Менаџмент на информациски системи [2018]

Професор: д-р Слободан Калајџиски

Содржина

[Предавања – I Колоквиум 3](#_Toc530501669)

[1. Вовед во информациски системи 3](#_Toc530501670)

[2. Методологии и развој на информациски системи 6](#_Toc530501671)

[3. BSP – Business System Plan 14](#_Toc530501672)

[4. DFD – Data Flow Diagram 21](#_Toc530501673)

[5. UML – Класен дијаграм 28](#_Toc530501674)

# Предавања – I Колоквиум

## 1. Вовед во информациски системи

**Карактеристики на концептот на информациски систем**

* Има граници т.е. има работи кои се надвор од системот (системот опстојува во некоја околина)
* Може да има други системи во рамките на еден систем
* Составните делови или нешта може да се луѓе (корисници, технички лица, поддршка, ИТ професионалци...) и технологија (хардвер и софтвер)
* Главен предизвик на информациските системи е можното интегрирање на технолошкото решение во организацискиот или социјалниот контекст
* Организирана комбинација на:
  + Хардвер
  + Софтвер
  + Комуникациски мрежи
  + Податоци
  + Луѓе

**Која улога ја играат информациските системи во бизнисот?**

* Во споредба со пред-компјутерските информациски системи, компјутерските базирани ИС обезбедуваат:
  + Зголемена брзина
  + Ефикасност
  + Квалитет
  + Мерливост

**Главни типови на информациски системи**

* Operations Support Systems [OPS – Поддршка за бизнис операции]
  + Transaction Processing Systems [TPS – Процесирање на бизнис трансакции]
  + Process Control Systems [PCS – Контролирање на индустриски процеси]
  + Enterprise Control systems [ECP – Колаборација на работни групи и тимови]
* Management Support systems [MSS – Поддршка при менаџирање на маркетинг одлуки]
  + Management Information Systems [MIS – Преназначен отчет за менаџери]
  + Decision Support Systems [DSS – Интерактивна поддршка за одлуки]
  + Executive Information Systems [EIS – Информации направени по мерка за извршители]

**Transaction Processing Systems**

* Поддршка на секојдневните операции
* Полнење и одржување на тековните бази на податоци
* Типични примери:
  + Продажба и маркетинг
  + Производство и продажба
  + Финансии и сметководство
  + Управување со човечки ресурси

**Process Control Systems**

* Поддршка на производствени операции
* Мониторирање и контрола на индустрискиот – производствен процес
* Типични примери:
  + Рафинерија на нафта
  + Електрична централа
  + Производство на автомобили

**Enterprise Collaboration Systems**

* Поддршка на тековните операции
* Тимска работа, комуникација и колаборација
* Создавање на т.н. „виртуелна“ канцеларија
* Типични примери:
  + E-mail
  + Chat
  + Видео канцеларија
  + Календар

**Management Information Systems**

* Поддршка на менаџмент
* Анализи и извештаи
* Charts, Graphs, Summary Tools
* Типични примери:
  + Управување со универзитет
  + Spreadsheet – Првата и најпроста алатка
* Менаџерско ниво
  + **ВЛЕЗ:** огромна количина на податоци
  + **ОБРАБОТКА:** прости модели за пресметка
  + **ИЗЛЕЗ:** сумарни извештаи
  + **КОРИСНИЦИ:** менаџери на оддел

**Decision Support Systems**

* Поддршка на менаџментот
* What-if Analysis, моделирање на одлуките, креирање на сценарија, висока интеракција
* Менаџерско ниво
  + **ВЛЕЗ:** ниско ниво на податоци
  + **ОБРАБОТКА:** интерактивна обработка
  + **ИЗЛЕЗ:** анализа на одлуки
  + **КОРИСНИЦИ:** професионалци, вработени
* Флексибилност, прилагодливост
* Кориснички контролиран влез и излез
* Поддршка на процесот за донесување одлуки
* Софистицирани моделирачки алатки

