektiva sprinta (engl. *sprint retrospective*).

Artefakti u Scrumu

- projektni dnevnik zaostataka (engl. *product backlog*)
- sprint dnevnik (engl. *sprint backlog*) je skup stavaka odabране za Sprint
- inkrement
- → završenost se određuje nekim *pravilima* Scruma dogovorenima unutar svake tvrtke

REFERENCE I LITERATURA

- A. Jović, N. Frid, D. Ivošević: Procesi programskog inženjerstva, 3. izdanje, Sveučilište u Zagrebu, FER ZEMRIS, 2019.
- I. Sommerville, Software engineering, 10th ed., Pearson, 2016.
- G. Overgaard, B. Selic, C. Bock, OMG, 2000.
- S. Tockey: How to Engineer Software, Wiley-IEEE Computer Society Press, 2019.

Hvala na pažnji!

Pitanja?



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

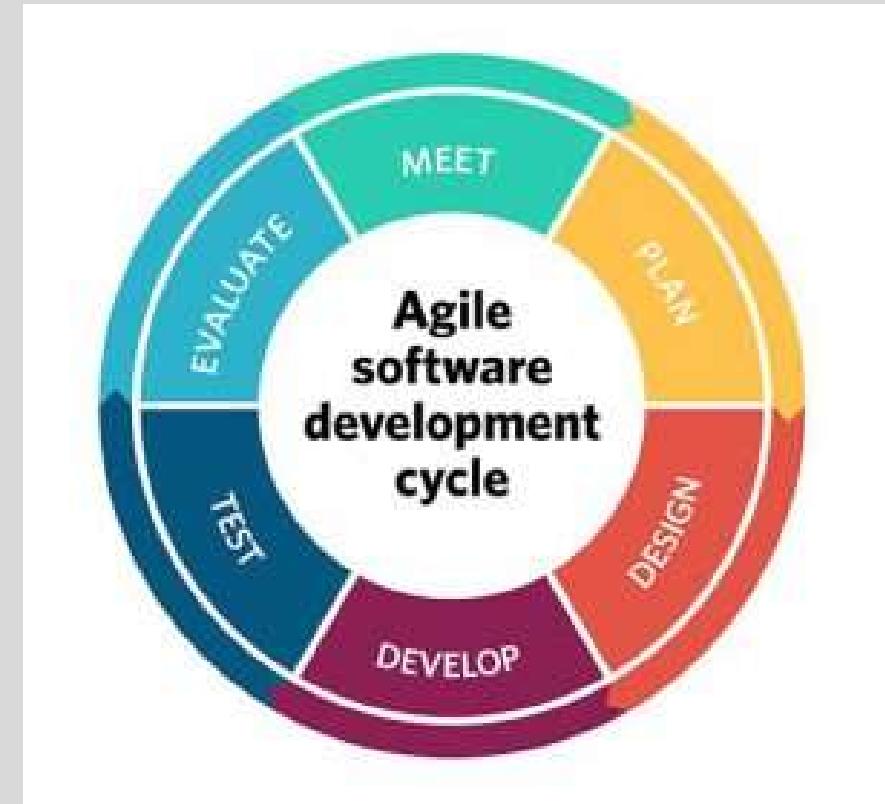
Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

AGILNA PROJEKTNA METODOLOGIJA



Uvod (1)

- Kako bi odgovorile na dinamičnost suvremenog tržišta, tvrtke **moraju brzo reagirati i prilagoditi se promjenama.**
 - Razvoj novih poslovnih aplikacija mora moći pratiti zahtjeve tržišta.
- Često je **najvažniji zahtjev mogućnost isporuke programskog proizvoda u što kraćem vremenu.**

Uvod (2)

- Ali zbog dinamične i promjenjive prirode tržišta, nije moguće potpuno definirati **sve zahtjeve** na samom početku razvoja programskog proizvoda.
- **Standardni modeli** razvoja informacijskih sustava, koji imaju jasno definirane i odvojene faze razvoja, nisu adekvatni i često ne mogu udovoljiti ovim zahtjevima.

Uvod (3)

- Potreba za novim metodama razvoja programske potpore, koje će moći odgovoriti na novonastalu situaciju na IT tržištu, dovela je do razvoja **agilnog pristupa razvoju programske potpore.**
 - Nazivi:
 - Agilna projektna metodologija,
 - Agilna metoda razvoja programske potpore,
 - Agilno upravljanje.

Agilna metodologija – definicija

- Agilni pristup razvoju programske potpore podrazumijeva skupinu metoda za razvoj programske potpore kojima je zajedničko:
 1. **iterativni razvoj uz male inkremente i**
 - 2. brz odziv na korisničke zahtjeve.**
- Agilni model koristi se **za razvoj manjih i srednjih projekata** u stalnoj interakciji s klijentima putem **stalne demonstracije** (predočavanja) **novih poboljšanja**, uz **relativno slabo dokumentiranje**.

Motivacija za agilni razvoj (1)

- Principi agilnog pristupa izneseni su u proglašu (*Agile Manifesto*) koji je sastavilo 17 istaknutih programskih inženjera u SAD-u 2001.:

Manifesto for Agile Software Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

Motivacija za agilni razvoj (2)

- Definicije agilnog principa:
 - **Pojedinci i njihovi međusobni odnosi** – u agilnom pristupu važni su samoorganizacija, motivacija i interakcije među pojedincima u timu.
 - **Upotrebljiva programska potpora** – važnije je naručitelju isporučiti programski proizvod koji je odmah upotrebljiv, nego sastaviti iscrpnu dokumentaciju.
 - **Suradnja s naručiteljem** – budući da svi zahtjevi nisu poznati odmah na početku, suradnja s naručiteljem i budućim korisnicima je neizostavna.
 - **Reagiranje na promjenu** – promjene u zahtjevima su stalne i na njih se mora odgovoriti u što kraćem roku.

Temeljna načela agilnog razvoja (1)

- Postoji 12 temeljnih načela agilnog razvoja:
 1. Najvažnije je zadovoljstvo naručitelja koje se postiže ranom i neprekinutom isporukom programskog proizvoda koji donosi vrijednost.
 2. Spremno prihvatanje promjena zahtjeva, čak i u kasnoj fazi razvoja. Agilni procesi uprežu promjene kako bi naručitelju stvorili kompetitivnu prednost.
 3. Česta isporuka upotrebljivog programskog proizvoda, u razmacima od nekoliko tjedana do nekoliko mjeseci, nastojeći da razmak bude čim kraći.
 4. Poslovni ljudi i razvojni inženjeri moraju svakodnevno zajedno raditi, tijekom cijelog trajanja projekta.

Temeljna načela agilnog razvoja (2)

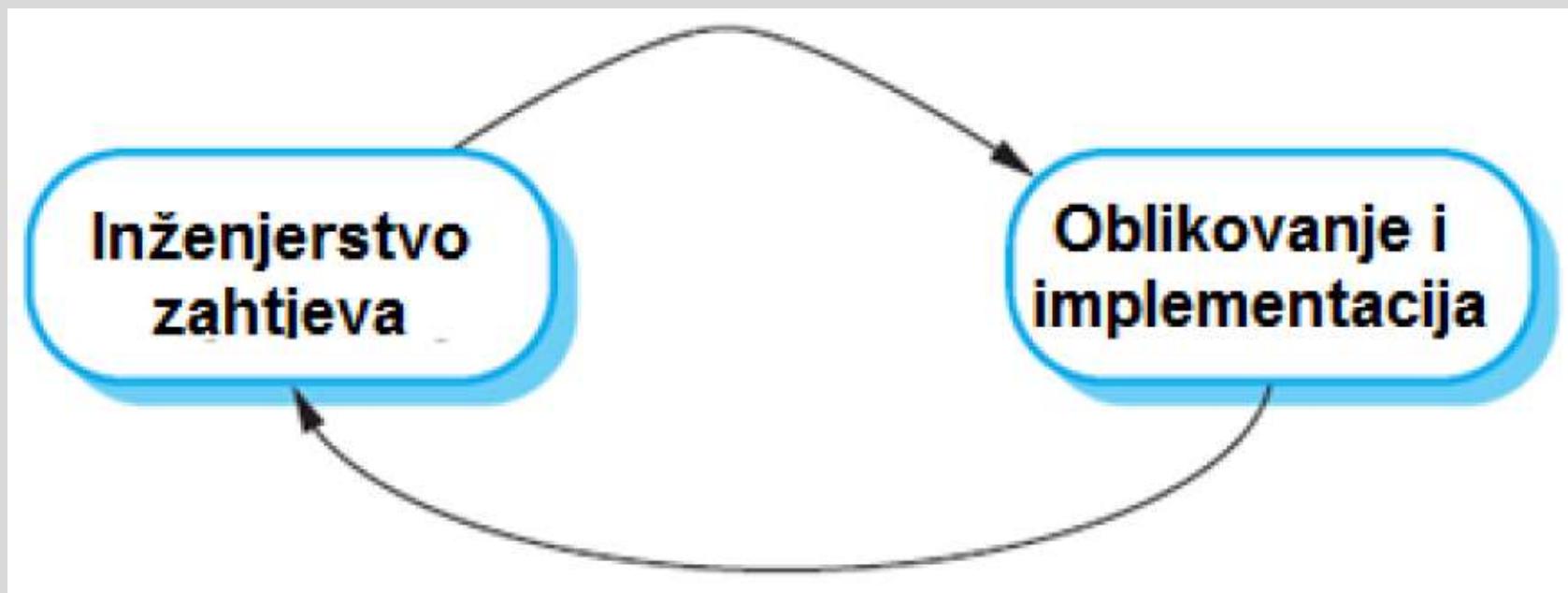
- Postoji 12 temeljnih načela agilnog razvoja:
 5. Projekti se ostvaruju oslanjajući se na motivirane pojedince. Pružene su im potrebno okruženje i podrška, i prepušten im je posao s povjerenjem.
 6. Razgovor uživo je najučinkovitiji način prijenosa informacija razvojnom timu i unutar tima.
 7. Upotrebljiv programski proizvod je osnovno mjerilo napretka.
 8. Agilni procesi potiču i podržavaju održivi razvoj. Pokrovitelji, razvojni inženjeri i korisnici trebali bi moći neograničeno dugo zadržati jednak tempo rada.

Temeljna načela agilnog razvoja (3)

- Postoji 12 temeljnih načela agilnog razvoja:
 9. Neprekinuti naglasak na tehničkoj izvrsnosti i dobrom oblikovanju pospješuju agilnost.
 10. Jednostavnost – vještina povećanja količine posla koji ne treba raditi – je od suštinske važnosti.
 11. Najbolje arhitekture, projektne zahtjeve i oblikovanje programske potpore stvaraju samo-organizirajući timovi.
 12. Tim u redovitim razmacima razmatra načine da postane učinkovitiji i zatim usklađuje i prilagođava svoje ponašanje u tom smjeru.

Agilni razvoj

- Agilni razvoj odvija se kroz iteracije u kojima se neprekidno smjenjuju faza razrade zahtjeva za sljedeći inkrement u razvoju i faza implementacije funkcionalnosti tog inkrementa.



Obuhvaćene metode

- Pojam agilnog razvoja obuhvaća mnoštvo različitih metoda, od kojih su neke nastale i prije samog *Agile* proglaša:
 - Ekstremno programiranje (XP)
 - *Scrum*
 - Agilno modeliranje
 - Adaptivni razvoj programske potpore (ASD)
 - *Crystal clear* i ostale metode *crystal*
 - Metoda dinamičnog razvoja sustava (DSDM)
 - Razvoj vođen karakteristikama (FDD)
 - “Čisti” razvoj (engl. *lean development*)
 - Agile Unified Process (AUP)
 - ...

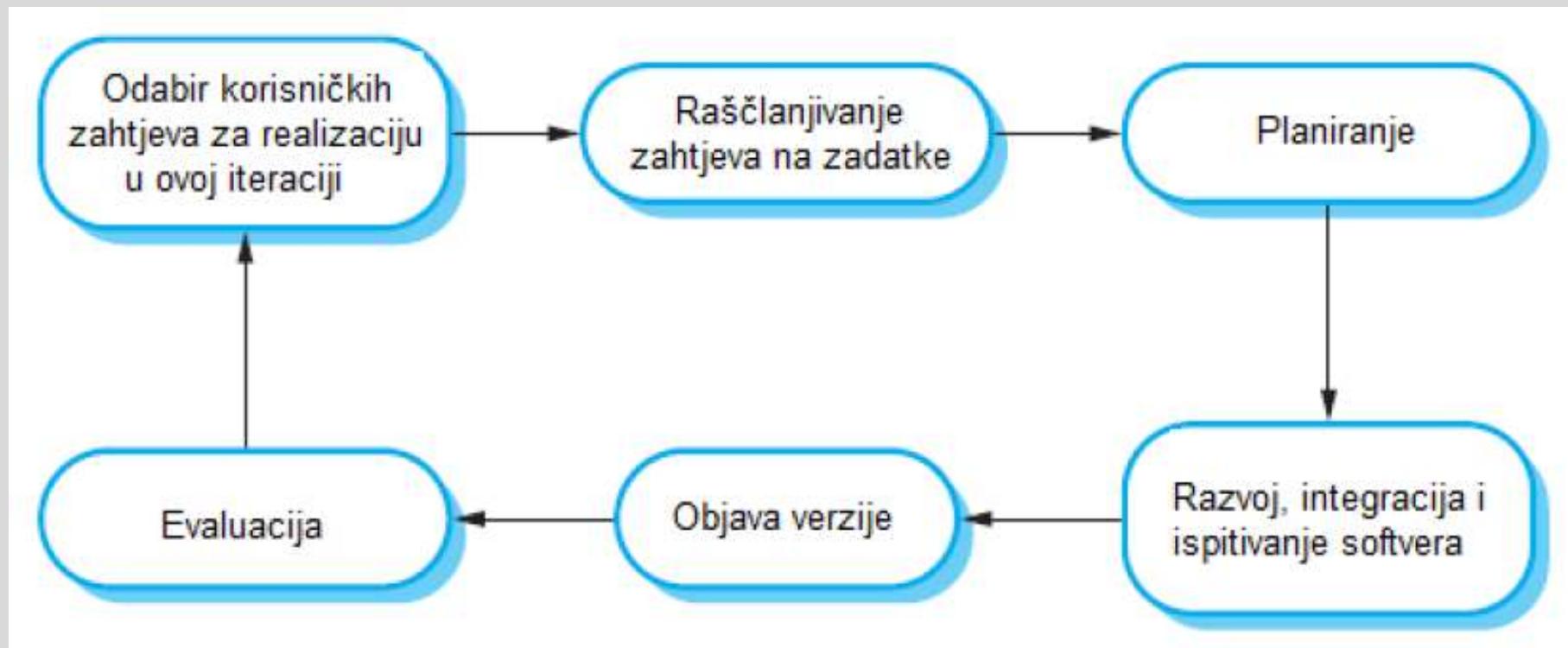
Ekstremno programiranje (1)

- **Ekstremno programiranje (*eXtreme Programming, XP*)** je jedna od najpoznatijih metoda agilnog pristupa.
- Javlja se sredinom 1990-tih kao otpor strukturiranom (ponajviše vodopadnom) procesu programskega inženjerstva za koji se smatra da unosi previše birokracije u proces oblikovanja.
- **U ekstremnom programiranju jedina mјera napretka je funkcionalni programski proizvod.**
- Ekstremno programiranje **zasniva se na razvoju, oblikovanju i isporuci funkcionalnih dijelova programske potpore.**

Ekstremno programiranje (2)

- Ekstremno programiranje uključuje **kontinuirano poboljšavanje koda i sudjelovanje korisnika/naručitelja u razvojnog timu**.
- U razvoju sustava najčešće se programira u paru (jedno radno mjesto uz međusobno provjeravanje, ***pairwise programming***).
- **Glavni nedostatci ovakvog pristupa su nestabilnost zahtjeva i nerazumljivost sustava zbog nedokumentiranja, što rezultira da postojeće rješenje uglavnom ne podupire ponovnu uporabu (*reuse*) rješenja.**

Dijagram procesa XP razvoja



Čisti razvoj programske potpore

- *Lean razvoj programske potpore (Lean software development, hrv. mršav, slab)* odnosi se na metodu koja je izvorno nastala u proizvodnjoskom sustavu Toyote (*Lean manufacturing principle*), a čiji je cilj razviti u što kraćem vremenu sustav koji će zadovoljiti korisnike.



