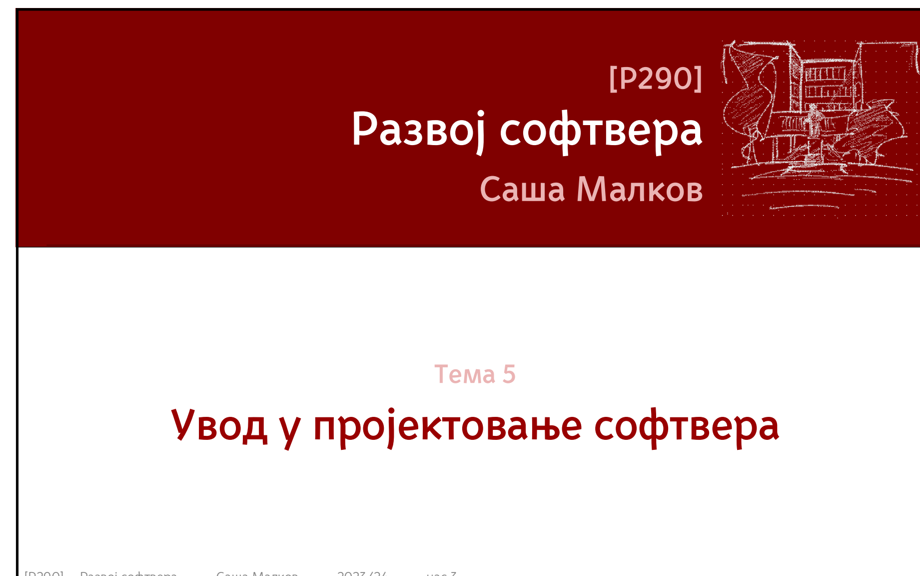
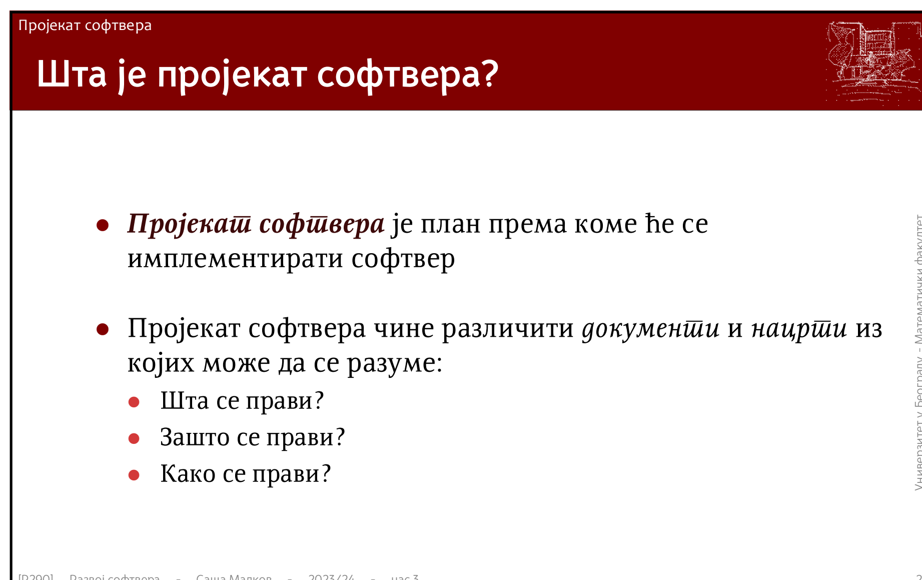
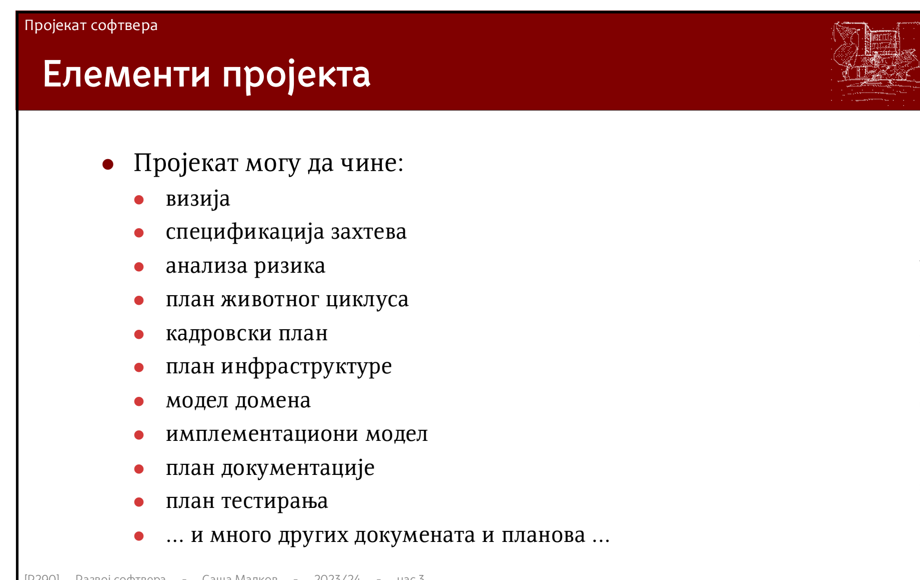
Развој софтвера 3
Саша Малков
Универзитет у Београду
Математички факултет
2023/2024


[P290]
Развој софтвера
Саша Малков
Тема 5
Увод у пројектовање софтвера
[P290] Развој софтвера - Саша Малков - 2023/24 - час 3
1


Пројекат софтвера
Шта је пројекат софтвера?
Пројекат софтвера је план према коме ће се имплементирати софтвер
Пројекат софтвера чине различити документи и нацрти из којих може да се разуме:
Шта се прави?
Зашто се прави?
Како се прави?
Универзитет у Београду - Математички факултет
[P290] Развој софтвера - Саша Малков - 2023/24 - час 3
2