**Executive Support Systems**

* Поддршка на највисокото ниво на стратегиски менаџмент
* Обезбедува критични информации од другите системи (MIS и DSS)
* **Концепт на портал**: едно место со линкови од сите информации
* ESS вклучуваат **надворешни информации**
* Помагаат во стратегиски донесување одлуки, кое не мора да биде тактичко.
  + Tactical – doing things the right way
  + Strategic – doing the right things
* Стратегиско ниво
  + **ВЛЕЗ:** агрегатни податоци
  + **ОБРАБОТКА:** интерактивна обработка
  + **ИЗЛЕЗ:** проекции
  + **КОРИСНИЦИ:** Senior менаџери
* Менаџмент од највисоко ниво
* Дизајнирано кон индивидуата
* Го поврзува CEO со сите нивоа од организација
* Премногу скапо за одржување

## 2. Методологии и развој на информациски системи

**Методологии за развој на ИС**

* Методологии за развој на софтвер е еден вид структура која треба да се направи во рамките на процесот на изработка на еден информациски систем
* Вклучува процедури, техники, алатки и документација со чија помош тие што го имплементираат поефикасно ќе ја завршат својата должност
* Целта на методологијата е да се формализира она што е веќе направено, со што истото ќе може да се повторува и искористува повеќе пати

**Потребата од методологијата за развој на информациски системи**

* Во текот на последните 4 декади, **голем број** развојни методологии се појавуваат на пазарот
* Нивните пораки понекогаш може да се **збунувачки** и **противречни**
* Според студијата спроведена од ***Стендиш*** групатаспроведена во 2009, 32% од сите проекти се испорачани на време и во рамките на буџетот, 44% биле со задоцнување и со надминат буџет и 24% биле откажани пред да дозавршат или испорачаат
* Студијата спроведена од ***Forester*** истражувачката група заклучува дека скоро 1/3 од сите IT проекти во просек доцнат до три месеци. Во повеќето случаи на неуспех, тоа било заради не применување на методологијата или користење на погрешна методологија.
* Според ***CIO.com*** околу 3/4 од сите IT проекти од интернет ерата реализирани во последните 7 години се соочиле со некој од следните проблеми: целосен неуспех, надминување на трошоците и времето или испорака на помалку модули отколку што било планирано.

**Методологии за развој на ИС**

* Во најраните фази на развој на ИС, кодот бил пишуван, а после дебагиран. Не постојат формално чекорите на анализа и дизајн
* Најпрвин се разработува пристапот за развој на сложени хардверски системи, што по аналогија сличниот пристап се применува и во развојот на информациските системи, односно софтверот.
* Според овие методи на информациските системи подлежи на парадигмата requirements/design/build со стандардни, добро – дефинирани процеси. Овие методологии може да се категоризираат како **тешки методологии.**
* Покрај овие се појавуваат и нови методологии кои многу лесно се адаптираат на промени. Тие не се фокусираат на долгиот процес на развој на еден ИС, туку на кратки итерации, лесни процеси и блиска вклученост на клиентот. Овие методологии се познати како **лесни** или **агилни** **методологии**.

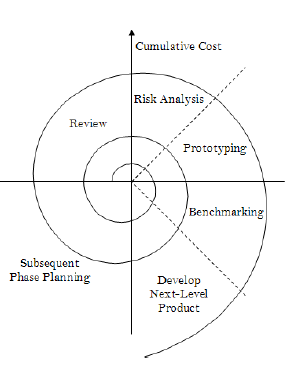
**Преглед на клучните тешки и лесни методологии**

* Методологиите за развој на софтвер може да се поделат на две групи:
  + **Тешки методологии** 
    - Waterfall methodology
    - Rational Unified Process (RUP)
    - Spiral methodology
  + **Агилни методологии** 
    - Extreme programming (XP)
    - Scrum