Principi čistog razvoja (1)

- Ova metoda temelji se na sedam principa oblikovanja:
 1. **Eliminacija svega suvišnoga** (koda, nejasnih zahtjeva, birokracije, spore interne komunikacije).
 2. **Naglašeno učenje** (stalna komunikacija s klijentom, stalna ispitivanja i brze nadogradnje).
 3. **Odlaganje odluke** (predlaganjem opcija klijentu, skupljanjem činjenica, *Just In Time - JIT*).
 4. **Dostava proizvoda što je brže moguće** (niz metoda, uglavnom mali inkrementi, više timova radi isto).

Principi čistog razvoja (2)

- Ova metoda temelji se na sedam principa oblikovanja:
 5. **Motivacija** i suradnja unutar tima.
 6. **Ugradnja cjelovitosti** u sustav (kupac mora biti zadovoljan sa sustavom u cjelini: funkcionalnošću, intuitivnošću korištenja, cijenom).
 7. **Razumijevanje modela** (svaki član tima mora znati kako i zašto čisti razvoj programske potpore treba funkcionirati).

Agilne metode – prednosti

- Agilne metode općenito su pogodne u situaciji kada je naručitelj/korisnik spremna usko surađivati s razvojnim timom.
- Zbog manjka formalnosti u cijelom procesu razvoja, izravna komunikacija između članova razvojnog tima je nezaobilazna te stoga timovi moraju biti sastavljeni od članova spremnih na visok stupanj povezanosti.

Agilne metode – nedostaci (1)

- Ali **kada je u pitanju razvoj „velike“ programske potpore ili pak razvoj programske potpore s posebnim zahtjevima** (sigurnost, pouzdanost, odaziv u stvarnom vremenu, kasnije nadogradnje, dugotrajno održavanje, ...), **agilne metode ne mogu dati zadovoljavajuće rješenje te je potreban strukturirani razvojni proces s višom razinom formalnosti u komunikaciji.**

Agilne metode – nedostaci (2)

- Agilne metode nisu se pokazale dobrima kada je u pitanju razvoj programske potpore koja će se koristiti kroz dugi vremenski period.
- Zbog manjka formalne i opsežne dokumentacije, gotovo je nemoguće raditi na kasnijem održavanju i nadogradnji jednom kada je gotov proizvod isporučen.

REFERENCE I LITERATURA

- Predavanja ovog predmeta
- Manifesto for Agile Software Development,
<http://agilemanifesto.org/>, pristupljeno 03/2015.
- R. C. Martin, Agile Software Development: Principles, Patterns and Practices, Prentice Hall, Upper Saddle River, NJ, 2002.
- Nastavni materijali kolegija Oblikovanje programske potpore, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

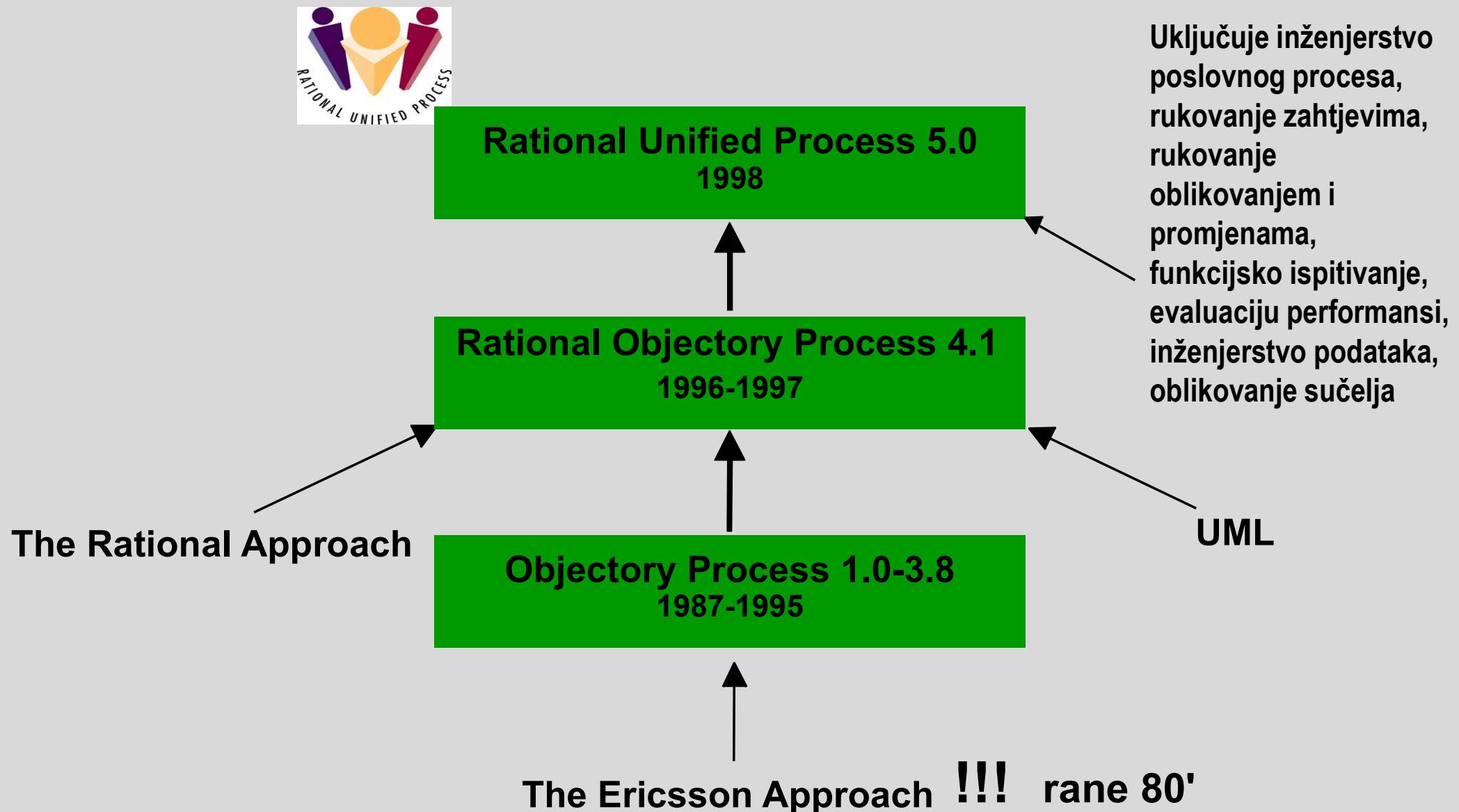
Akademska godina 2021./2022.
Ljetni semestar



Izvor: Ivar Jacobson: Applying UML in The Unified Process

UNIFIED PROCESS (UP)

Kako je nastao UP? (1)



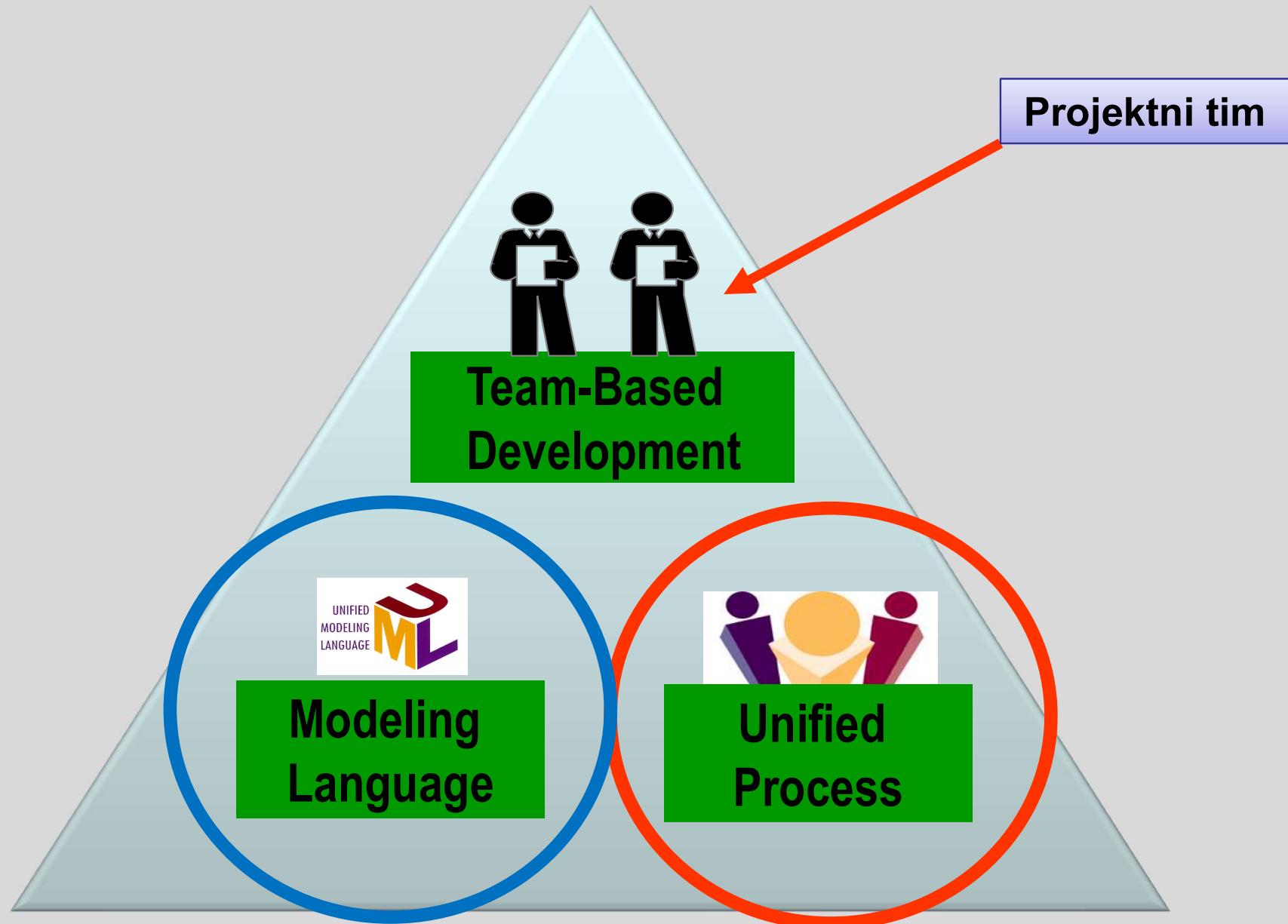
Kako je nastao UP? (2)

- Razvoj UP-a započeo je ranih 1980-ih u tvrtci Ericsson kao odgovor na potrebu za novom projektnom metodologijom koja će optimalno odgovarati različitim potrebama tvrtke za razvojem novih proizvoda u programskom inženjerstvu, ali i ne samo u njemu.
- Prema inačici UP-a koja je završena krajem 90-ih, određeno je da opseg i aktivnosti ove metodologije uključuju inženjerstvo poslovnog procesa, rukovanje zahtjevima, rukovanje oblikovanjem i promjenama, funkcionalno ispitivanje, vrednovanje performansi, inženjerstvo podataka te oblikovanje sučelja.
- Nakon 2003. godine razvoj UP-a nastavljen je u tvrtci Rational Software unutar IBM-a → Rational Unified Process (RUP)

Što je UP?

- **Unificirani proces** (engl. *Unified Process*, UP) je metodologija razvoja programske potpore koja se temelji na oblikovanju pomoću modela (engl. *Model Based Design*, MBD), odnosno iterativnom razvoju, obrascima uporabe i usmjerenjem na arhitekturu sustava.

Tim + UML + UP = dobitna kombinacija



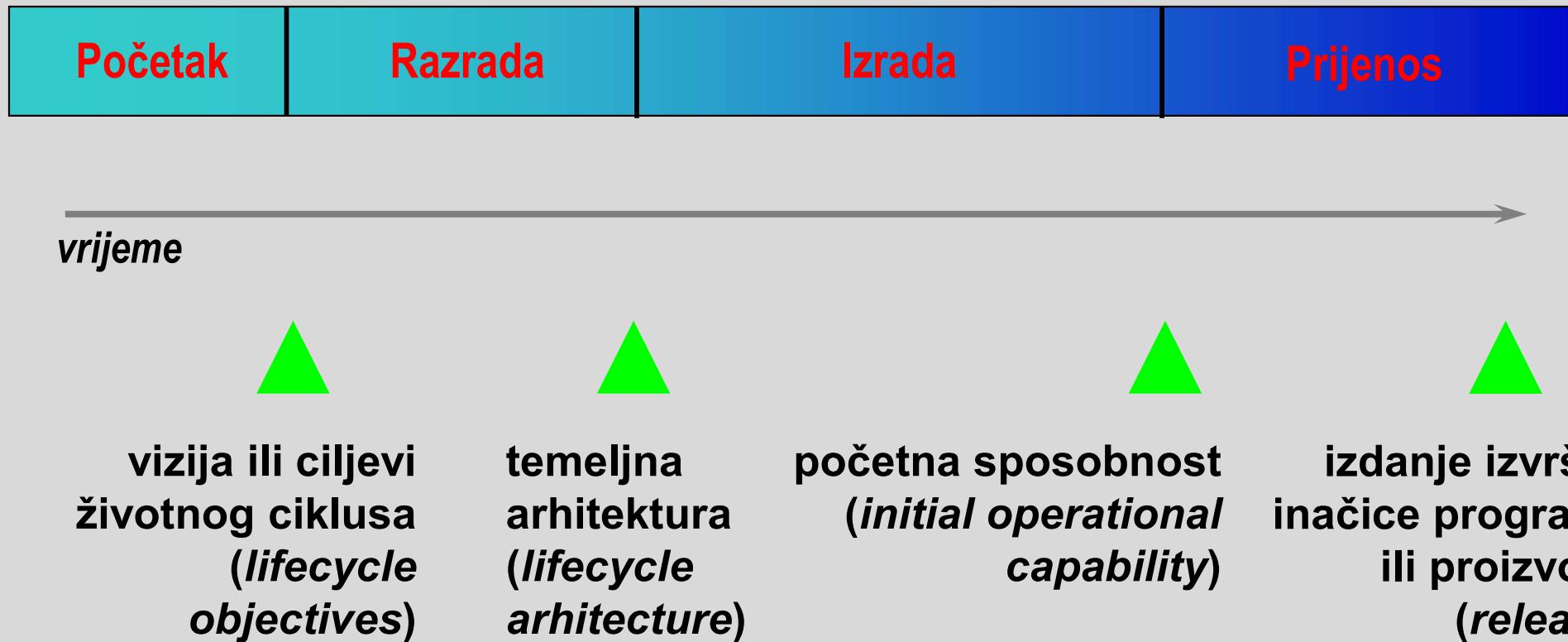
Prikaz UP-a

- UP se prikazuje kroz tri temeljna obilježja:
 1. Promiče **iteracije** na pojedinim **fazama** oblikovanja programske potpore
 2. Temelji se na **obrascima uporabe** (*use case*, tj. predlošku scenarija)
 3. U fokusu UP-a je **arhitektura** sustava