Пројекат софтвера
Елементи пројекта
Пројекат могу да чине:
визија
спецификација захтева
анализа ризика
план животног циклуса
кадровски план
план инфраструктуре
модел домена
имплементациони модел
план документације
план тестирања
... и много других докумената и планова ...
Универзитет у Београду - Математички факултет
[P290] Развој софтвера - Саша Малков - 2023/24 - час 3
3

Пројекат и методологија

- Развојна методологија прописује или бар утиче на то
  - Како се гледа на пројектовање?
  - Како се бирају основне полазне тачке?
  - Које врсте докумената и нацрта се праве у ком тренутку?
- Поделићемо све методологије на три групе:
  - Класичне
  - Објектно-оријентисане
  - Агилне

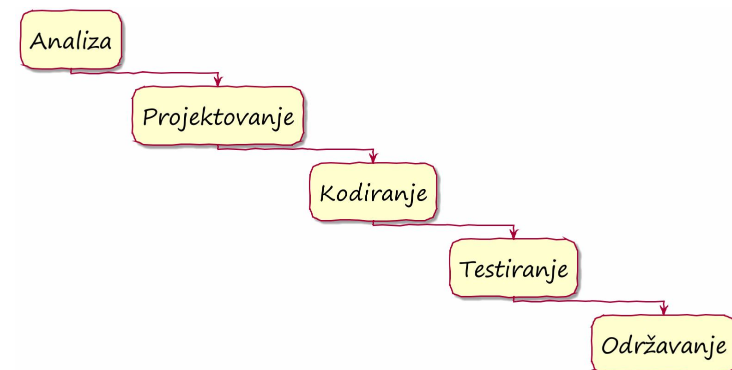
Пројектовање у класичним методологијама

- ***Класичне*** методологије се фокусирају на:
  - стриктно обликовање животног циклуса пројекта
  - описивање пословних процеса
  - структурирање елемената пројекта и развојног процеса

Животни циклус у класичним мет. (1)

- Животни циклус у класичним мет. почива на
  - препознавању различитих развојних фаза
  - строгом раздвајању фаза
  - линеарном проласку кроз фазе *у једном смеру*
- Тзв. животни циклус по *моделу водопада*
  - Постоје и различита унапређења и модификације

Животни циклус у класичним мет. (2)



Однос животног циклуса и специјалности

- Животни циклус није случајно обликован са строго раздвојеним фазама
- Класичне методологије претпостављају релативно уске специјалности развијалаца софтвера
- У свакој фази учествују различите специјалности
  - срећу се различити називи специјалности, али је важно препознавање релативно строгог повезивања специјалности са одговарајућим фазама развоја
  - На пример:
    - анализом система се баве *систем-аналитичари*
    - пројектовањем се баве *програмери аналитичари* или системски програмери
    - кодирањем се баве *"обични" програмери*

Универзитет у Београду - Математички факултет

Улога процеса у класичним методологијама

- Фокусирање на процесе
  - Полазне основе пројекта су препознати кључни процеси у пословном домену
  - Цео пројекат се темељи на моделирању процеса
    - има сличности са моделирањем алгоритама блок-дијаграмима
  - Све остало је надградња
  - То посебно важи за *процесне мет.*
  - Касније *структурне мет.* и *комбиноване мет.* део пажње у раним фазама пројектовања пребацују на структуру, али процеси и даље остају кључни

Универзитет у Београду - Математички факултет

Пројектовање у ОО методологијама

- ***Објектно оријентисане*** методологије:
  - настају од почетка 1980-их
  - пребацују фокус са процеса на објекте
  - углавном остаје стриктан животни циклус
- Под објектима се заправо мисли на класе
  - функционалност је испољена преко интерфејса
  - енкапсулирани су структура и имплементација

Универзитет у Београду - Математички факултет

Однос објеката и процеса

- Обично се каже да су објекти *стабилнији* од процеса
  - у простору пословног домена процеси имају већу тенденцију настајања и нестајања него класе
  - стварање, нестајање и мењање процеса се пресликава у промену понашања класа
  - такве промене на класама су обично локалнијег карактера
    - између осталог и због *енкапсулације*
- Због тога су пројекти засновани на класама стабилнији
  - постојанији у времену
  - лакше се прилагођавају променама у домену

Универзитет у Београду - Математички факултет

Пројектовање у агилним методологијама

- ***Агилне*** методологије:
  - настају од средине 1990-их
  - практично бришу оштру границу између различитих фаза развоја
  - стварају потребу да већина развијалаца буде оспособљена и за пројектовање и за имплементирање
  - углавном ортогоналне у односу на фокус
    - али најчешће ОО

Агилни животни циклус

- Практично се бришу оштре границе између различитих фаза развоја, па и између пројектовања и имплементирања
- Раније релативно слабо заступљена специјалност *програмера аналитичара* постаје далеко заступљенија у развојним тимовима
- Агилним мет. ћемо се посветити за неколико часова
- За сада је важно да уочимо да је промена у односу према животном циклусу тесно повезана са променама у односу према специјалностима развијалаца

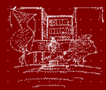
Архитектура и дизајн

- Уместо термина *пројекат софтвера* веома често се користе термини
  - дизајн софтвера и
  - архитектура софтвера

Шта је дизајн софтвера?

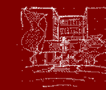
- ***Дизајн софтвера*** је структура софтверског система
- ...али и
  - дисциплина развоја софтвера која се бави структуром софтверског система
  - документација која описује структуру софтверског система

Шта чини дизајн софтвера?



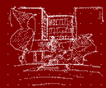
- Структуру софтверског система чине сви његови елементи
- Зато и дизајн софтвера чине, или на њега утичу, сви његови елементи
  - компоненте
  - имплементација
  - структуре података
  - инфраструктура
  - помоћни алати
  - ...и све друго што је део развојног процеса

Шта је архитектура софтвера?