**Waterfall методологијата**

* Во текот на 1960-тите години, “code & fix” принципот беше најчесто користен метод од страна на софтверските корисници.
* Со оглед на сложената природа на принципот, Winston Royce во 1970 ја предлага **waterfall методологијата** со цел да може да се справи со зголемената комплексност на вселенскиот софтвер.
* Овој пристап ја потенцира структурната прогресија помеѓу дефинираните фази. Секоја фаза се состои од предефинирани активности и цели кои треба да се исполнат пред да се почне со следната фаза.
* Предности:
  + Дисциплиниран процес
  + Бара да се има комплетно множество на кориснички барања пред да се почне со нивна анализа
  + Ги форсира анализата и дизајнот да бидат на прво место
* Недостатоци:
  + Нема ран feedback (prototyping)
  + Тромава техника и споро реагира на промени
  + Висок број на пропуштени нејасни кориснички барања
  + Оптимизирана е за хардвер, со што ги занемарува основните карактеристики на софтверот
* На корисниците генерално им е **многу тешко да даваат прецизни објаснувања за нивните потреби** во форма лесно разбирлива од страна на развивачот на системот.
* **Големината на системите** постојано се зголемува, што бара поголеми тимови од развивачи и покомплексна комуникација и координација.
* Во големите системи, постои **комбинаторна експлозија** со што бројот на патеки низ програмскиот код е огромен. Ова предизвикува сеопфатното тестирање на кодот да биде скоро невозможно.
* **Недостатокот на математички алатки** и неможноста целосно да се моделира однесувањето на големи системи, не тера да се задоволуваме со под-оптимални нивоа на доверба во врска со нивната исправност.

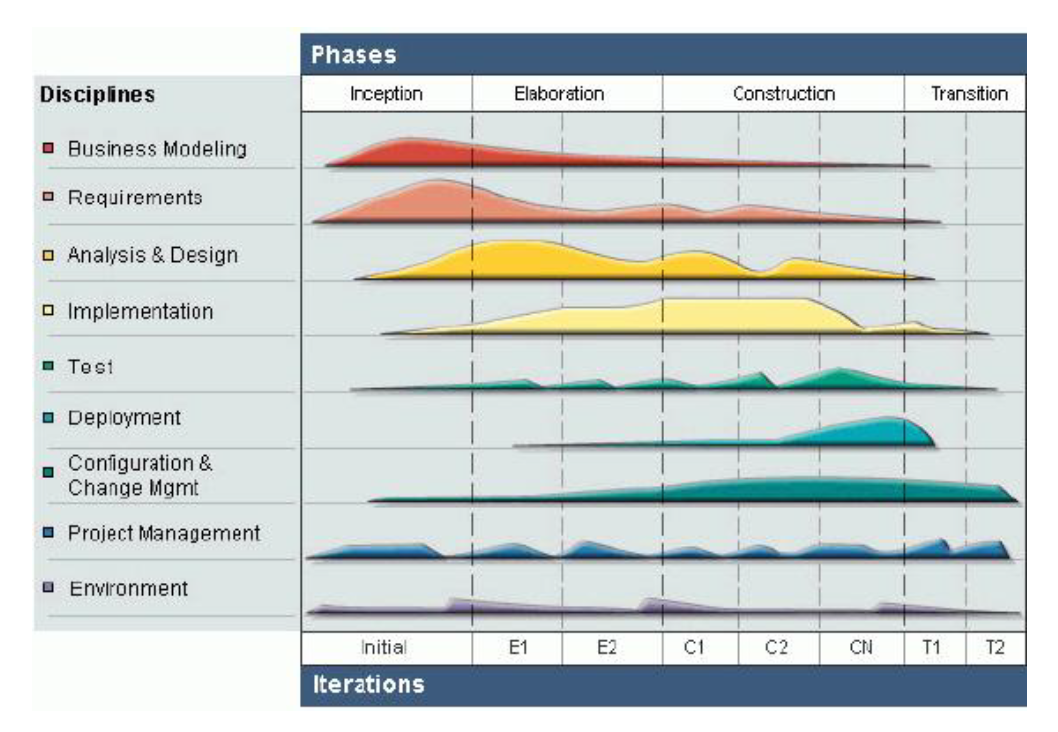
**Еволуција на методологиите за развој на информациските системи**

* Методологиите за развој на информациските системи се развивале според следните работи:
  + Оптимизација на waterfall моделот
  + Овозможување на софтверското прототипирање за подобро разбирање на корисничките барања
  + Намалување на времето за испорака и притисоците за подобар квалитет
  + Зголемување на можноста од ре-искористливост