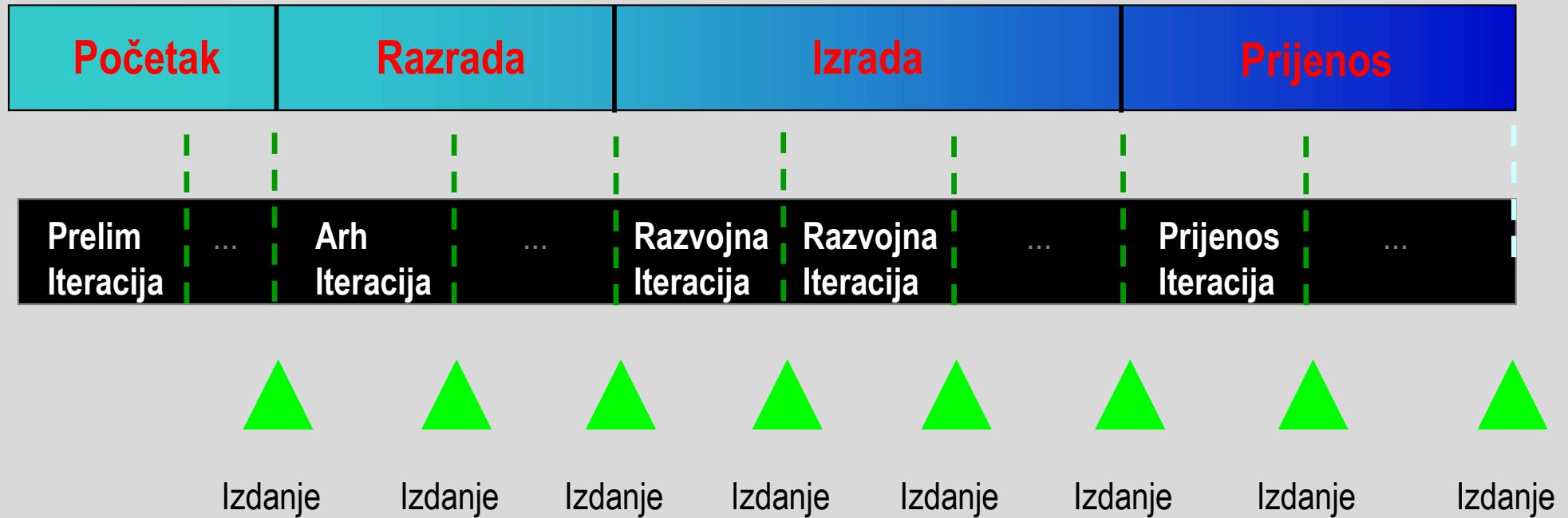
Faze u životnom ciklusu UP procesa

- Kao i svaka druga projektna metodologija, UP određuje faze životnog ciklusa (engl. *lifecycle*) procesa i dokumente koji se moraju izraditi završetkom izvođenja svake faze.
- **Faze životnog ciklusa UP-a su:**
 1. **Početak** (*inception*), u kojemu se definira doseg projekta, razvoj modela poslovnog procesa, specifikacija početnih zahtjeva,
 2. **Razrada, elaboracija** (*elaboration*), kada se definiraju plan projekta, specifikacija značajki i temelji arhitekture sustava,
 3. **Izgradnja, izrada** (*construction*), koja se sastoji od oblikovanja, programiranja i ispitivanja,
 4. **Prijenos** (*transition*), pri čemu se završni proizvod prenosi korisnicima i postavlja u radnu okolinu.

Glavne ključne točke i pridruženi dokumenti ("Milestones")



Faze i pridružene **iteracije**



Svaka faza ima barem jednu iteraciju. Stoga, u kontekstu UP-a, iteracija je niz ili slijed aktivnosti u okviru prihvaćenog plana i kriterija evaluacije. **Ishod svake iteracije je dokument, a na kraju i jedna izvršna inačica (tj. izdanje) programa ili sustava.**

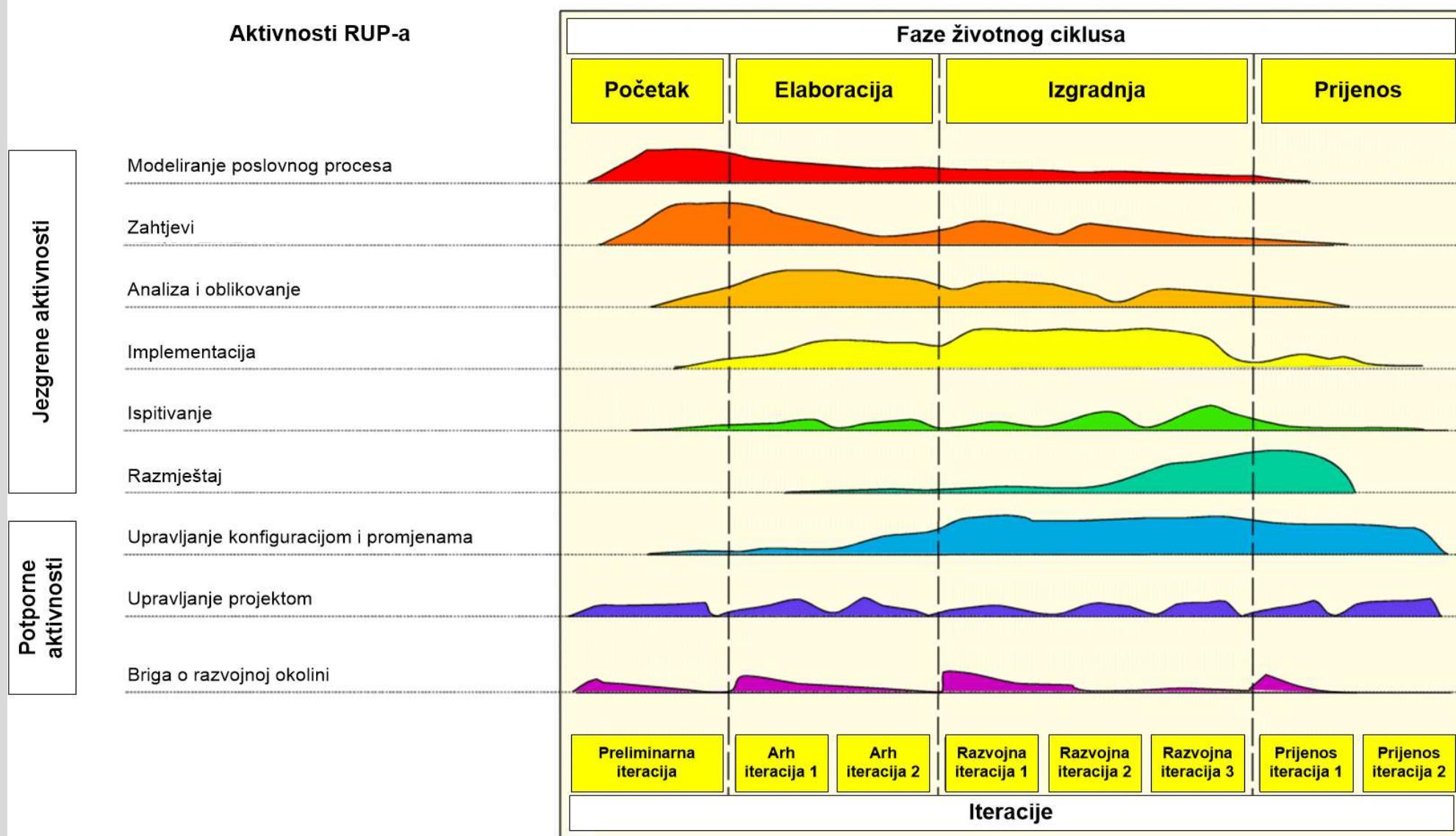
Iteracije i aktivnosti

- Osim faza životnog ciklusa, UP definira i niz aktivnosti koje se mogu odvijati u svim fazama, ali obično s različitim intenzitetom.
- **Jezgrene aktivnosti (core workflows)** UP-a su:
 1. Modeliranje poslovnog procesa (*business modelling*),
 2. Zahtjevi (*requirements*),
 3. Analiza i oblikovanje (*analysis and design*),
 4. Implementacija (*implementation*),
 5. Ispitivanje (*test*),
 6. Razmještaj (*deployment*)

Iteracije i potporne aktivnosti

- Vrlo često se dodaju dodatne, **potporne aktivnosti (*support workflows*)** vezane uz organizaciju i provedbu životnog ciklusa, kao što su: **upravljanje konfiguracijom i promjenama** (*configuration and change management*), **upravljanje projektom** (*project management*), te **briga o razvojnoj okolini** (*environment*; svodi se na ispravnu interakciju s dionicima projekta).
- Ove dodatne aktivnosti se zbog svoje namjene zajedno nazivaju potporne aktivnosti.

Iteracije i aktivnosti



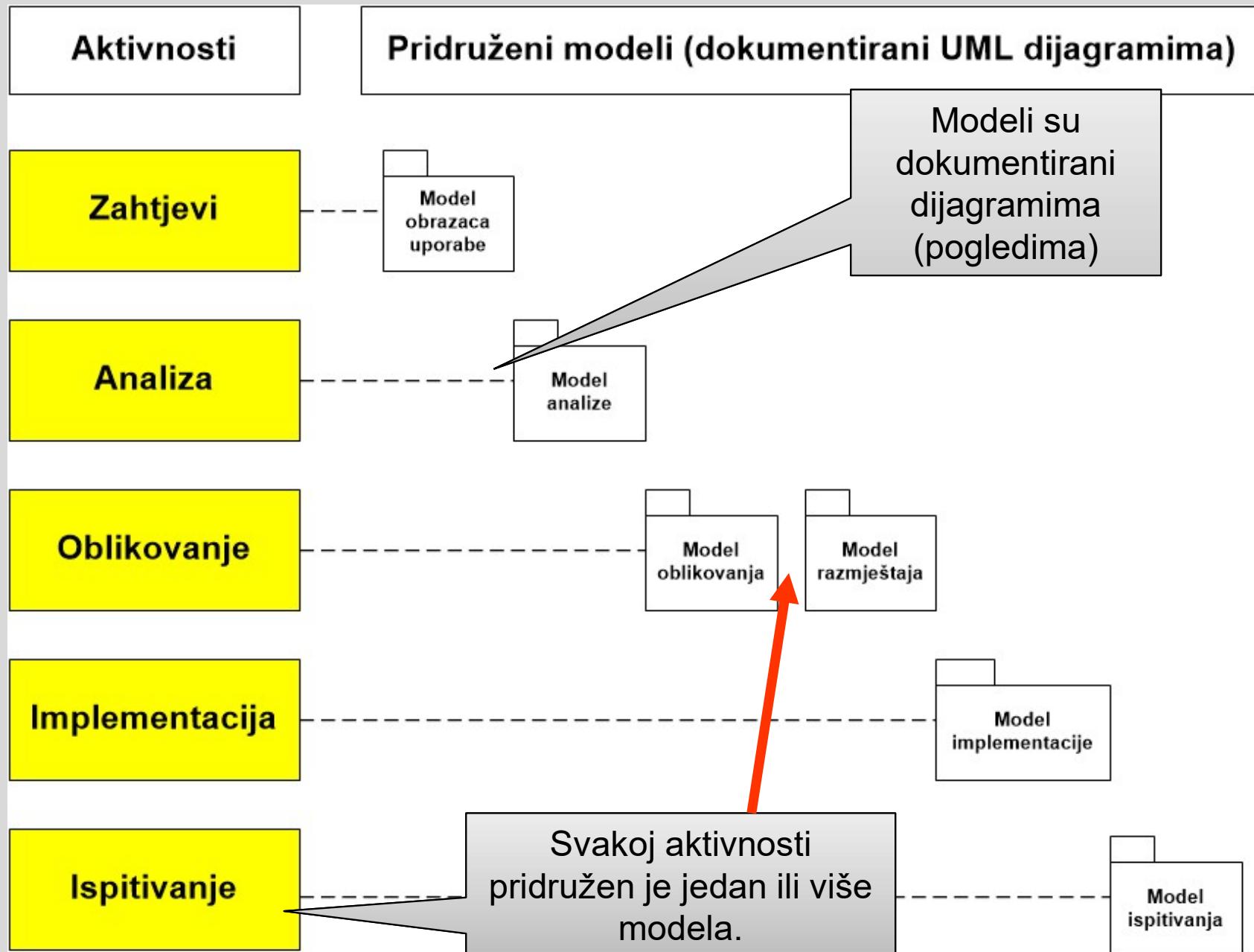
Vođenje projekata u modelno-usmjerenom razvoju

- **Ovisno o fazi u kojoj se nalazi razvoj proizvoda, bit će potrebno dodijeliti više ljudi (projektnih resursa) na određene aktivnosti, a manje na druge i obrnuto.**
- **Određivanjem prioriteta i raspoređivanjem resursa bavi se voditelj projekta (*project manager*).**
- Osim što se u svojoj strukturi temelji na iteracijama u svakoj pojedinoj fazi, UP intenzivno koristi obrasce uporabe (*use cases*) i usmjeruje se na arhitekturu sustava, odnosno na strukturu programske potpore koja se izrađuje.

Aktivnosti i pridruženi modeli

- Za opis pojedinih aktivnosti (tijeka rada) koriste se odgovarajući modeli (tj. UML dijagrami).
- Modeli su dokumentirani s jednim ili, u praksi češće, više dijagrama programske potpore. Dijagrami moraju biti definirani UML standardom.
 - Stoga, prema UP-u, oblikovana arhitektura sustava sadrži skup UML dijagrama kojima su opisani različiti pogledi (*views*) u modele sustava (*models*).
- Svakoj aktivnosti pridružen je jedan ili više modela za opis, koji su dokumentirani s jednim ili više dijagrama (tj. pogleda).

Aktivnosti i pridruženi modeli



Perspektive opisa

- UP je najčešće opisan kroz tri perspektive koje se ogledavaju i u korištenim UML dijagramima:
 1. Dinamička perspektiva, koja pokazuje slijed faza procesa kroz vrijeme,
 2. Statička perspektiva, koja pokazuje aktivnosti u pojedinim fazama procesa, i
 3. Praktična perspektiva, koja sugerira aktivnosti kroz iskustvo i dobru praksu.

Što znači: Temelji se na obrascima uporabe?