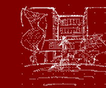
- ***Архитектура софтвера*** је структура софтверског система, посматрана на релативно високом нивоу
- ...али и
  - дисциплина развоја софтвера која се бави прављењем архитектуре софтверског система
  - документација која описује архитектуру софтверског система

Однос архитектуре и дизајна (1)



- У чему је разлика између архитектуре и дизајна?
- Дизајн софтвера је структура софтверског система
- Архитектура софтвера је структура софтверског система, ***посматрана на релативно високом нивоу***

Однос архитектуре и дизајна (2)



- Свака архитектура је дизајн, али није сваки дизајн архитектура
- *“Архитектура представља значајне одлуке о дизајну које дају облик систему, где је значајност одређена ценом прављења измена”*
  - *Grady Booch*

Однос архитектуре и дизајна (3)

- При прављењу архитектуре софтверског система предузимају се неке специфичне активности и узимају у обзир специфични аспекти проблема
  - велики значај ширине посматрања
  - мањи значај појединости

Где су границе архитектуре и дизајна?

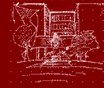
- Границе архитектуре и дизајна не могу да се једнозначно поставе
- Најважнији критеријуми су:
  - степен функционалне декомпозиције и
  - процењен значај који би детаљније пројектовање имало на цео систем
- Не смемо да занемаримо ниједан од ових критеријума

Основни поступци при пројектовању

- Два основна поступка при пројектовању софтвера су:
  - апстраховање и
  - декомпоновање

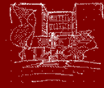
Апстраховање

- Апстраховање је уопштавање карактеристика посматраног проблема и елемената решења
  - тежимо да решење одвојимо од конкретних оквира и специфичности
  - тежимо да решење буде на релативно високом концептуалном нивоу
    - да може да се примени на шири скуп сродних проблема.
  - занемарујемо појединости
- Каже се и да је апстракција *уопштено решење*



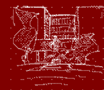
Видови апстраховања

- Најважнији видови апстраховања су:
  - класификација
  - генерализација
  - обликовање сервиса
  - компоновање



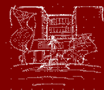
Декомпозиција

- Декомпоновање је
  - постепено прецизирање система кроз препознавање мањих целина (компоненти) које га чине
  - рекурзивном применом декомпозиције се долази до физичког дизајна
- Представља везу између различитих нивоа апстракције
  - декомпозиција апстрактног система води његовој конкретнијој структури
- Има сличности са механизмом дедукције
  - посматрањем и разлагањем општег случаја тежимо да направимо конкретнији (прецизнији) модел
- Каже се и да је декомпозиција *разложено решење*



Врсте декомпозиције

- Функционална декомпозиција
  - подела целине на компоненте са препознатим и раздвојеним функцијама
- Декомпозиција према променљивости
  - препознајемо тачке и осе променљивости
  - тежимо да тачке променљивости раздвојимо а да осе групишемо
- Логичка декомпозиција
  - подела целине на скупове делова које је разумно развијати заједно



Апстракција и декомпозиција

- Добар дизајн настаје усклађеном применом апстраховања и декомпоновања
- Апстраховањем се добија општост решења
- Декомпозицијом се добијају елементи структуре решења и односи међу њима

Поступак пројектовања

- На почетку се тежи да се сагледају *основне идеје о систему*
  - велика ширина посматрања – цео пројекат
  - релативно низак ниво детаљности
  - апстраховање
- Затим се систем дели на *основне структурне целине*
  - функционална декомпозиција
  - у комбинацији са декомпозицијом према променљивости
- Затим се посматрање сужава на појединачне препознате делове и поступак се итеративно понавља...
  - ...све док може да се врши даља функционална декомпозиција
- На крају се врши детаљно структурно моделирање на нивоу класа

Универзитет у Београду - Математички факултет

Поступак пројектовања (2)

- Резултати и међурезултати се деле по нивоима
  - на највишем нивоу имамо само један елемент – цео систем
  - на следећем имамо његове основне функционалне делове – компоненте
  - и тако даље, све док може да се врши функционална декомпозиција (уз разматрање тачака и оса променљивости)
  - на крају се свака компонента даље декомпонује на своје структурне целине – класе
- По потреби се више пута пролази од највишег до најнижег нивоа, док се не отклоне све уочене слабости
  - критички посматрамо оно што смо добили
  - можда нека од декомпозиција није добра
  - можда има простора да апстрахујемо сличне компоненте

Универзитет у Београду - Математички факултет

Поступак пројектовања (3)

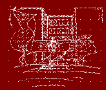
- На крају имамо детаљну структуру софтвера, коју по нивоима чине
  - компоненте и њихови односи и
  - класе, њихове структуре и односи
- Паралелно са препознавањем структурних елемената софтвера, препознајемо и развојне задатке

Универзитет у Београду - Математички факултет

“Све” је дизајнирање

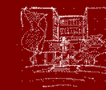
- Дизајн софтвера се прави и мења кроз
  - планирање (пројектовање)
  - писање изворног кода
  - тестирање
  - дебаговање
  - ...
- Практично сваки аспект развоја софтвера ***мења*** дизајн софтвера или ***утиче*** на дизајн софтвера

Универзитет у Београду - Математички факултет



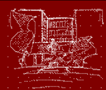
Да ли правити пројекат пре кода?

- Ако је *све* дизајнирање, да ли се онда исплати дизајнирати више пута исти софтвер?
  - најпре кроз планирање
  - па кроз кодирање
  - па тестирање
  - па дебаговање
  - ...
- "Вероватно је боље пустити оригиналне пројектанте да напишу оригинални код, него да неко други преводи њихов дизајн на програмски језик."
  - *Jack Reeves*



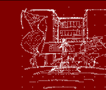
Како онда радити?