**Спирална методологија**

* Спиралниот модел е методологија за развој на ИС првично развиена во 1998 година од страна на Larry Boehm
  + Како што е првично предвидено, итерациите обично траат од 6 до 24 месеци
  + Ги комбинира протипирачкиот и waterfall моделот
  + Се применува кај големи, скапи и комплицирани проекти
* Целта на оваа методологија е да се стави акцент на проценката на ризикот и резолуцијата

**RUP (Rational Unified Process)**

* Rational Unified Process обезбедува препораки, шаблони и алатки кои ги користат учесниците во тимот за да може целосно да се искористат следните најдобри практики:
  + Развој на софтверот итеративно и инкрементално
  + Управување со корисничките барања преку use case
  + Користење на компонентно-базирани архитектури
  + Визуелно моделирање на информацискиот систем преку UML
  + Верификација на квалитетот на софтверот
  + Контрола на промените во софтверот
* Хоризонталната оска го претставува времето и ги прикажува динамичките аспекти на процесот и се претставува во форма на циклуси, фази, итерации и пресвртници (milestones).
* Вертикалната оски ги дава статичните аспекти од процесот: како процесот се прикажува во форма на активности, артефакти, работници и податочни текови (workflows).

**Заеднички карактеристики на тешките методологии**

* **Предвидлив пристап –** Тешките методологии имаат тенденција првин да испланираат во најситни детали голем дел од софтверскиот процес. **Овој пристап го следи инженерскиот пристап каде што развојот е предвидлив и повторувачки.**
* **Сеопфатна документација –** Традиционалниот развој на ИС го третира документот со корисничките барања како клучен де од документацијата. Главниот процес во тешките методологии е големиот однапред дизајнерски процес (big design upfront – BDUF)
* **Процесно ориентирани –** Целта на тешките методологии е да се дефинира процесот кој ќе работи добро без разлика кој ќе го користи истиот. Процесот се состои од одредени задачи кои мора да се извршат од страна на менаџерите, дизајнерите, програмерите итн. За секоја од овие задачи постои добро дефинирана процедура.
* **Tool ориентирани –** Проектниот менаџмент и алатките за развој на софтвер мора да се користат за завршување и испорака на секоја од задачите.

**Scrum**

* Scrum е методологија за развој на ИС наменет за итеративно – инкрементирачките софтверски развојни проекти.
* На крајот од секоја итерација тој продуцира потенцијален сет од функционалности.
* Изразот “scrum” оригинално произлегува од стратегијата во рагби играта. Во рагбито има значење на „добивање на топка надвор од игра, назад во игра – getting an out-of-play ball back into the game“ – овозможено со тимска игра.
* Scrum се потпира на само-организација, тимот одлучува што ќе прави, додека менаџментот ги средува конфликтите и ги отстранува пречките во изработка на ИС.
* Главни карактеристики:
  + Клиентите стануваат дел од развојниот тим
  + Чести испораки на модули со исполнета функционалност
  + Чест ризик и ублажувачки планови развиени од страна на самиот им на развој
  + Дискусии на дневно ниво на кој секој член се прашува:
    - Што сработил од вчера до денес?
    - Што планираш да сработиш до утре?
    - Дали има некој проблем што спречува да се оствари зацртаниот план?
  + Чести состаноци на засегнатите страни на кои се следи напредокот на процесот.

**Extreme Programing (XP)**

* Екстремното програмирање е итеративна и инкрементална методологија за развој на ИС.
* XP vs Scrum
  + XP се фокусира на програмирањето, додека пак Scrum се фокусира на проектниот менаџмент
  + И двете се лесни техники и најчесто се користат заедно
  + Scrum итерациите се подолги отколку кај XP (2 недели vs 4 недели за Scrum)
  + Scrum тимовите не дозволуваат промени во нивните итерации, додека пак XP тимовите се многу прилагодливи на промените на нивните итерации
* Клучни XP Практики
  + **Планирање:** брзо се прави груб план кој се детализира кога работите стануваат појасни
  + **Програмирање во парови:** групи од развивачи што го пишуваат целиот код
  + **Тест – управуван развој:** развивачите пишуваат и тестови за секоја функција склона на грешка уште пред да ја испрограмираат истата
  + **Refactoring:** техника за подобрување на кодот без промена на функционалноста. Секој XP тим безмилосно прави refactoring
  + **Континуирана интеграција:** XP тимовите ги интегрираат своите кодови повеќе пати на ден, откога ќе ги добијат сите тестови на модулот за испробување
  + **On-site клиент:** XP клиент има потреба од достапна личност од клиентска страна која ќе ги појаснува корисничките барања и која прави критични бизнис одлуки