Aktivnosti:



Obrasci uporabe kao pokretači iteracija

- **Obrasci uporabe su vrlo važni u modelno-usmјerenom razvoju**, jer povezuju i pokreću brojne aktivnosti u životnom ciklusu oblikovanja programske potpore, kao što su:
 1. Stvaranje i validacija arhitekture sustava,
 2. Definicija ispitnih slučajeva, scenarija i procedura,
 3. Planiranje pojedinih iteracija,
 4. Stvaranje korisničke dokumentacije,
 5. Razmještaj sustava (engl. *deployment*)
- Obrasci uporabe pomažu u sinkroniziranju sadržaja različitih modela.

Koja projektna metodologija je najbolja?

- **Ne postoji univerzalno optimalan proces oblikovanja programske potpore!**
- Svaki pristup ili metodologija imaju svoje komparativne prednosti i nedostatke.
- Stoga se tijekom razvoja UP-a vodilo računa o fleksibilnosti i budućim proširenjima, jer se time omogućuju razne strategije životnog ciklusa projekta.
- Na primjer, moguće je odabratи које artefakte treba proizvesti, definirati nužne aktivnosti i potrebne resurse (ukupni trošak, potrebno vrijeme, nužni programeri, vještine i znanja, programska potpora, sklopoljje, itd.), te modelirati koncepte sustava.

UP ZAKLJUČCI

- U središtu UP procesa su **obrasci uporabe** sustava koji semantički povezuju sve aktivnosti.
- UP definira i međusobno povezuje dinamičke **faze** i statičke **aktivnosti** procesa (pomoću prikaza iteracija i aktivnosti).
- Za opis pojedinih **aktivnosti** koriste se odgovarajući **modeli**.
- **Modeli** su dokumentirani s jednim ili više **dijagrama** (pogleda na sustav).
- **Dijagrami** su definirani **UML standardom**.
- Oblikovana **arhitektura** sustava sadrži **skup pogleda u modele** (tj. **skup dijagrama**).

REFERENCE | LITERATURA

- Predavanja ovog predmeta
- I. Sommerville, Software Engineering, 9th ed., Addison-Wesley, Harlow, England, 2011.
- R. Pressman, Software Engineering: A Practitioner's Approach (7th Edition), McGraw-Hill Science/Engineering/Math, 2009.
- T. C. Lethbridge and R. Laganière, Object-Oriented Software Engineering: Practical Software Development Using UML and Java, 2nd ed., McGraw-Hill Publishing Company, London, 2004.
- Nastavni materijali kolegija Oblikovanje programske potpore, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



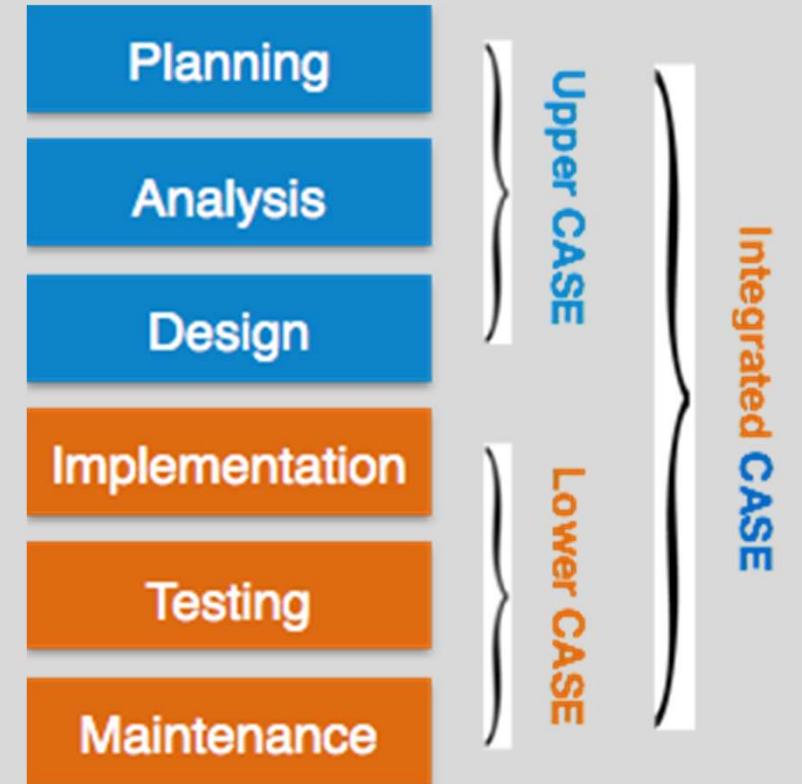
TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar



11. CASE-ALATI

Uvod (1)

- **CASE-alati** (engl. *Computer-Aided Software Engineering Tools*) su posebni **računalni alati** koji se koriste u procesu razvoja programske potpore **kako bi automatizirali dio aktivnosti, ubrzali izradu, pribavili korisne informacije o proizvodu koji se razvija te olakšali kasnije održavanje proizvoda.**

Uvod (2)

- Formalna definicija:
- **CASE-alati** su programski proizvodi koji podupiru proces programskog inženjerstva, a posebice **aktivnosti specifikacije, oblikovanja (modeliranja), implementacije i evolucije**.

Prednosti CASE-alata

- Glavne prednosti korištenja ovih alata su:
 1. Mogu se koristiti u svakoj fazi životnog ciklusa programskog proizvoda.
 2. Povećanje kvalitete konačnog proizvoda kroz normiranje načina prikaza i dijeljenja informacija.
 3. Smanjenje vremena i napora potrebnog za izradu programske potpore.
 - Postiže se automatizacijom dijela procesnih aktivnosti (npr. izrada i organizacija dokumentacije) te povećanjem ponovne iskoristivosti (engl. *reusability*) modela i komponenti.

Automatizacija

- **Automatizacija je iznimno važna značajka CASE-alata.**
- CASE podupire automatizaciju oblikovanja raznim alatima kao što su:
 - grafički uređivači za razvoj modela sustava,
 - rječnici i zbirke za upravljanje entitetima u oblikovanju,
 - okruženja za oblikovanje i konstrukciju korisničkih sučelja,
 - alati za pronalaženje pogrešaka u programu,
 - automatizirani prevoditelji koji generiraju nove inačice programa itd.

Nedostaci CASE-alata

- Napredni CASE-alati mogu biti **složeni** do te mjere da **od korisnika zahtijevaju ulaganje značajnog napora u savladavanje korištenja alata**, a niti **cijena alata** (većina naprednih alata je komercijalna) nije zanemariva.
- Osim toga, **ograničavaju kreativnost.**
 - Nastojanja da se kroz alat ostvari što veće normiranje i stupanj automatizacije su često u sukobu s potrebom za kreativnosti i pronalaženjem inovativnih rješenja koje programsko inženjerstvo zahtijeva.

Klasifikacija CASE-alata

- Pri klasifikaciji CASE-alata koriste se tri različite perspektive:
 1. **Funkcijska perspektiva** – alati se klasificiraju prema specifičnoj funkciji koju obavljaju.
 2. **Procesna perspektiva** – alati se klasificiraju prema aktivnostima koje podupiru u procesu.
 3. **Integracijska perspektiva** – alati se klasificiraju prema njihovoj organizaciji u integrirane cjeline.

Funkcijska perspektiva (1)

- Vrste CASE-alata prema funkcijskom pogledu:
 - Vrsta: Alati za planiranje (engl. *planning tools*)
 - Primjer: PERT alati, tablični kalkulatori (npr. Excel)
 - Vrsta : Alati za uređivanje (engl. *editing tools*)
 - Primjer: Uređivači teksta, dijagrama (npr. Notepad, Word, Visio)
 - Vrsta : Alati za upravljanje promjenama (engl. *change management tools*)
 - Primjer: Sustavi za praćenje promjena u zahtjevima i proizvodu (npr. IBM Rational DOORS, Borland Caliber)

Funkcijska perspektiva (2)

- Vrste CASE-alata prema funkcijskom pogledu:
 - Vrsta: Alati za upravljanje konfiguracijom (engl. *configuration management tools*)
 - Primjer: Sustavi za kontrolu i upravljanje inačicama (npr. Subversion, Git)
 - Vrsta: Alati za izradu prototipa (engl. *prototyping tools*)
 - Primjer: Jezici vrlo visoke razine apstrakcije (npr. UML)
 - Vrsta: Alati za potporu metodama (engl. *method-support tools*)
 - Primjer: Različiti podatkovni rječnici, generatori koda

Funkcijska perspektiva (3)

- Vrste CASE-alata prema funkcijskom pogledu:
 - Vrsta: Alati za jezično procesiranje (engl. *language-processing tools*)
 - Primjer: Kompajleri, interpreteri
 - Vrsta: Alati za programsku analizu (engl. *program analysis tools*)
 - Primjer: Različiti alati za analizu statičkih i dinamičkih performansi programa
 - Vrsta: Alati za ispitivanje (engl. *testing tools*)
 - Primjer: Generatori ispitnih skupova

Funkcijska perspektiva (4)

- Vrste CASE-alata prema funkcijskom pogledu:
 - Vrsta: Alati za ispravljanje pogrešaka (engl. *debugging tools*)
 - Primjer: Alati ugrađeni u razvojne okoline (npr. Visual Studio, Eclipse...)
 - Vrsta: Alati za izradu dokumentacije (engl. *documentation tools*)
 - Primjer: Različiti alati za prijelom i uređivanje slika
 - Vrsta: Alati za reinženjering (engl. *re-engineering tools*)
 - Primjer: Posebni alati za unakrsnu usporedbu dijelova sustava i restrukturiranje programa

Procesna perspektiva

S obzirom na generičke aktivnosti procesa programskog inženjerstva

Alati za reinženjering

Alati za ispitivanje

Alati za ispravljanje pogrešaka

Alati za programsku analizu

Alati za jezično procesiranje

Alati za potporu metodama

Alati za izradu prototipa

Alati za upravljanje

konfiguracijom

Alati za upravljanje promjenama

Alati za izradu dokumentacije

Alati za uređivanje

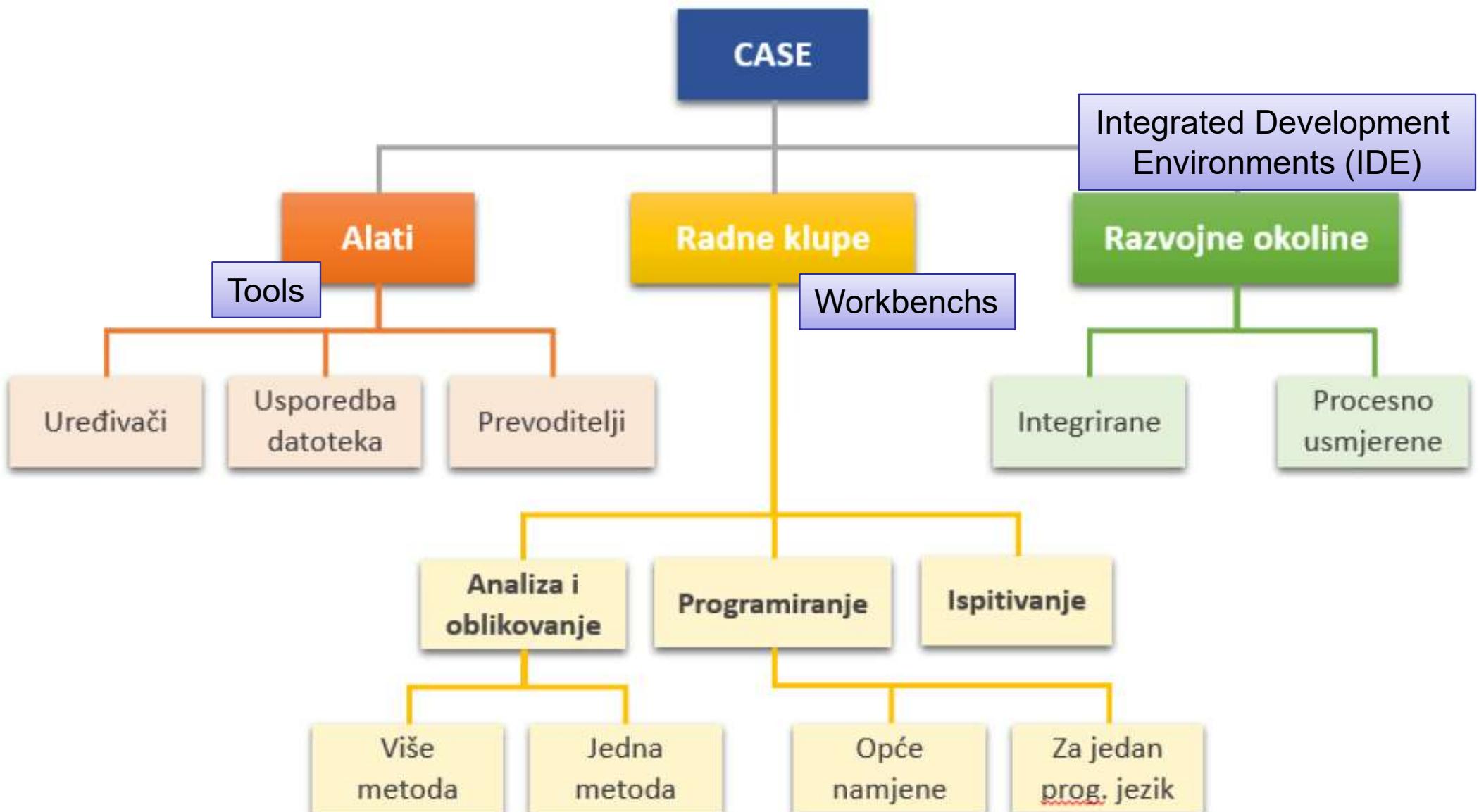
Alati za planiranje

	Specifikacija	Oblikovanje	Implementacija	Validacija i verifikacija
Alati za reinženjering			✓	
Alati za ispitivanje			✓	✓
Alati za ispravljanje pogrešaka			✓	✓
Alati za programsku analizu			✓	✓
Alati za jezično procesiranje		✓	✓	
Alati za potporu metodama	✓	✓		
Alati za izradu prototipa	✓			✓
Alati za upravljanje konfiguracijom		✓	✓	
Alati za upravljanje promjenama	✓	✓	✓	✓
Alati za izradu dokumentacije	✓	✓	✓	✓
Alati za uređivanje	✓	✓	✓	✓
Alati za planiranje	✓	✓	✓	✓

Integracijska perspektiva

- **Ova perspektiva promatra alate s obzirom na stupanj integracije alata u cjelinu.**
- Alati (u užem smislu) podupiru individualne zadatke u procesu (npr. oblikovanje, provjeru dosljednosti zahtjeva, uređivanje teksta, ...).
- **Dvije kategorije alata prema ovoj perspektivi:**
 1. **Viši CASE, upper-CASE alati** (*front-end*), koji se koriste u raniji fazama kao što su izlučivanje zahtjeva, analize i modeliranja
 2. **Niži CASE, lower-CASE alati** (*back-end*), koji se koriste u kasnijim fazama implementacije, ispitivanja i održavanja.