- "Пројекат пре кода"
  - даје јасну слику пројекта пре програмирања
  - омогућава програмерима да имају јаснију слику шта праве
  - нуди јасну и стабилну велику слику пројекта
  - често се мења током развоја, чак и потпуно
  - представља озбиљан али често неажуран вид документације
- "Пројекат је код"
  - искусни програмери могу да истовремено прилагоде и код пројектовању и пројекат начину кодирања
  - штеди се време на пројектовању и може брже да се започне са имплементацијом појединачних модула
  - може да доводи до отежаног повезивања компоненти
  - не постоји јасна велика слика пројекта у раним фазама развоја



Савремена пракса (1)

- Ако је део пројекта нижег нивоа, онда се ређе прави пре самог програмског кода
  - дизајн софтвера се све ређе прави и документује пре кодирања
  - ако се прави, онда само умерено детаљно
- Ако је део пројекта нижег нивоа, онда је неопходније његово пажљиво планирање и документовање пре кодирања
  - архитектура софтвера се најчешће прави и документује пре кодирања
  - прескакање архитектуре обично има скупе последице
- Граница између дизајна и архитектуре је обично ниво ***компоненте***
  - шта год то било у конкретном случају...



Савремена пракса (2)

- Планирају се
  - компоненте
  - њихови интерфејси
  - њихови међусобни односи
- То уобичајене спада у *архитектуру*
- Не планира се детаљно имплементација компоненти
- То уобичајено спада у *дизајн*

Савремена пракса (3)

- “Пројектовање” се као посебна фаза практично изједначава са “обликовањем архитектуре”
- На тај начин се избегавају највећи проблеми оба приступа:
  - добија се јасна велика слика декомпозиције система на компоненте и јасно се дефинишу њихови интерфејси
  - не губи се сувише времена на пројектовање појединости у раним фазама развоја, а које ће се касније често мењати
- Што је систем сложенији, то се архитектура прави на више различитих нивоа апстракције, тј. “пројектовање” мора да се ради детаљније

Универзитет у Београду - Математички факултет

Обавезе архитекте (1)

- Разумевање свих аспеката развоја
  - познавање, планирање и праћење развојног процеса
  - узимање у обзир свих практичних ограничења
- Разумевање сопствене улоге
  - преузимање одговорности за кључне одлуке
  - не залажење у (из угла архитектуре) небитне детаље

Универзитет у Београду - Математички факултет

Обавезе архитекте (2)

- Комуницирање са улагачима
  - архитектура мора да задовољава потребе улагача
- Извођење решења из пословних потреба
  - препознавање и разумевање потреба
  - архитектура је “слика” пословног окружења
- Утицање на захтеве
  - препознавање компромиса и предочавање улагачима
  - помагање улагачима да донесу исправне одлуке

Универзитет у Београду - Математички факултет

Обавезе архитекте (3)

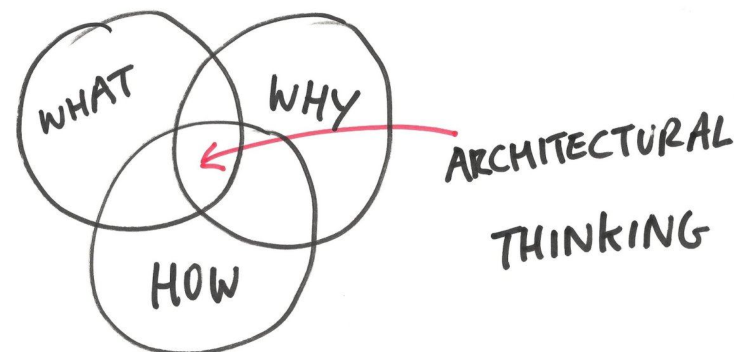
- Управљање ризицима и променама
  - итеративно унапређивање и прилагођавање архитектуре
- Поновљена употреба
  - употреба постојећих добара штеди време и новац
- Добро одмеравање учешћа
  - архитекта мора да донесе одлуке у његовом домену
  - не сме да изађе ван свог домена

Универзитет у Београду - Математички факултет

Обавезе архитекте (4)

- Прецизирање архитектуре према технолошким ограничењима
  - архитектура се увек имплементира у неком технолошком контексту
- Уважавање општих оквира
  - свака конкретна архитектура је само *део* ширег пословног контекста
  - она мора да се слаже са осталим елементима окружења

Аспекти архитектуре (1)



Peter Cripps

Аспекти архитектуре (2)

- Шта?
  - захтеви које софтвер мора да задовољи
- Зашто?
  - кључне одлуке на основу којих се обликује архитектура
- Како?
  - компоненте, интерфејси, односи
  - на нижем нивоу и дизајн и имплементација решења

Кључни утицаји на архитектуру

- Међусобно супротстављене три основне *силе*:
  - жеље
    - све оно што улагачи желе
  - изводљивост
    - све оно што технологија може да пружи
  - одрживост
    - све оно што у датим условима може да се постигне
    - време, новац и други ресурси
- Нешто мање важна и јака четврта *сила*:
  - естетика
    - сви они неформални (ирационални?) аспекти решења који га чине квалитетнијим

Универзитет у Београду - Математички факултет

Процењивање квалитета пројекта

- ***Квалитативно процењивање***
  - препознајемо добре и лоше карактеристике пројеката
  - сагледавамо и разматрамо карактеристике посматраног пројекта
  - на основу њих правимо квалитативну оцену пројекта
- ***Квантитативно процењивање***
  - дефинишемо и израчунавамо различите нумеричке оцене
  - сагледавамо и разматрамо такве оцене посматраног пројекта
  - тиме се бави област *метрике софтвера*, о чему ће бити речи на наредним часовима

Пожељене карактеристике софтвера (1)

- Основне пожељне карактеристике су:
  - исправност и
  - ефикасност
- Без њих је софтвер обично неупотребљив
- Али се фокусирањем на њих често занемарују друге веома важне карактеристике

Пожељене карактеристике софтвера (2)

- У основне пожељне карактеристике спадају и:
  - флексибилност и
  - проширивост
- Без њих софтвер има само временски локалну вредност
  - није прилагодљив променама у домену
  - не може да се употреби на другом месту
  - тешко се одржава

Пожељене карактеристике софтвера (3)

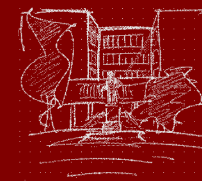
- Да би софтвер могао да буде флексибилан и проширив, пожељно је да има још неке карактеристике:
  - модуларност
  - раздвојеност одговорности компоненти
  - прецизност и јасноћу интерфејса
  - доследност енкапсулације
  - високу кохезију
  - ниску спрегнутост