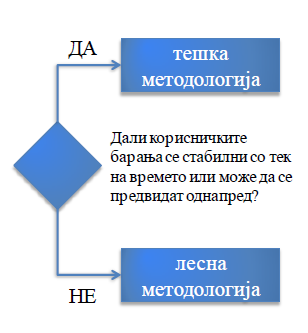
**Користење агилни методологии е возможно во следните случаеви:**

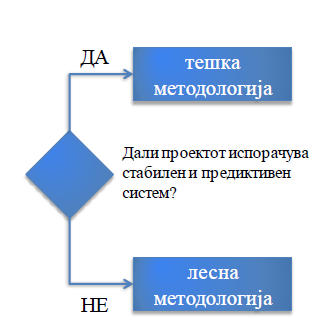
* Околини во кои се случуваат нагли и постојани промени
  + Нејасни/нови кориснички барања
* Проекти со висок приоритет/приход – каде времето на пазарот е критично
  + Агилните методологии се смислени за on-time испорака, и ако е потребно да се продуцираат рани инкрементации или функционалности
* Санација/спасување на проекти
  + Со фокусирање на итна испорака на функционалноста
  + Постојано испорачување на софтвер кој работи без грешки (bug-free) може релативно брзо да изгради доверба помеѓу бизнис страната на тимот

**Избор на правата методологија**

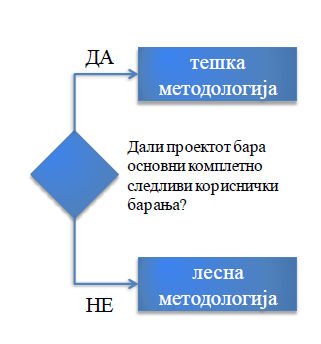
* 7 клучни карактеристики на проектите
  + Големина на проектниот тим
  + Рата на очекуваните промени
  + Главна цел на проектот
  + Управување со корисничките барања
  + Проектната комуникација
  + Однос со клиентот
  + Организациска култура на клиентот

**Рата на очекуваните промени**

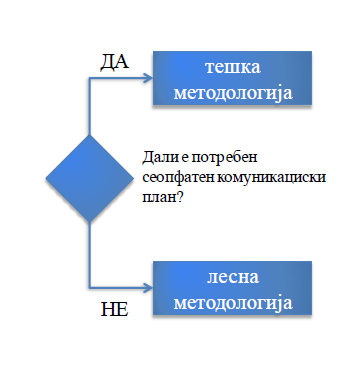
* Лесни методологии:
  + Не се грижат како проектот се интегрира во целата инфраструктура
  + Поради нискиот overload, може да одговори на промените побрзо и може да ги поддржи измените на производот па дури и прилично доцна во развојниот циклус
* Тешки методологии:
  + Цената на промената, особено во подоцните фази на развој е многу повисока
  + Овие се применуваат само во случаеви кога корисничките барања се глобално предвидливи однапред и остануваат стабилни

 **Главна цел на проектот**

* Прав избор за проектите кои бараат критична сигурност и безбедност е тешката методологија, бидејќи бара документиран сет на планови и спецификации
* Лесните методологии пак не подлежат на плански – одредени цели туку тие работат на брзо градење на работите и преку искуство одредуваат на кои карактеристики да дадат најголем придонес во следната фаза

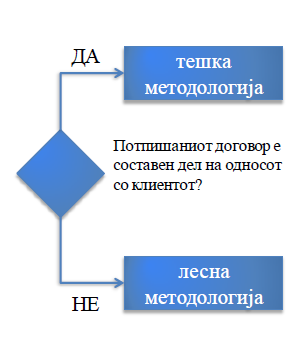


**Управување со корисничките барања**

* Тешките методологии прво ги идентификуваат барањата на корисникот па потоа ги дефинираат истите и на крај ги прегледуваат со соодветниот тим
* Тешките исто така се фокусираат и на справување со количество не-функционални барања како што е сигурноста, перформансите или скалабилноста.
* Лесните пак ги опишуваат барањата во форма на променливи, неформален приказ. Постои блиска интеракција на клиент и тој што го развива софтверот за да се одредат кои барања ќе бидат приоритет.
* Содржината на секоја следна итерација се прави со преговори.