Integracijska klasifikacija CASE-alata



Radne klupe

- **Radne klupe (engl. *workbenches*)** podupiru pojedine aktivnosti (faze) procesa (npr. specifikaciju).
- Radne klupe objedinjuju više različitih alata za potporu u nekoj fazi procesa programskog inženjerstva.
- **Najčešće podržavaju jednu od tri aktivnosti:**
**1) analizu i oblikovanje, 2) programiranje te
3) ispitivanje.**

Razvojne okoline

- **Razvojne okoline (engl. *environments*) podupiru cijeli ili značajan dio procesa programskog inženjerstva.**
- **Uključuju nekoliko integriranih radnih klupa.**
- Primjeri razvojnih okolina:
 - Visual Studio, Eclipse, IntelliJ, NetBeans, ...

Primjeri CASE-alata i radnih klupa

- CASE-alati i radne klupe:
 - Subversion, Git – upravljanje konfiguracijama programske potpore.
 - ArgoUML, Astah Community Edition, Enterprise Architect – oblikovanje zahtjeva, oblikovanje modela objektno usmjerene arhitekture.
 - Microsoft Visio - oblikovanje zahtjeva, oblikovanje modela objektno usmjerene arhitekture.
 - Microsoft Visual Studio – implementacija programske potpore vezana uz Microsoftove tehnologije
 - Eclipse – implementacija programske potpore vezana uz Java tehnologije (i druge).

Upravljanje izvornim kodom (1)

- Prilikom razvoja programske potpore nezaobilazna je međusobna suradnja većeg broja osoba i timova koji se često nalaze na razdvojenim lokacijama (npr. različitim državama i vremenskim zonama), no rade nad istim datotekama.
- Kontrola inačica datoteka i vođenje evidencije o promjenama koje su nastale je potpuna nužnost!

Upravljanje izvornim kodom (2)

- Formalna definicija:
- U programskom inženjerstvu, kontrola inačica programske potpore je svaki postupak koji prati i omogućava upravljanje promjenama nastalima u datotekama s izvornim kodom ili dokumentacijom.

Važni pojmovi

- **Revizija** je osnovni pojam kojim se opisuje slijed razvoja.
- Jedinstveni slijed razvoja u kojem nema grananja naziva se **osnovna razvojna linija** (engl. *trunk*).
- Ako postoji potreba za razvojem dodatnih značajki programske potpore mimo osnovne linije, tada dolazi do razdvajanja razvoja u dvije ili više **grana** pri čemu svaku granu još nazivamo **pomoćnom razvojnom linijom** (engl. *branch*).

Važni pojmovi – Vršna revizija

- Kad nema grananja svaka revizija temelji se isključivo na jednoj jedinoj reviziji koja joj neposredno prethodi te **sve revizije čine jednu liniju**.
- U takvom nizu postoji jedinstvena zadnja inačica koja se često naziva **vršna revizija** (engl. *head*).

Važni pojmovi – Graf revizija

- Ako postoji grananje, iz jedne revizije može nastati nekoliko novih, a također je moguće da se nova revizija ne temelji na neposrednoj prethodnici nego na nekoj ranijoj reviziji. U takvom slučaju **graf revizija** umjesto linijskog poprima stablasti oblik te se vršna revizija mora izričito zadati.

Važni pojmovi – Spajanje linija

- Proces koji dovodi do spajanja revizija iz dviju ili više razvojnih linija u jedinstvenu reviziju naziva se **spajanje linija** (engl. *merge*). U praksi je ovaj proces iznimno teško izvesti i predstavlja jedan od najsloženijih aspekata kontrole inačica.

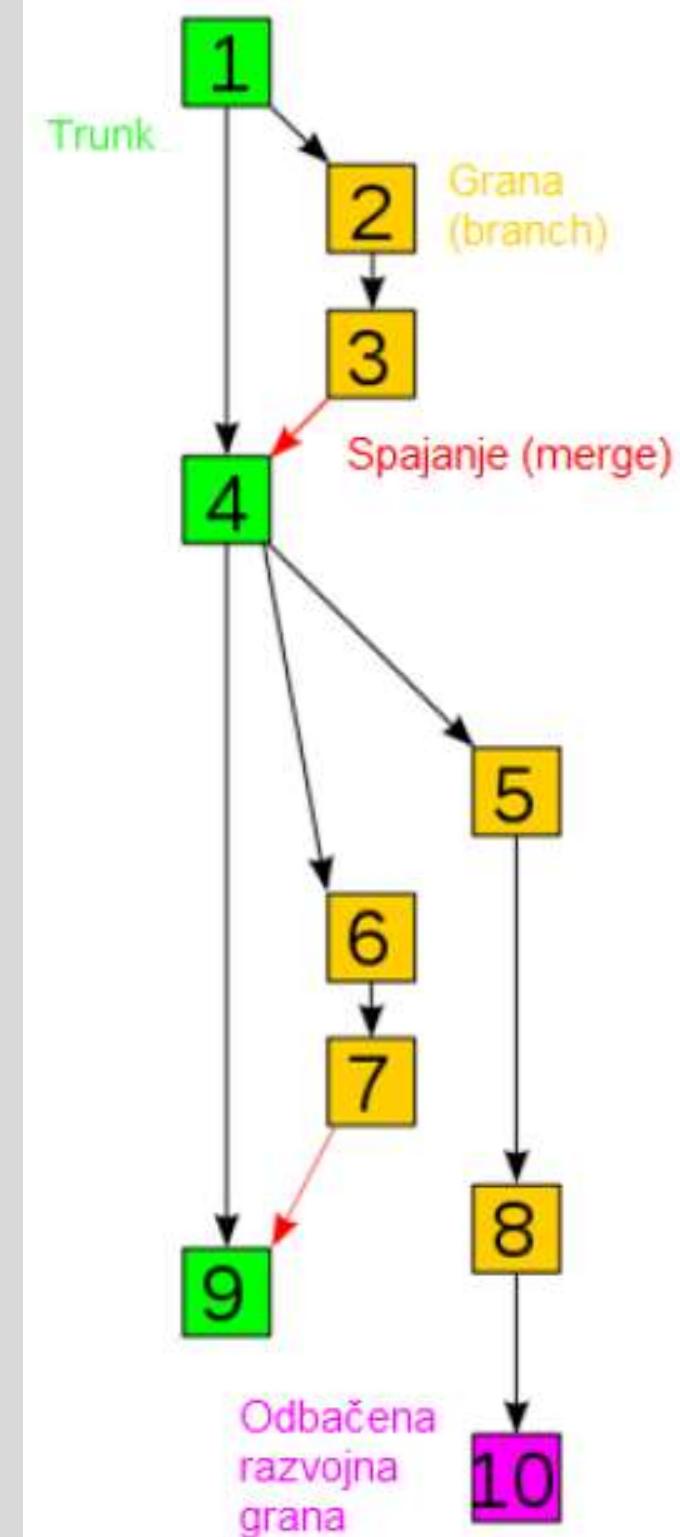
Najčešće korišteni pojmovi u sustavima za kontrolu inačica (1)

Pojam	Definicija
<i>Branch</i> (pomoćna razvojna linija)	Skup datoteka obuhvaćen sustavom kontrole inačica koji se u određenom trenutku odvaja od osnovne linije razvoja i dalje razvija zasebno i neovisno o ostalim pomoćnim linijama razvoja.
<i>Checkout</i> (dohvaćanje)	Proces stvaranja lokalne radne kopije repozitorija. Korisnik može dohvatiti vršnu reviziju ili može odabrati bilo koju dostupnu reviziju. Ovaj pojam se ponekad koristi i kao imenica koja označava samu radnu kopiju repozitorija.
<i>Commit</i> (provedba)	Proces zapisivanja promjena iz radne inačice natrag u repozitorij. Koristi se još i izraz <i>check-in</i> .
<i>Conflict</i> (sukob, spor)	Sukob koji nastaje kad dvoje ili više korisnika napravi promjene u istom dokumentu te sustav ne može automatski objediniti promjene u novu inačicu. Tada jedan od korisnika mora riješiti spor (<i>resolve</i>) tako što će sam uklopiti različite promjene ili odbaciti sve osim jedne inačice nove inačice.
<i>Export</i> (izvoz)	Proces sličan <i>checkout</i> -u s razlikom u tome da se u strukturi kazala ne nalaze i datoteke s meta-podacima potrebnim za kontrolu inačica.

Najčešće korišteni pojmovi u sustavima za kontrolu inačica (2)

Pojam	Definicija
<i>Head</i> (vršna inačica)	Najnovija inačica u svakoj grani. <i>Head</i> se upotrebljava za najnoviju glavnu inačicu.
<i>Import</i> (uvoz)	Proces uvoza lokalne strukture kazala u središnji repozitorij po prvi put (nije radna kopija).
<i>Merge</i> (spajanje)	Spajanje dviju ili više inačica skupa datoteka u jedinstvenu inačicu.
<i>Repository</i> (središnji rezitorij)	Mjesto (na poslužitelju) na kojem se nalaze sve datoteke i popratni meta-podaci.
<i>Resolve</i> (razrješavanje)	Razrješavanje sukoba koji nastaje kada više korisnika istovremeno pokuša unijeti promjene u isti dokument.
<i>Trunk</i> (osnovna razvojna linija)	Osnovna ili glavna linija razvoja.
<i>Update</i> (osvježavanje)	Osvježavanje lokalne radne kopije s promjenama koje su nastale u središnjem repozitoriju.

Primjer grafa kontrole inačica s osnovnom linijom i grananjima.



Vrste sustava za kontrolu inačica programske potpore

- Danas postoji mnogo različitih sustava za kontrolu inačica programske potpore.
- Najčešće korišteni:
 - Apache Subversion/Tortoise SVN
 - Git
- Nisu isti. Osim što se razlikuju po tipu licence – komercijalni i sustavi otvorenog koda, prije svega se razlikuju po način rada: lokalnosti pristupa te centraliziranosti, tj. postojanju središnjeg rezervorija.

Lokalizirani model

- **Lokalizirani model** (engl. *Local data model*) osnovni je tip sustava za kontrolu inačica.
- Pohrana i pristup podacima ograničeni su na jedno računalo na kojem su također sadržani podaci o ranijim promjenama.
- Promjene se prate za svaku datoteku zasebno, što znači da nije moguće odjednom raditi s cijelim skupom datoteka (npr. projektom). Kao dio GNU projekta razvijen je RCS sustav (*Revision Control System*) koji je kasnije poslužio kao okosnica za razvoj naprednijih sustava poput CVS-a i SVN-a.



Model klijent-poslužitelj

- **Model klijent-poslužitelj** podrazumijeva postojanje središnjeg repozitorija na poslužiteljskom računalu čije radne inačice korisnici pohranjuju na svojim računalima i u određenom trenutku sinkroniziraju sa središnjim repozitorijem.
- Jedan od prvih sustava otvorenog koda koji je implementirao ovaj model bio je CVS (*Concurrent Versions System*). Na temelju njega se kasnije razvio Apache SVN koji se i danas koristi.



Apache Subversion

- Apache Subversion (SVN) je jedan od najpoznatnijih sustava za kontrolu inačica programske potpore.
- Nasljednik je ranijeg CVS sustava, a najznačajnija unaprjeđenja su u vidu potpune atomarnosti operacije *commit*, zaključavanja datoteka u slučaju da više korisnika pokuša istovremeno raditi izmjene, dohvaćanje *readonly* inačice središnjeg rezervorija i dr.

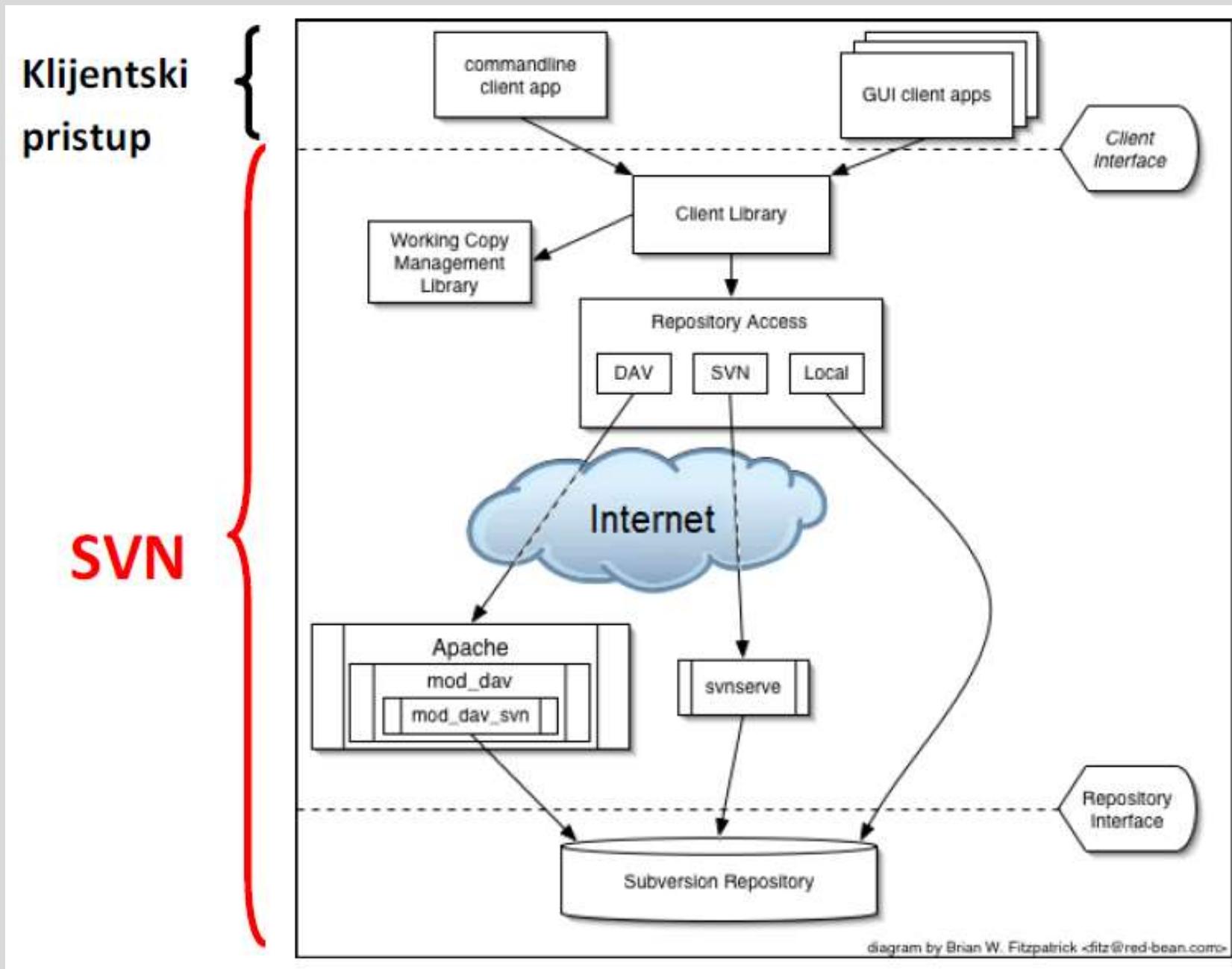
Najčešće korištene operacije

- Prilikom korištenja SVN-sustava, dvije najčešće korištene operacije su ***Update***, ***Commit*** i ***Checkout***.
- ***Update*** mijenja lokalnu radnu kopiju kako bi bila identična stanju središnjeg repozitorija.
- ***Commit*** izvozi u središnji repozitorij promjene nastale u lokalnoj radnoj kopiji.
- Prilikom stvaranja lokalne radne kopije (prvo dohvaćanje sadržaja središnjeg repozitorija) koristi se operacija ***Checkout***.