Универзитет у Београду - Математички факултет

Садржај квантитативног процењивања

- Квантитативно процењивање има исти циљ као и квалитативно
- Разлика је само у тежњи да се мерила квалитета изразе нумерички, како би се лакше упоређивала
- На пример, да бисмо могли да нумерички проценимо прилагодљивост или степен модуларности
- Квантитативним проценама (тј. мерењем) различитих аспеката софтверског пројекта бави се област *софтверске метрике*

Универзитет у Београду - Математички факултет



[P290]

Развој софтвера

Саша Малков

Тема 6

Кохезија и спрегнутост

Кохезија и спрегнутост

- ***Кохезија*** је степен међусобне повезаности елемената једног модула
- ***Спрегнутост*** (енгл. *coupling*) је степен међусобне повезаности елемената двају различитих модула
- Добро дизајниран систем се одликује
  - **високом кохезијом** и
  - **ниском спрегнутошћу** компоненти

Универзитет у Београду - Математички факултет

Кохезија и спрегнутост (2)

- Кохезија и спрегнутост су веома значајне мере
  - описују сложеност софтверског система
  - описују карактеристике повезаности делова система
  - имају изузетно важан утицај на скоро све одлуке које доносимо током пројектовања
- Морамо да их добро упознамо
  - да бисмо у свом пројекту уочили њихове лоше облике
  - да бисмо разумели како да их поправимо

Универзитет у Београду - Математички факултет

Карактеристике кохезије

- Различите облике кохезије ћемо разликовати према ***врсти***
- Неке врсте кохезије су добре, а неке нису
- Тежимо да имамо чврсте облике кохезије

Врсте кохезије

- Разликујемо следеће врсте кохезије (од јачих према слабијим):
  - функционална кохезија
    - (најјача и најпожељнија)
  - секвенцијална кохезија
  - комуникациона кохезија
  - процедурална кохезија
  - временска кохезија
  - логичка кохезија
  - коинцидентна кохезија
    - (најслабија и најнепожељнија)

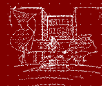
Функционална кохезија

- *Функционална кохезија* представља повезаност делова једног модула на основу међусобне функционалне зависности, а у циљу остваривања функције за коју је компонента одговорна
  - сви делови модула су прикупљени са једним истим основним циљем
  - сваки део је потребан – ниједан део није вишак
- Функционална кохезија је идеал коме се тежи при пројектовању

Секвенцијална кохезија

- Кохезија је *секвенцијална* ако су елементи компоненте пројектовани тако да излаз једне представља улаз друге
  - не постоји пуна функционална зависност
  - потенцијално делови секвенце обављају различите послове
  - потенцијално отворен проблем одговорности у односу на целовит посао

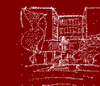
Комуникациона кохезија



- Кохезија је *комуникациона* ако су елементи компоненте прикупљени зато што користе исте податке
  - не постоји функционална зависност
  - обично делови обављају различите послове
  - обично нису добро раздвојене одговорности
    - једна компонента има више одговорности
    - једна одговорност подељена на више компоненти

Универзитет у Београду - Математички факултет

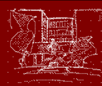
Процедурална кохезија



- Кохезија је *процедурална* ако су елементи компоненте прикупљени зато што се користе при обављању неког целовитог посла
  - нпр. отварање датотеке, проверавање исправности,...
  - вероватно не постоји функционална зависност
  - често делови обављају потпуно разнородне послове
    - који се, додуше, често обављају један за другим
  - обично нису добро раздвојене одговорности
    - једна компонента има више одговорности
    - једна одговорност подељена на више компоненти

Универзитет у Београду - Математички факултет

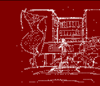
Временска кохезија



- Кохезија је *временска* ако су елементи компоненте прикупљени заједно зато што се користе у "истом" периоду времена
  - нпр. различити елементи иницијализације система,...
  - вероватно не постоји функционална зависност
  - често делови обављају потпуно разнородне послове
  - обично нису добро раздвојене одговорности
    - једна компонента има више одговорности
    - једна одговорност подељена на више компоненти

Универзитет у Београду - Математички факултет

Логичка кохезија



- Кохезија је *логичка* ако су елементи компоненте прикупљени заједно зато што имају логички сличну (или чак исту) улогу у систему
  - нпр. различити начини читања података са улаза...
  - најчешће не постоји функционална зависност
  - делови обично обављају потпуно разнородне послове
    - који могу представљати алтернативу једни другима
  - обично нису добро раздвојене одговорности
    - једна компонента има више одговорности
    - једна одговорност подељена на више компоненти

Универзитет у Београду - Математички факултет

Коинцидентна кохезија

- Кохезија је *коинцидентна* ако су елементи компоненте међусобно неспрегнути или су веома слабо спрегнути
  - нпр. компонента обухвата све јединице кода писаног неким алатом,...
  - не постоји никаква или постоји само сасвим ниска функционална зависност
  - делови обављају потпуно разнородне послове
  - уопште нису раздвојене одговорности
- Коинцидентна кохезија је најнижи ниво кохезије

Универзитет у Београду - Математички факултет

Спрегнутост је неминовна

- Рекли смо да желимо да имамо што нижи ниво спрегнутости
  - али не можемо да је елиминишемо
- Да би компоненте система сарађивале оне ***морају*** да буду спрегнуте
  - ако компоненте комуницирају или сарађују на било који начин, онда међу њима постоји спрега
  - ако су компоненте потпуно независне, онда оне не могу да сарађују
- Спрегнутост није проблем сама по себи
  - проблем могу представљати неке карактеристике спрегнутости

Универзитет у Београду - Математички факултет

Карактеристике спрегнутости

- Врста спреге
- Ниво спреге
- Ширина спреге
- Смер спрегнутости
- Начин остваривања спреге
- Интензитет спрегнутости