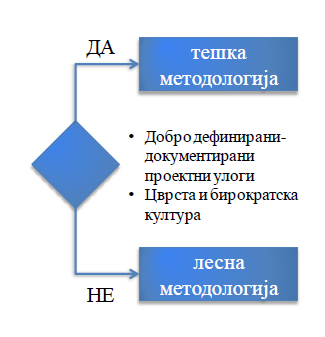
**Проектна комуникација**

* Лесните методологии препорачуваат face-to-face комуникација и не се грижат за документирање (водење записници) на комуникацијата помеѓу тимот
* Иако е можен дистрибуиран развој со лесните методологии, сепак треба да се смисли, испланира и спроведе дополнителен систем на колаборација (за можна промена на системот).



**Однос со клиент**

* Лесните методологии се потпираат на изработка на софтвер во соработка со клиентот.
* Додека пак тешките генерално зависат од некоја форма на договор потпишан меѓу клиентите и развивачите на ИС

**Организациска култура**

* Кај лесните методологии:
  + Не постојат улоги/задачи за секој од членовите
  + Се очекува секој учесник да извршува било каква задача која придонесува на проектот
* Тешките методологии најдобро се применуваат доколку организациската култура налага јасни процедури за секоја улога на секој член (притоа истата е добро документирана и дефинирана).

**Заклучок**

* Повеќето модерни методологии за развој на ИС се базирани на итеративен пристап
* Waterfall методологијата сеуште постои и е сеуште валиден метод на употреба. Сепак треба да се користи каде е соодветен не секаде.
* Резултатите не треба да се сфатат како замена за добар проектен менаџмент туку како насоки за работа.
* Во случаи кога стеблата на одлука се контрадикторни едно со други, тогаш седумте карактеристики мора да се направат приоритетни според организациската култура и тековната проектна ситуација.

## 3. BSP – Business System Plan

**Проблеми во имплементација на ИС**

* ИС се развиле според bottom-up метода (формирање на уникатна перцепција за компанијата, нејзините цели и вработени, го мери оперативниот ризик, го алоцира капиталот и дава на вработените право на глас)
* Нема размена на податоци во една фирма
* Се развиваат посебни апликации за различна намена

**Што е потребно за успешно реализирање на ИС**

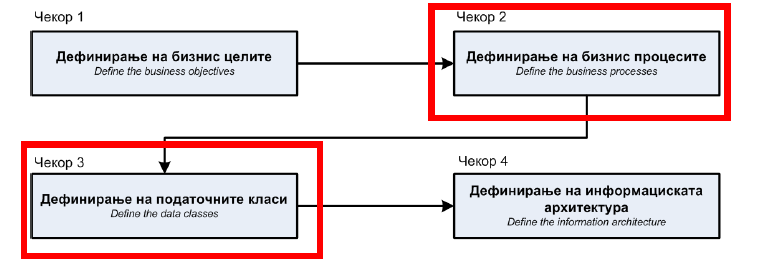
* Поддршка на целата организација, особено раководство
* Правилно идентификување на целите и стратегиите на претпријатието кои се во склоп на неговите бизнис цели од аспект на врвното раководство.
* Утврдување на постоечкиот ИС (доколку постои) и план за миграција од него кон тој „нов“ ИС
* Обезбедување на важни информации потребни за управување и поддршка на процесот