Operacije *Revert* i *Resolve*

- U slučaju sukoba inačica na kojima je istovremeno radilo više korisnika, sukob se može riješiti **odbacivanjem vlastitih promjena** – operacija ***Revert*** – ili se ide u **razrješavanje spora** – operacija ***Resolve*** – u kojem se odabire koja će se inačica zadržati (base, mine, full, working).

Arhitektura SVN-a



SVN klijenti

- Budući da Apache SVN podržava isključivo naredbeno-linijski način rada, razvijeni su posebni klijenti poput Tortoise SVN-a koji imaju integrirano grafičko korisničko sučelje.
 - URL: <http://tortoisessvn.net/>



Raspodijeljeni način rada

- U **raspodijeljenom modelu** (engl. *distributed model*) ne postoji jedinstveni središnji repozitorij s kojim se klijenti sinkroniziraju. **Svaka radna kopija repozitorija predstavlja repozitorij za sebe.**
- Tada su svi postojeći repozitoriji su ravnopravni, a korisnici mogu međusobno usklađivati repozitorije prema načelu ***peer-to-peer***.
- Glavni predstavnici ovakvog modela su sustavi otvorenog koda Git i Mercurial.

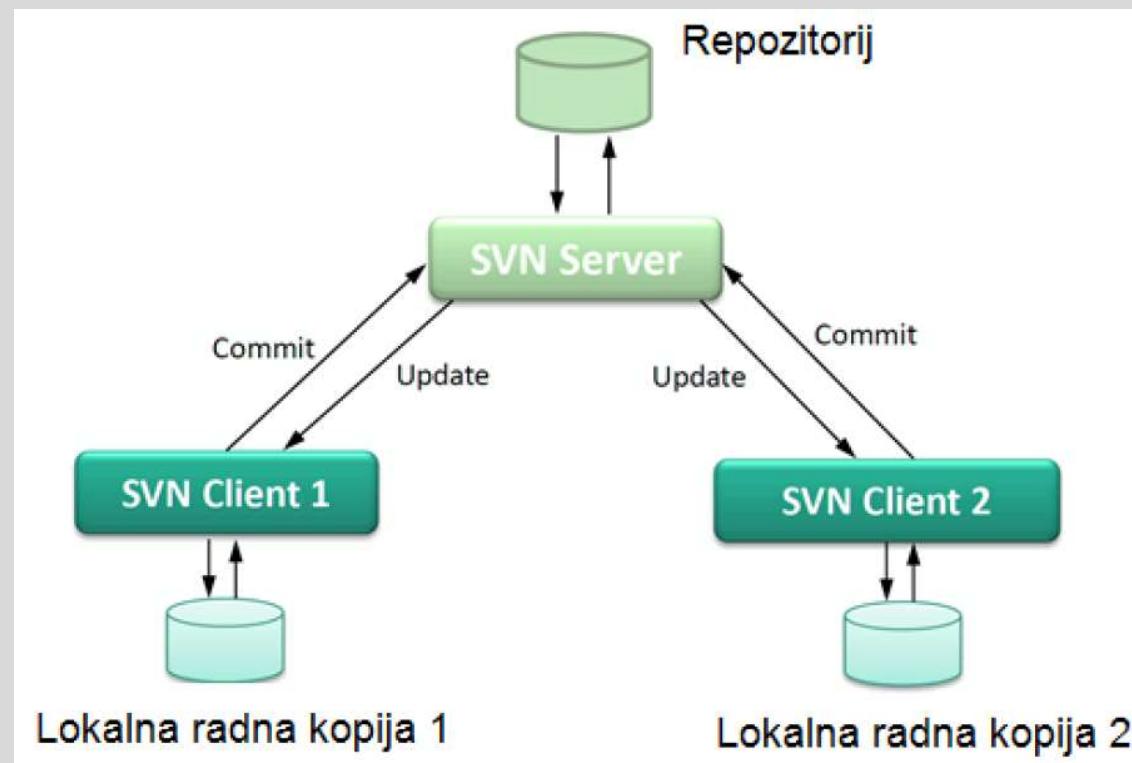
Git



- Git je danas vjerojatno najrašireniji sustav otvorenog koda za kontrolu inačica programske potpore.
- U odnosu na SVN znatno je složeniji za savladavanje i korištenje, ali nudi dodatne funkcionalnosti.
- Git-ova raspodijeljena arhitektura omogućava istovremeno postojanje više smjerova razvoja programske potpore koji se mogu, ali i ne moraju objediniti.

Arhitektura Git-a

- Arhitektura sustava Git podrazumijeva rad s dva repozitorija – lokalnim i udaljenim.
- Rad s lokalnim repozitorijem nalikuje na rad sa središnjim repozitorijem u SVN-u.



Najvažnije operacije

- Najvažnije operacije Git-a:
 - operacija *Commit* zapisuje promjene u lokalni repozitorij.
 - operacija *Checkout* osvježava radnu kopiju lokalnog direktorija (operacija *Update* pod tim imenom ne postoji).
 - Lokalni repozitorij usklađuje se s udaljenim repozitorijem operacijama *Push* i *Fetch*
 - Ako se želi direktno osvježiti lokalna radna kopija umjesto *Fetch* + *Merge* može se direktno izvršiti operacija *Pull/Rebase*.
 - Prilikom prvog dohvatanja sadržaja udaljenog direktorija koristi se operacija *Clone*.

Kvazi-repozitorij Index

- Dodatna mogućnost koju nudi Git je Index – lokalni kvazi-repozitorij u koji se pohranjuju nove i izmijenjene datoteke prije sinkronizacije s pravim lokalnim repozitorijem (operacije *Add* i *Checkout*).

Sustav *Git*

- Repozitorij – mjesto pohrane promjena datoteka
- Raspodijeljeni razvoj – na više mjesta
- Svaka mapa/kazalo – potpuni repozitorij
- → potpuni datotečni sustav
- Najviše korišteni alat za upravljanje inačicama datoteka
- Klijenti za standardna razvojna okruženja
- Različita ostvarenja *Git* poslužitelja:
 - → *GitLab* – web hosting usluga za *Git* temeljene projekte

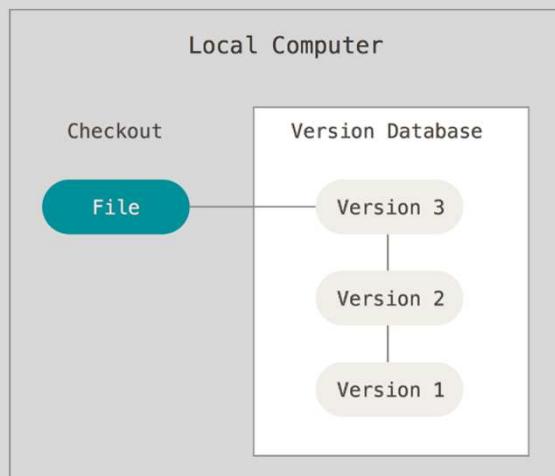
Sustav *Git* + *GitLab*

- Instalacija i dokumentacija *Git*-a dostupne na:
 - <https://git-scm.com>
- *GitLab* pristup na:
 - <https://gitlab.com>

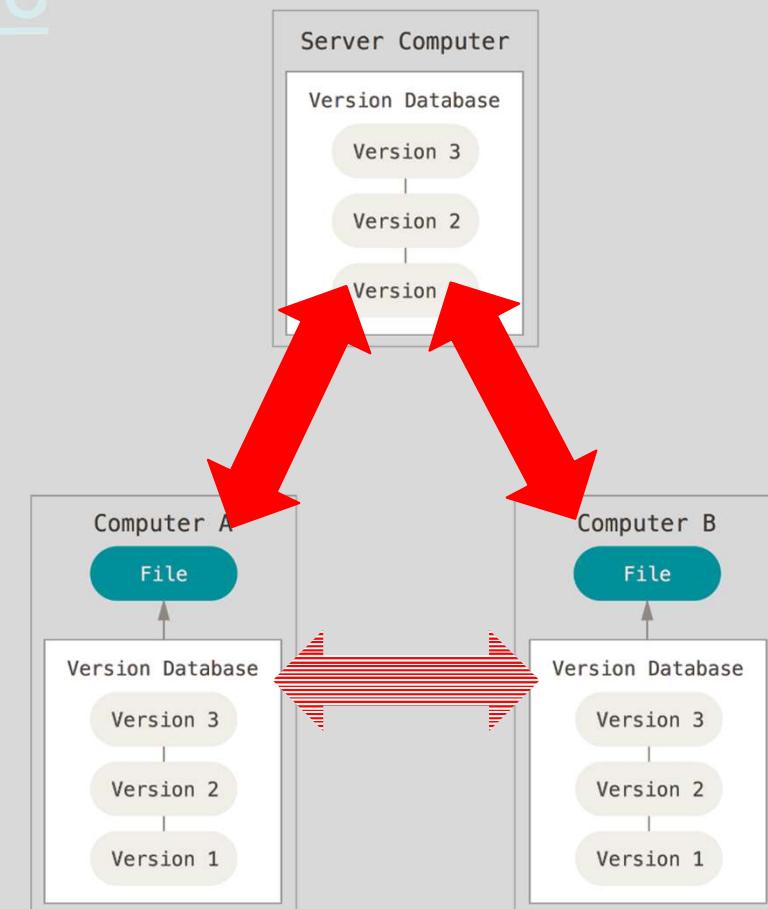
Sustav *Git* - Ideja

- <https://git-scm.com/doc>

Lokalno upravljanje:

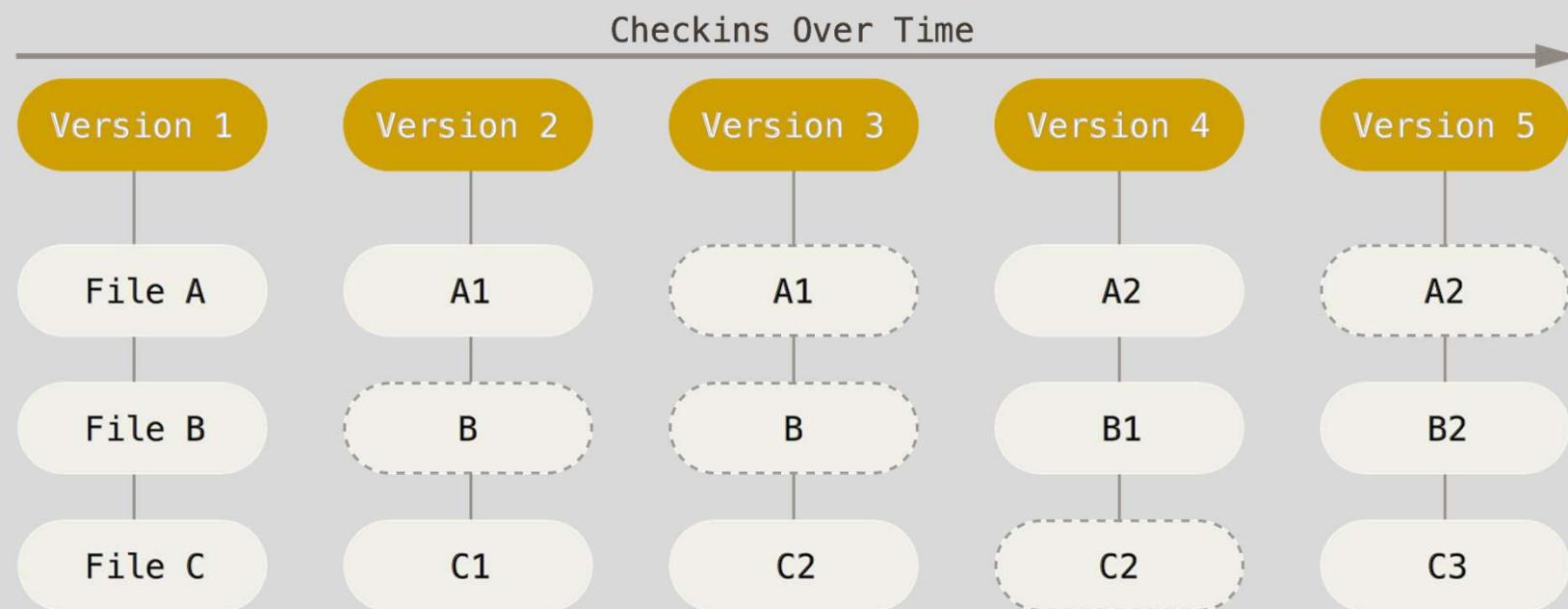


Raspodijeljeno upravljanje:



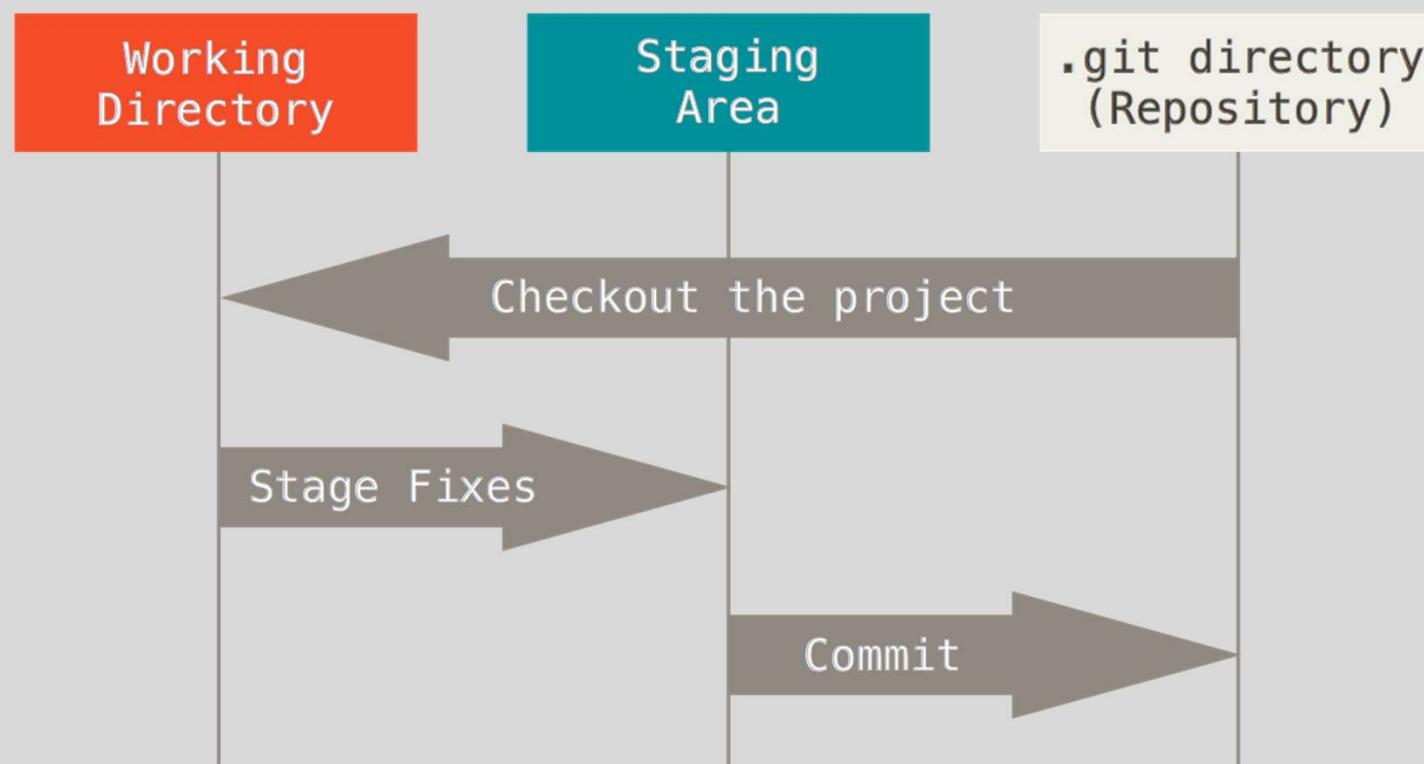
Sustav Git - Stanja

- <https://git-scm.com/doc>

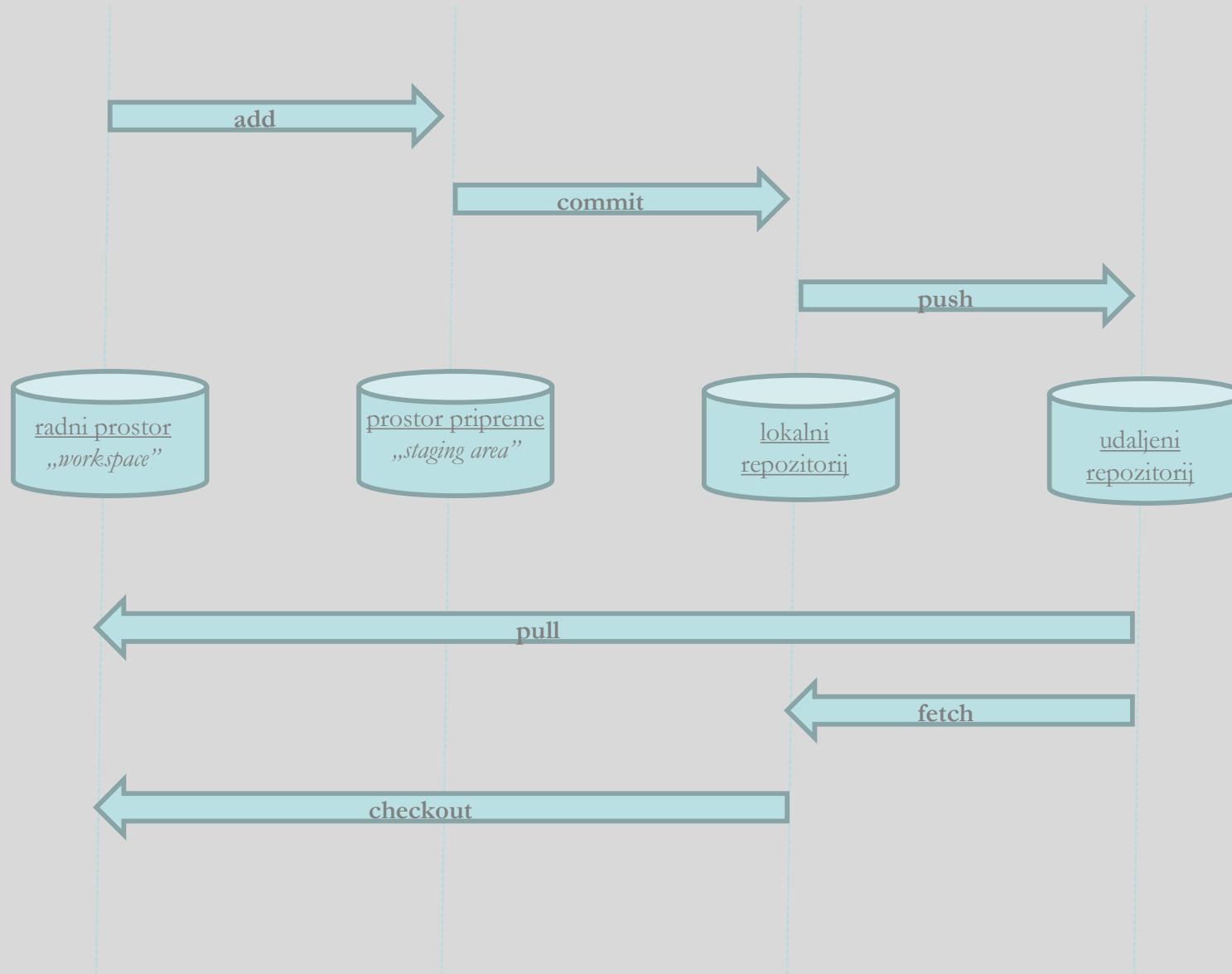


Sustav *Git* – Lokalni rezervorij

- <https://git-scm.com/doc>



Sustav *Git* – Središnji repozitorij

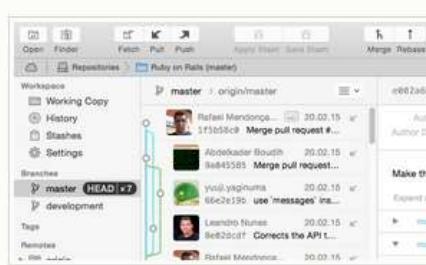


Git GUI klijenti



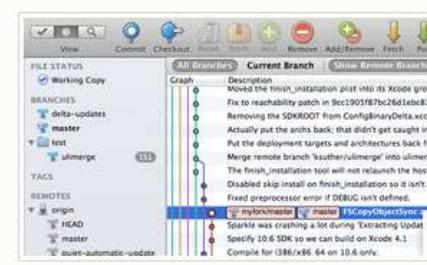
GitHub for Mac

Platforms: Mac
Price: Free



Tower

Platforms: Mac
Price: \$69/user (Free 30 day trial)



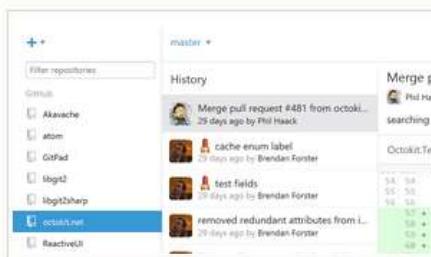
SourceTree

Platforms: Mac, Windows
Price: Free



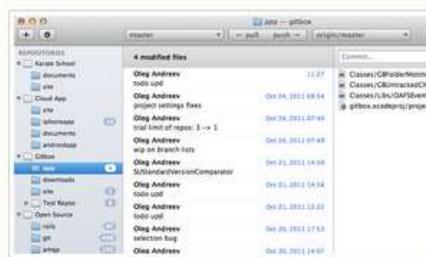
git-cola

Platforms: Windows, Mac, Linux
Price: Free



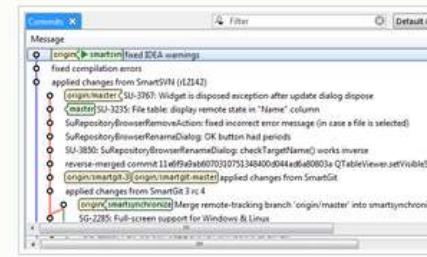
GitHub for Windows

Platforms: Windows
Price: Free



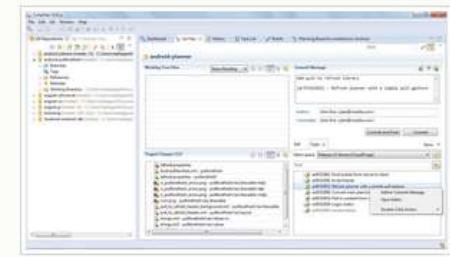
Gitbox

Platforms: Mac
Price: \$14.99



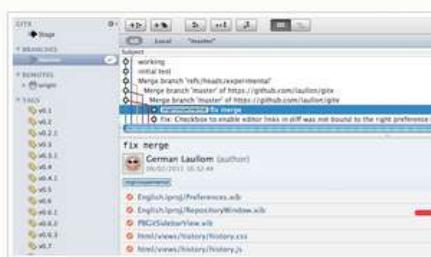
SmartGit

Platforms: Windows, Mac, Linux
Price: \$79/user / Free for non-commercial use



GitEye

Platforms: Windows, Mac, Linux
Price: Free



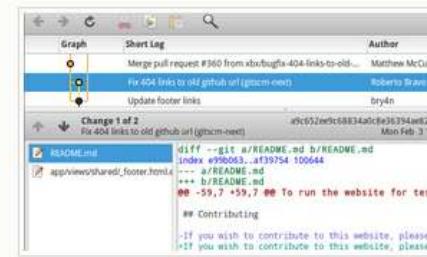
GitX-dev

Platforms: Mac
Price: Free



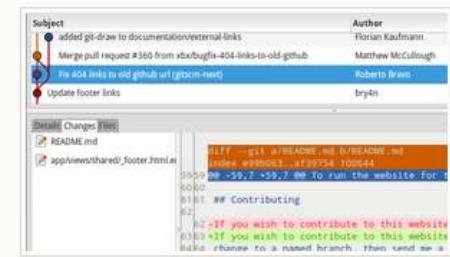
Git Extensions

Platforms: Windows
Price: Free



giggle

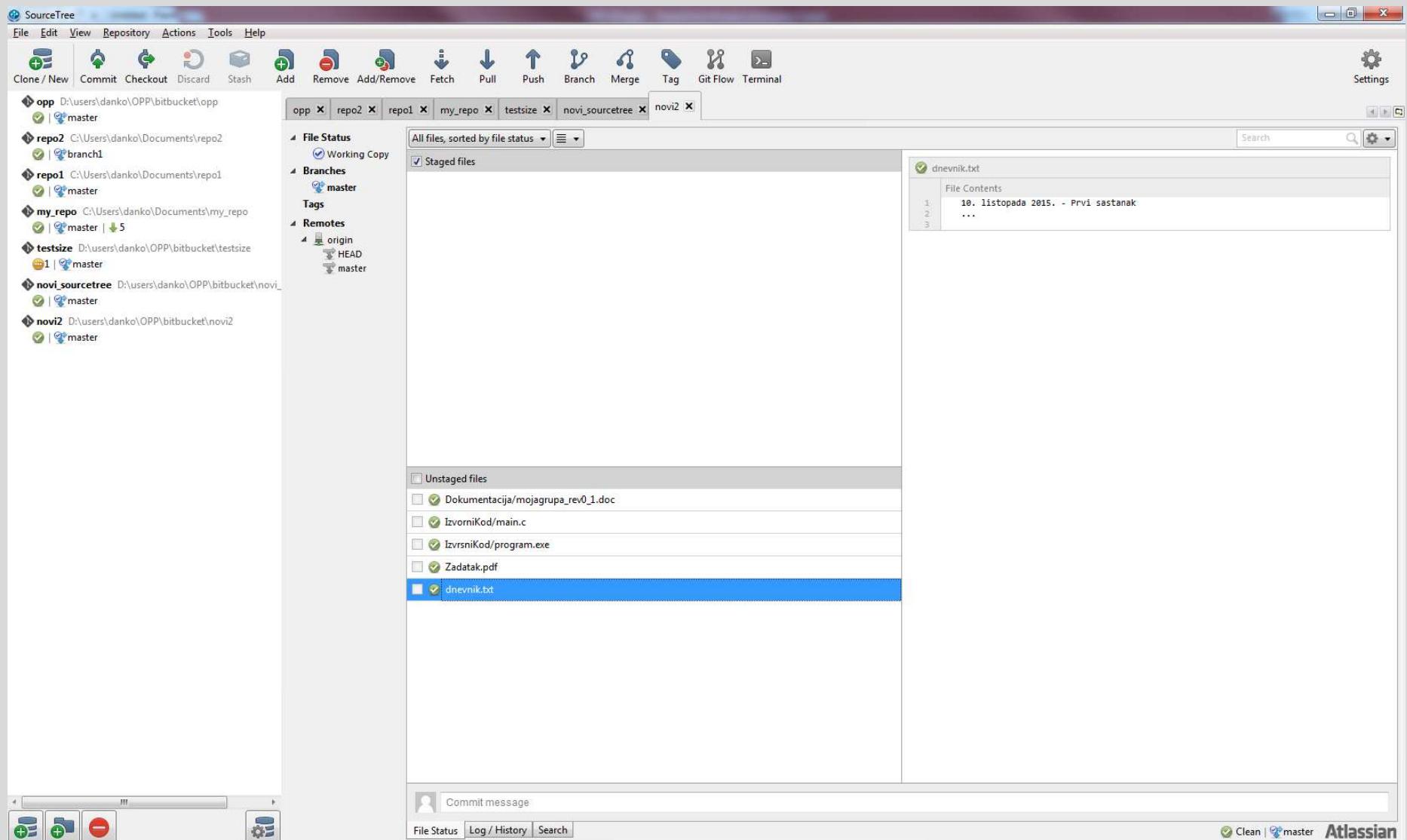
Platforms: Linux
Price: Free



git

Platforms: Linux
Price: Free

Git – SourceTree klijent



Integracija s razvojnim okruženjem

GUI Git Clients

All GUI Clients

All Git GUI Clients from all companies are compatible with GitLab.

<http://git-scm.com/downloads/guis>

Git Tower

Easy version control in a beautiful, efficient, and powerful app for Mac OS X.

www.git-tower.com

Eclipse

Eclipse has the Egit Team provider that also supports GitLab. [Eclipse Git Team Provider](#)
[Working with remote repositories](#)

JetBrains

Lets you interact with gitlab from within your IDE.

Visual Studio

The Visual Studio Tools for Git is an extension for Team Explorer that provides source control integration for Git.

[visualstudiogallery](http://visualstudiogallery.msdn.microsoft.com/2d4d0e1c-0e0d-400d-a6c1-228d008d04a1)

GitKraken

GitKraken is a visual git client. Please note that it requires you to sign up with a working email.