Универзитет у Београду - Математички факултет

Врсте спрегнутости

- Основне врсте спрегнутости су:
  - спрега логике
  - спрега типова
  - спрега спецификације

Универзитет у Београду - Математички факултет

Спрега логике

- Ако компоненте деле информације или претпоставке једна о другој, онда је у питању спрега логике
- Примери:
  - компоненте обављају различите делове једног истог посла
  - компонента А претпоставља да имплементација метода Б::М почива на неком конкретном алгоритму
    - нпр. А и Б имплементирају комплементарне алгоритме (кодирање/декодирање, писање/читање,...)
  - компоненте деле претпоставке о неким конкретним подацима
    - нпр. А записује датотеку под неким именом а Б је чита
- Спрега логике је високо проблематична и непожељна

Спрега типова

- Спрега типова означава да једна компонента користи неки тип дефинисан у оквиру друге компоненте
- Може бити:
  - одређена – ако компонента А прави инстанце типа Б
  - неодређена – ако компонента А не прави инстанце типа Б
    - тј. може у пракси добијати на употребу инстанце подтипова
- Примери:
  - компонента А користи класу имплементирану у компоненти Б
- Добро имплементирана спрега типова не представља проблем
  - посебно ако је неодређена

Спрега спецификације

- Спрега спецификације је још апстрактнија него спрега типова
- Претпоставља се да није познат тип који се користи већ само неке претпоставке о његовом интерфејсу
- Примери:
  - употреба параметарског или имплицитног полиморфизма
  - употреба референци на методе објеката
  - апстрактне базне класе хијерархија
  - имплементације интерфејса
- Добро имплементирана спрега спецификације не представља проблем

Нивои спрегнутости

- Спрегнутости се деле по нивоу апстрактности делова који се деле (од најјаче према најслабијој):
  - по садржају
  - преко заједничких делова
  - спољашња спрегнутост
  - преко контроле
  - преко маркера
  - преко података
  - преко порука

Универзитет у Београду - Математички факултет

Спрегнутост по садржају

- Спрегнутост *по садржају* представља отворено и непосредно коришћење и/или мењање садржаја једне компоненте од стране друге
- Поједностављен пример:

```
... A::metod(...)
{
    ... objB->podatak ...
}
```

Универзитет у Београду - Математички факултет

Спрегнутост по садржају (2)

- Проблем:
  - нарушена је енкапсулација
  - доведене су у питање одговорности компоненти
  - веома је тешко одржавати компоненту од чијих интерних аспеката имплементације зависи друга компонента
- Спрегнутост по садржају је најјача и најнепожељнија врста спрегнутости

Универзитет у Београду - Математички факултет

Спрегнутост преко заједничких делова

- Спрегнутост *преко заједничких делова* је на делу ако две или више компоненти непосредно приступају неким подацима
- Поједностављен пример:

```
struct ZajednickiDelovi {...};
... A::metod1(...){
    ... zajednickiDelovi->podatak ...
}
... B::metod2(...){
    ... zajednickiDelovi->podatak ...
}
```

Универзитет у Београду - Математички факултет

Спрегнутост преко заједничких делова (2)

- Проблем:
  - разликује се од спрегнутости по садржају само по томе што су дељени подаци одвојени од понашања и стављени на употребу другим компонентама
  - енкапсулација само начелно није нарушена, а заправо није ни успостављена
  - ни одговорности нису успостављене
  - веома је тешко одржавати овако спрегнуте компоненте
- Спрегнутост преко заједничких делова је скоро једнако непожељна као спрегнутост по садржају

Универзитет у Београду - Математички факултет

Спољашња спрегнутост

- *Спољашња спрегнутост* је на делу ако више компоненти употребљава исти од споља наметнут концепт (интерфејс, формат података, комуникациони протокол,...)
- Поједностављен пример:

```
... A::metod1 {
  ...external::oper1(...);
  ...external::oper2(...);
};
... B::metod2(...){
  ...external::oper3(...);
  ...external::oper4(...);
}
```


Спољашња спрегнутост (2)

- Проблем:
  - често понављање кода
    - ако се дати концепт измени, морају се мењати све компоненте које га употребљавају
  - подељене одговорности
    - пожељно је размотрити учауривање елемената спољашњег концепта у нову компоненту (или компоненте), коју би затим користиле овако спрегнуте компоненте
- *Спољашња спрегнутост* је често сасвим прихватљива
  - ако су спољашњи концепти релативно стабилни и поуздани
  - на пример, *слика* имплементира основне методе а *трансформације* имплементирају сложене операције над сликама

Спрегнутост преко контроле

- Спрегнутост је *преко контроле* ако једна компонента ултимативно управља радом друге компоненте
  - управљана компонента није у стању да функционише без спољашње контроле или
  - управљана компонента очекује сва или нека упутства за рад по којима затим самостално поступа
- У односу на спољашњу спрегнутост, контролер је једина или примарна тачка контроле

Спрегнутост преко контроле (2)

- Поједностављен пример:

```
class Kontrolisana {
  ...oper1(...);
  ...oper2(...);
};
... Kontroler::metod(...){
  ...kontrolisana->oper1(...)...
  ...kontrolisana->oper2(...)...
}
```

- На пример, контролисана класа може да буде *слика*, а све сложеније трансформације се имплементирају у другој класи која представља *контролер*

Спрегнутост преко контроле (3)

- Проблем:
  - потенцијално нејасна подела одговорности
    - ако је контролор одговоран за виши ниво операција, онда то мора бити његова једина одговорност
    - често такво решење ни по чему није боље од интегрисања свих операција у један исти модул, тј. у оба случаја потенцијално имамо модуле са сувише одговорности
- Спрегнутост *преко контроле* је релативно јака, али може бити веома корисна у неким случајевима
  - нпр. контролисана компонента имплементира елементарне поступке, а различите стратегије обезбеђују сложене начине управљања радом компоненте
  - да се вратимо на пример са сликом и трансформацијама:
    - на нивоу класа, може да буде добро да се за сваку трансформацију прави нова класа
    - на нивоу компоненти, све трансформације се обично окупљају у једну компоненту