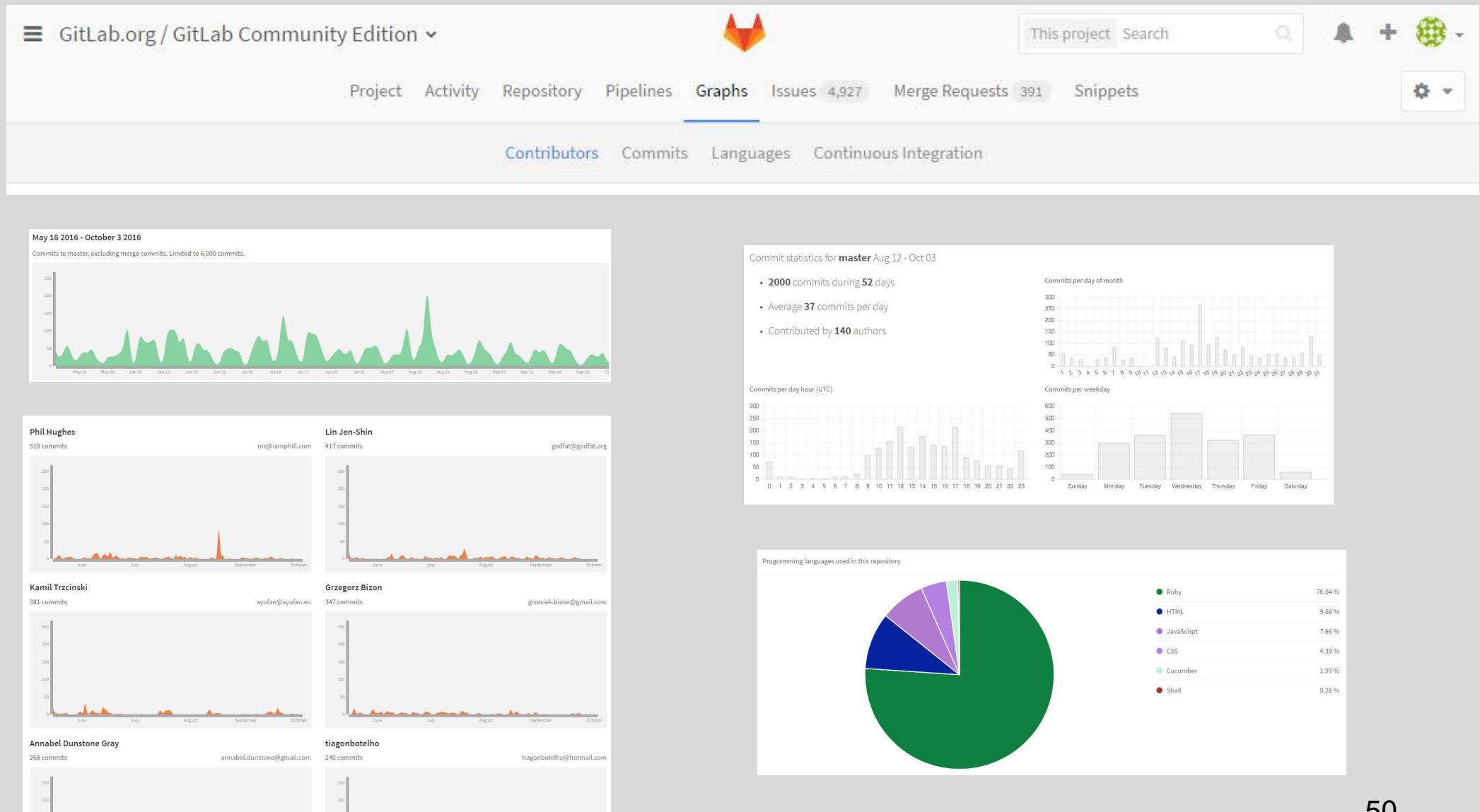
GitKraken.com

PhpStorm

This IDE for the PHP programming language has a GitLab plugin.

[Plugin on the Jetbrains site](#)

Pregled GitLab statistike



REFERENCE I LITERATURA

- Predavanja ovog predmeta
- GNU Project, RCS,
<https://www.gnu.org/software/rcs/rcs.html>, pristupljeno 03/2015.
- Apache, Apache Subversion, <http://subversion.apache.org/>, pristupljeno 03/2015.
- Git, <http://git-scm.com/>, pristupljeno 03/2015.
- Nastavni materijali kolegija Oblikovanje programske potpore, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu.



TEHNIČKO VELEUČILIŠTE U ZAGREBU
POLYTECHNICUM ZAGRABIENSE

Objektno orijentirani razvoj programa

ISVU: 130938/19881

Dr. sc. Danko Ivošević, dipl. ing.
Predavač

Akademska godina 2021./2022.
Ljetni semestar

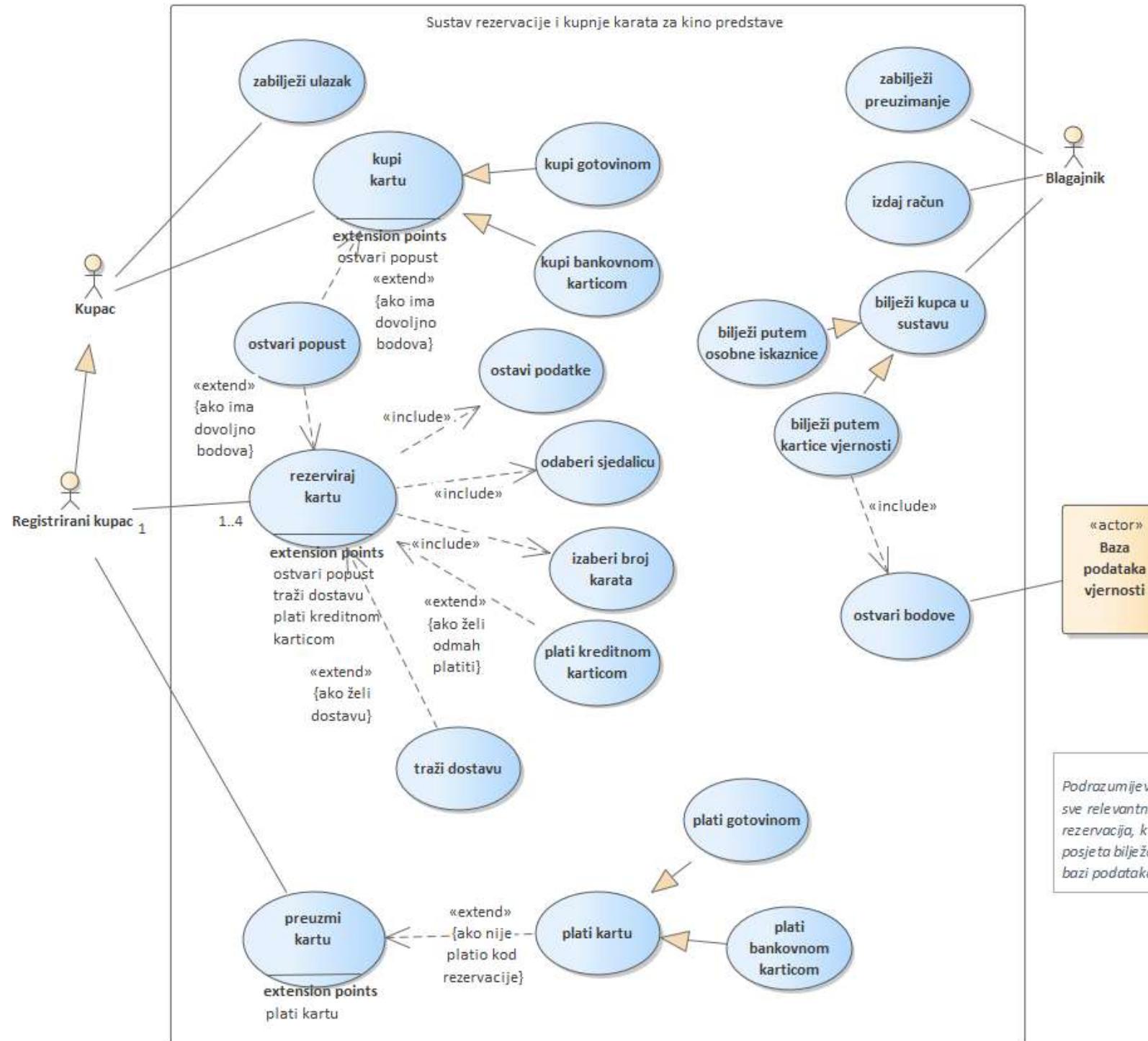
UML modeliranje

Primjeri zadataka

Dijagram obrazaca uporabe

Dijagramom obrazaca uporabe modelirajte sustav rezervacije i kupnje karata/ulaznica za kino predstave. Bilo koji kupac može kupiti kartu za predstavu na ulazu u kino dvoranu na način da plati i preuzme kartu na blagajni. Registrirani kupac može rezervirati kartu putem usluge web sučelja. Pri rezervaciji karte on najprije mora ostaviti osobne podatke, odabrati brojeve sjedalica, potvrditi broj karata s ukupnom cijenom. Pritom može odmah i platiti kartu kreditnom karticom i tražiti dostavu karte na kućnu adresu. Inače kartu može platiti kartu kod podizanja na blagajni na licu mjesta. Karta se na blagajni može platiti gotovinom ili bankovnom karticom. Može rezervirati najviše četiri karte odjednom. Na blagajni dvorane radi blagajnik koji bilježi da je karta preuzeta od strane kupca te kupcu uručuje račun. Ako kupac nije radio rezervaciju preko weba, onda blagajnik provlači njegovu osobnu iskaznicu kroz čitač koji bilježi njegove podatke u sustavu. Ako posjetitelj ima karticu vjernosti onda provlači tu karticu umjesto osobne iskaznice gdje mu se bilježe bodovi vjernosti pri kupnji ulaznica. Bodovi se bilježe u posebnoj bazi podataka članova kluba vjernosti. Na temelju skupljenih bodova posjetitelj može ostvariti popust kod rezervacije odnosno kupnje karte za predstavu. Pri ulasku u dvoranu kupci provlače karte kroz čitač koji bilježi ulazak posjetitelja u dvoranu.

Podrazumijeva se da sustav kino multipleksa koji prikazuje predstave ima bazu podataka u koju se bilježe sve rezervacije, kupovine i posjete predstavama od strane korisnika.



Podrazumijeva se da se sve relevantne akcije rezervacija, kupnji i posjeta bilježe u glavnoj bazi podataka.

Dijagram razreda

Kino multipleks ima više lokacija. Lokacija je određena s nazivom i adresom.

Na svakoj lokaciji ima više dvorana. Svaka dvorana ima svoj naziv, broj sjedećih mesta i raspored predstava. Predstava je obilježena cijenom, satom i filmom u to vrijeme.

Film ima naslov, ime glavnog glumca i glumice, trajanje, žanr i oznaku primjerenosti ili zabrane gledanja ovisno o minimalnoj dobi gledatelja. Oznaka primjerenosti može: U12, U15 ili U18.

Za svaku predstavu se izdaje neka količina ulaznica. Svaka ulaznica ima oznaku sjedećeg mesta: kategoriju, red, broj sjedalice.

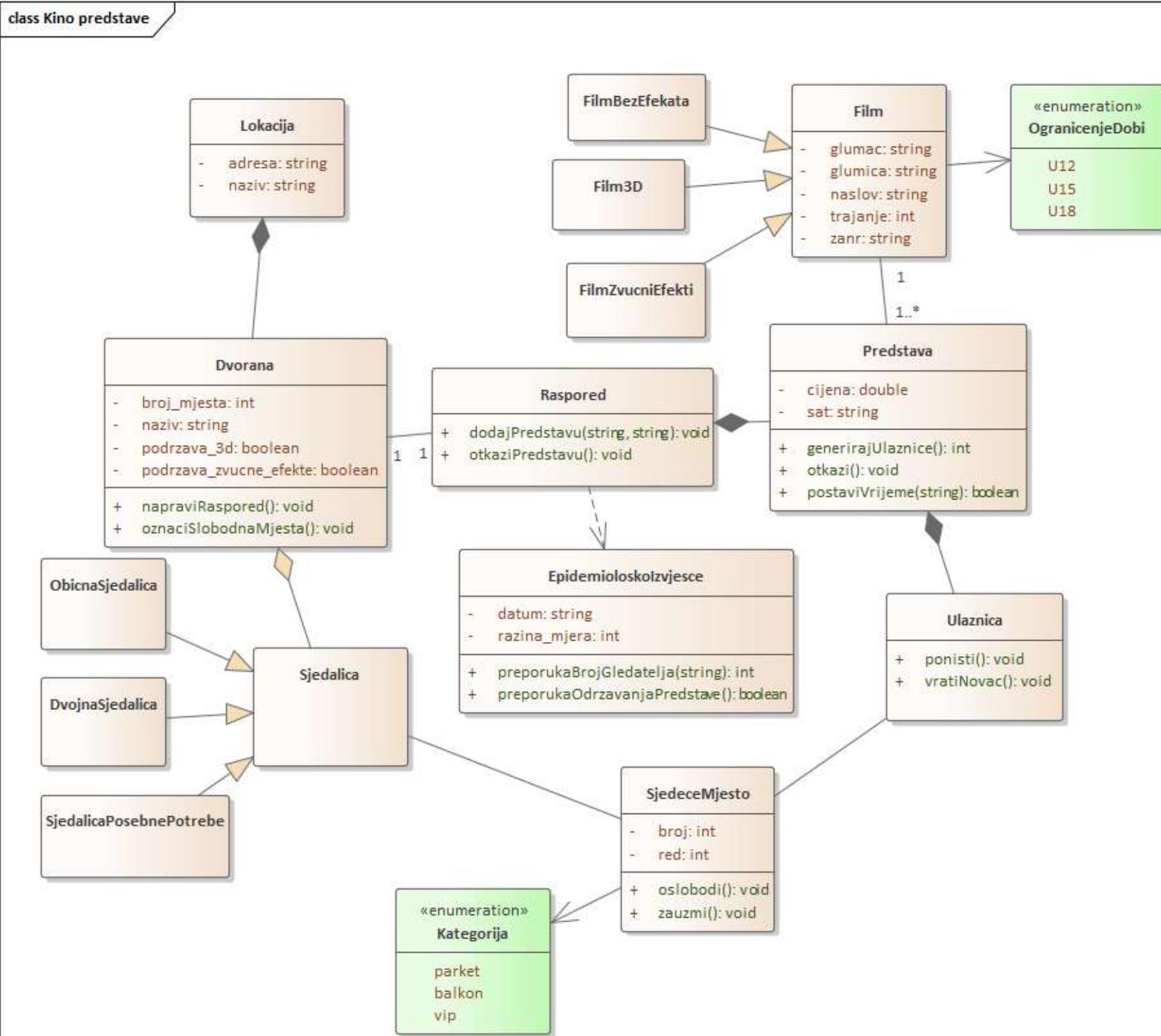
Kod dodavanja predstave u raspored generira se neki broj ulaznica po predstavi a vrijeme prikazivanja se može i promijeniti.

Ako se predstava otkaže zbog epidemiološke ili koje druge situacije ulaznice više ne vrijede, a novci se vraćaju te se poništavaju sve generirane ulaznice. Sjedeća mjesta se zauzimaju kod kupnje ili rezervacije ulaznica za pojedinu predstavu.

Dvorana ima neki broj sjedalica. Sjedalice su mobilne i mogu se premještati iz dvorane u dvoranu ovisno o potrebama. Dvorana je podijeljena prema kategorijama sjedenja kojih ima više. Kategorija može biti parket, balkon ili vip. Ako se dvorana prestane koristiti kategorije ostaju jer nisu ovisne o dvorani. Poseban entitet je epidemiološko izvješće o kojem ovisi svaka predstava (hoće li se održati ili neće). Izvješće sadrži datum i razinu epidemioloških mjera (cjelobrojna vrijednost) koji su na snazi za navedeni datum. Za svako izvješće mogu se dobiti posebne informacije o preporuci o održavanju predstave i o preporuci broja posjetitelja ovisno o dvorani. U tom slučaju se za pojedinu dvoranu mogu označiti odnosno napraviti raspored dozvoljenih sjedalica za korištenje.

Sama sjedalica po vrsti može biti obična, dvojna ili za osobe s posebnim potrebama.

Filmovi koji se prikazuju prema načinu projekcije mogu biti obični, 3D ili sa posebnim zvučnim efektima što već ovisi i o dvorani, tj. koliko pojedina dvorana podržava pojedine načine projekcije.



Hvala na pažnji!

Pitanja?