Универзитет у Београду - Математички факултет

Спрегнутост преко маркера

- Спрегнутост је *преко маркера* ако више компоненти међусобно размењује неку сложену структуру података (маркер) коју употребљавају на различите начине
  - ако непосредно користе исте податке маркера, онда је у питању спрегнутост преко заједничких делова
- Поједностављен пример:

```
...
parser->fillRequest( request );
environment->prepareRequest( request );
processor->performTransaction( request );
response = reporter->prepareReport( request );
...
```

Универзитет у Београду - Математички факултет

Спрегнутост преко маркера (2)

- Проблем:
  - потенцијално отворено питање одговорности маркера
    - шта је његова одговорност? зашто не све или неке операције?...
  - маркер практично није довољно енкапсулиран
  - постоји опасност од сукоба надлежности
- Слично као у случају спрегнутости преко контроле, у неким случајевима је овакво решење сасвим прихватљиво
  - ако маркер уопште нема сложено понашање
  - ако маркер није спрегнут на други начин осим овим путем
  - ако поступак обраде у коме маркер учествује није стабилан или реално зависи од више различитих компоненти

Универзитет у Београду - Математички факултет

Спрегнутост преко података

- Спрегнутост је *преко података* ако једна компонента користи интерфејс друге компоненте путем кога му преноси појединачне податке
- Поједностављен пример:

```
... simulacija::promenaSmera(...)
{
  ...
  automobil->skreniLevo( 5 );
  ...
}
```

Универзитет у Београду - Математички факултет

Спрегнутост преко података (2)

- Проблем:
  - зависност на нивоу интерфејса
- Овај вид спрегнутости је међу најнижим и у начелу је сасвим прихватљив

Спрегнутост преко порука

- Спрегнутост је *преко порука* ако једна компонента користи интерфејс друге компоненте путем кога му не преноси никакве податке
- Поједностављен пример:

```
... A::metod1(...) {
  ...
  transakcija->izvrsi();
  ...
}
```


Спрегнутост преко порука (2)

- Проблем:
  - зависност на нивоу интерфејса
- Ово је најнижи вид спрегнутости и у начелу је сасвим прихватљив

Ширина спреге

- Ширина спреге двеју компоненти одговара броју елемената компоненте А (објеката, података, метода, порука,...) које употребљава компонента Б
- Пожељно је да спрега буде што ужа
  - уска спрега значи да су јасни и подела одговорности међу компонентама и разлог њихове спрегнутости
  - ако је спрега широка, то може да значи да
    - одговорности између ове две компоненте нису недовољно диференциране
      - можда је могуће сузити спрегу имплементирањем неких сложенијих целина кода у оквиру компоненте А
    - одговорности компоненте А су прешироке
      - можда је потребно поделити компоненту А на више мањих

Универзитет у Београду - Математички факултет

Смер спреге

- Спрега може бити
  - једносмерна
    - компонента А употребљава елементе компоненте Б, али не и обратно
  - двосмерна
    - компонента А употребљава елементе компоненте Б, али и компонента Б употребљава елементе компоненте А
  - циркуларна
    - не постоји непосредно двосмерна спрега двеју компоненти, али постоји транзитивна циркуларна спрегнутост више компоненти

Смер спреге (2)

- Пожељно је да спрега буде једносмерна
- Двосмерна и циклична спрега значајно увећавају сложеност система
  - усложњава се подела одговорности међу компонентама
  - доводи се у питање енкапсулација
- Ипак, некада двосмерна или циркуларна спрегнутост може да значајно поједностави дизајн
  - на пример, инверзија одговорности код обрасца Посетилац

Начин остваривања спреге

- Спрегнутост може бити статичка и динамичка

Статичка спрегнутост

- Остварује се пре извршавања програма
- Експлицитно је исказана у програмском коду
- Пример:
  - наслеђивање класа
  - податак класе А је објекат класе Б
  - аргумент метода класе А је објекат класе Б
  - у методима класе А се прави или употребљава објекат класе Б
  - ...

Динамичка спрегнутост

- Остварује се у току извршавања програма
- Није експлицитно исказана кодом програма
- Може да зависи од конфигурације, улазних параметара,...
- Пример:
  - ако класа А садржи референцу на класу Б (статички), конкретан објекат (динамички конфигурисан) може припадати било којој класи која наслеђује Б
  - ...

Универзитет у Београду - Математички факултет

Однос статичке и динамичке спреге

- Статичка спрега је јача и непожељнија од динамичке
  - декларација (интерфејса...) и имплементација неке статички спрегнуте компоненте имају непосредног утицаја на имплементацију осталих спрегнутих компоненти
  - промена динамички спрегнуте компоненте је локализована – ако одговара наслеђеном интерфејсу, онда је све у реду, а ако не одговара, потребно је само поправити ту компоненту

Универзитет у Београду - Математички факултет

Интензитет спрегнутости

- Интензитет спрегнутости је сложена карактеристика
  - логичка спрега има већи интензитет од спреге типова и спецификација
  - виши ниво спрегнутости има већи интензитет
  - шира спрега има већи интензитет
  - циркуларна спрега има већи интензитет од двосмерне, а она од једносмерне
  - статичка спрега има већи интензитет од динамичке

Универзитет у Београду - Математички факултет

Мера спрегнутости

- Мера спрегнутости неке компоненте са системом се може изразити на више начина
  - број компоненти које се реферишу из те компоненте
    - број елемената других компоненти које се реферишу из те компоненте
  - број компоненти из којих се реферише та компонента
    - број елемената те компоненте које се реферишу из других компоненти
  - карактеристике посматраних спрега
  - ...

Универзитет у Београду - Математички факултет

Рачунање интензитета спреге

- Интензитет спрегнутости се може изразити нумерички на више начина, нећемо то овде детаљно разматрати
- На пример:
  - Интензитет спрегнутости двеју компоненти:
    - coefSprege(A,B) = coefVrste(A,B) + coefNivoa(A,B) + ...
  - Спрегнутост једне компоненте са системом:
    - coefSprege(A) = sum_B(coefSprege(A,B))
  - Укупна спрегнутост система:
    - coefSpregeSistema = sum_A(coefSprege(A))

"Аксиоме спрегнутости"

- 1. Што је компонента сложенија,
  то је важније да буде што мање спрегнута
- 2. Спрегу је пожељно уводити
  на што једноставнијим компонентама

Уобичајени облици спрега

- Неки облици спрегнутости су релативно чести
- Могу се препознати у узорцима за пројектовање
- Примери:
  - архитектура клијент-сервер
  - хијерархија припадности
  - циркуларна спрегнутост
  - спрега путем интерфејса
  - спрега путем параметара метода

Архитектура клијент-сервер

- Један од најчешћих и типичних облика спреге:
  - компонента *сервер* пружа услуге
  - компонента *клијент* користи услуге
- Ова спрега је
  - једносмерна
  - пожељно уска (сервер пружа само један тип услуга)
  - ниво спреге је обично релативно низак – преко порука или преко параметара
  - спрега типова или спецификација
  - низак интензитет спреге препоручује ову врсту спреге као идеалну

Хијерархија припадности

- Релативно чест облик спреге:
  - компонента *родитељ* садржи више *деце*, која заправо обављају функцију или делове функције
  - компоненте *деца* обављају свака свој део одговорности
  - *родитељ* је одговоран за координацију
- Пример
  - прозор – декоратери
- Ова спрега је
  - обично једносмерна
  - релативно уска
  - ниво спреге је обично релативно низак – преко порука или преко параметара
  - спрега типова или спецификација
  - ниво се често може даље спустити, нпр. применом обрасца градитељ

Циркуларна спрегнутост

- Релативно чест облик спреге:
  - компонента *родитељ* садржи више *деце*
  - свако *дете* зна ко му је родитељ и приступа му ради одређених послова
- Пример
  - прозор – контроле
- Ова спрега је
  - двосмерна
  - потенцијално широка
  - ниво спреге је обично релативно низак – преко порука или преко параметара
  - спрега типова или спецификација
  - ниво се често може даље спустити, нпр. применом заједничких надтипова и/или градивних образаца

Спрега путем интерфејса

- Типичан облик спреге у ОО развоју:
  - компонента *Б* декларише (и имплементира) интерфејс
  - компонента *А* употребљава компоненту *Б* посредством интерфејса
- Ова спрега је
  - потенцијално једносмерна
  - ширина зависи од квалитета дизајна интерфејса
  - ниво спреге је обично релативно низак – преко порука или преко параметара
  - спрега типова или спецификација

Спрега путем параметара метода

- Чест облик спреге у ОО развоју:
  - компонента *А* декларише метод који као параметар прима компоненту *Б*
  - компонента *ББ*, као специјализација компоненте *Б*, се предаје на употребу компоненти *А* као параметар декларисаног метода
- Ова спрега је
  - потенцијално једносмерна
  - ширина зависи од квалитета дизајна интерфејса
  - ниво спреге је обично преко маркера
  - обично спрега типова или спецификација

Класе и кохезија

- У контексту ОО програмирања:
  - Кохезија класе представља степен међуповезаности елемената једне класе
    - ако је низак ниво кохезије у класи, то указује да класа није добро обликована
      - ако постоји кохезија између неких елемената класе, али не свих, или
      - ако се препознаје више различитих скупова елемената класе унутар којих постоји снажна кохезија
    - могуће је да постоји проблем препознавања одговорности класе
    - класу је можда потребно поделити на више класа

Универзитет у Београду - Математички факултет

Класе и спрегнутост

- У контексту ОО програмирања:
  - Спрегнутост представља степен међуповезаности различитих класа
    - ако је висок ниво спрегнутости различитих класа, то указује да одговорности нису добро подељене међу класама
    - можда је од више класа потребно направити једну
    - можда је потребно другачије поделити одговорности међу класама

Универзитет у Београду - Математички факултет

Спрегнутост и обрасци за пројектовање

- Обрасци за пројектовање нуде решења за неке честе проблеме тако да
  - јасно дефинишу одговорности делова
  - укажу на неопходан ниво спрегнутости

Универзитет у Београду - Математички факултет

Спрегнутост и рефакторисање

- Више техника рефакторисања имају за циљ управо смањивање међусобне спрегнутости компоненти
  - смањивање нивоа
  - статичка у динамичку
  - двосмерна или циркуларна у једносмерну
  - ...

Универзитет у Београду - Математички факултет

Спрегнутост и архитектура

- Један од најважнијих задатака при пројектовању софтвера је остваривање потребне функционалности уз што вишу кохезију и што нижу спрегнутост компоненти
  - своди се на добро препознавање компоненти и њихових одговорности
- Већина савремених архитектура и методологија развоја софтвера узима у обзир управљање сложеностима проблема и решења
- Примери:
  - Вишеслојне архитектуре софтвера
  - Архитектуре оријентисане према услугама (енгл. *service oriented achitecture*)
  - Софтвер управљан догађајима
  - ...

Универзитет у Београду - Математички факултет

Литература за тему

- *Shari Lawrence Pfleeger, Joanne M. Atlee, **Software Engineering – Theory and Practice**, 3rd ed., Pearson Education, 2006.*
  - *превод на српски, CET, 2006.*
- *Jack Reeves, **What is Software Design?**, C++ Journal, 1992.*
  - *http://user.it.uu.se/~carle/softcraft/notes/Reeve_SourceCodeIsTheDesign.pdf*
- *Peter Eeles, Peter Cripps, **The Process of Software Architecting**, Addison-Wesley Professional, 2009.*
- *Peter Cripps, **Software Architecture Zen***
  - *http://softwarearchitecturezen.blogspot.com/*
- *Robert C. Martin, **Agile Software Development – Principles, Patterns and Practices**, Prentice Hall, 2003.*

Универзитет у Београду - Математички факултет

Хвала на пажњи!

МАТФ
Универзитет у Београду
Математички факултет