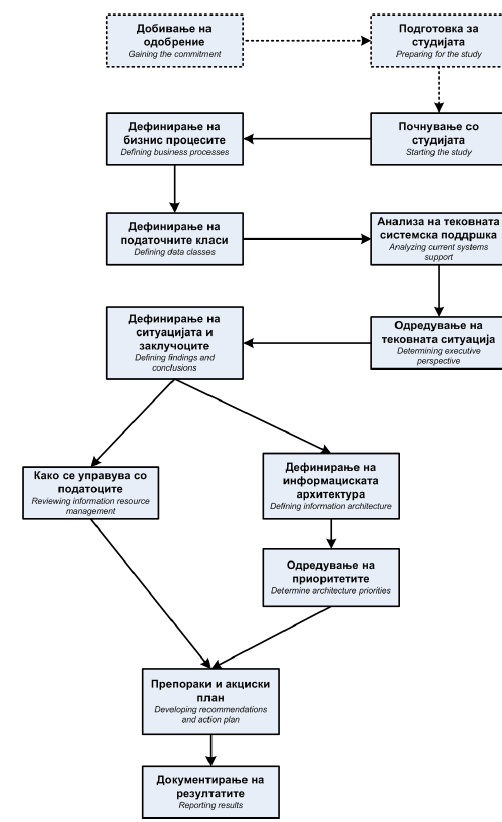
**BSP – метода**

* Развиена од IBM во почеток на 1970-тите.
* Иницијално развиена за интерна употребна од IBM,но подоцна почнуваат да ја нудат на клиенти.
* Методата е фокусирана на податоци во компанијата и посебно на бизнис процесите, што претставувало сосема нов начин на гледање на бизнисот и градење на ИС. Овој пристап е подоцна прифатен од многу други методологии.
* Градењето на ИС со BSP методата е многу детален и целосен процес, па поради тоа е бавно и скапо.
* Top-down пристап за инволвирање на поддршката на луѓето и анализа на работата на претпријатието (почнува од врвното раководство па надолу до најниско ниво во организација)
* Bottom-up пристап за имплементација на информацискиот систем е структурирана методологија и претставува дефиниран начин на транслација на бизнис целите во информациски барања.
* BSP методата е дизајнирана за да може да се дефинира **информациска архитектура** на компанијата. Основните градбени единици на архитектурата се:
  + Податочни класи – Категории на логички поврзани податоци кои се неопходни за поддршка и функционирање на бизнисот
  + Бизнис процеси – Групи на логички поврзани одлуки и активности потребни за да се менаџираат ресурсите на бизнисот



* План
* Дизајн
* Имплементација



**Тек на BSP студија**

* Добивање на одобрение
* Подготовка за студијата
* Почнување на студијата
* Дефинирање на бизнис процесите
* Дефинирање на податочните класи
* Анализа на тековната системска поддршка
* Одредување на тековната ситуација
* Дефинирање на ситуацијата и заклучоците
  + Како се управува со податоците
  + Дефинирање на информациската архитектура
  + Одредување на приоритетите
* Препораки и акциски план
* Документирање на резултатите

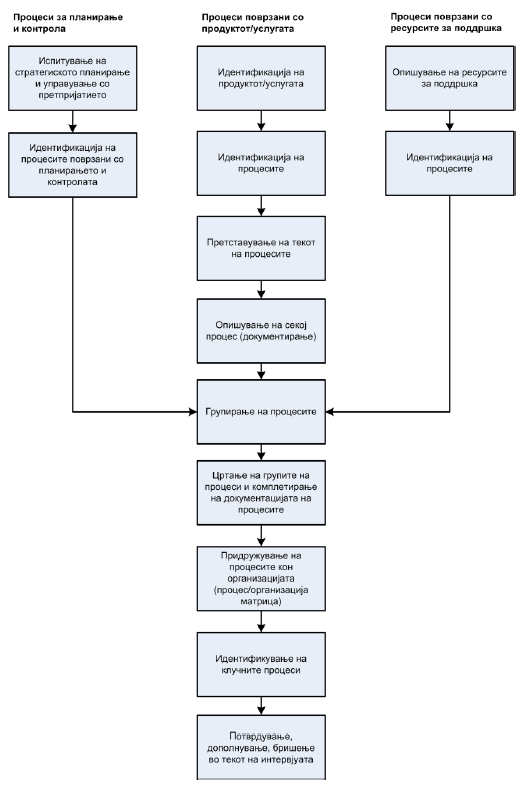
**Определување на бизнис процеси**

* Постојат два генерални начини за идентификување на процесите:
  + **Начин кој е предложен од страна на IBM**
  + **Генерален процесен модел**

**Начин на IBM**

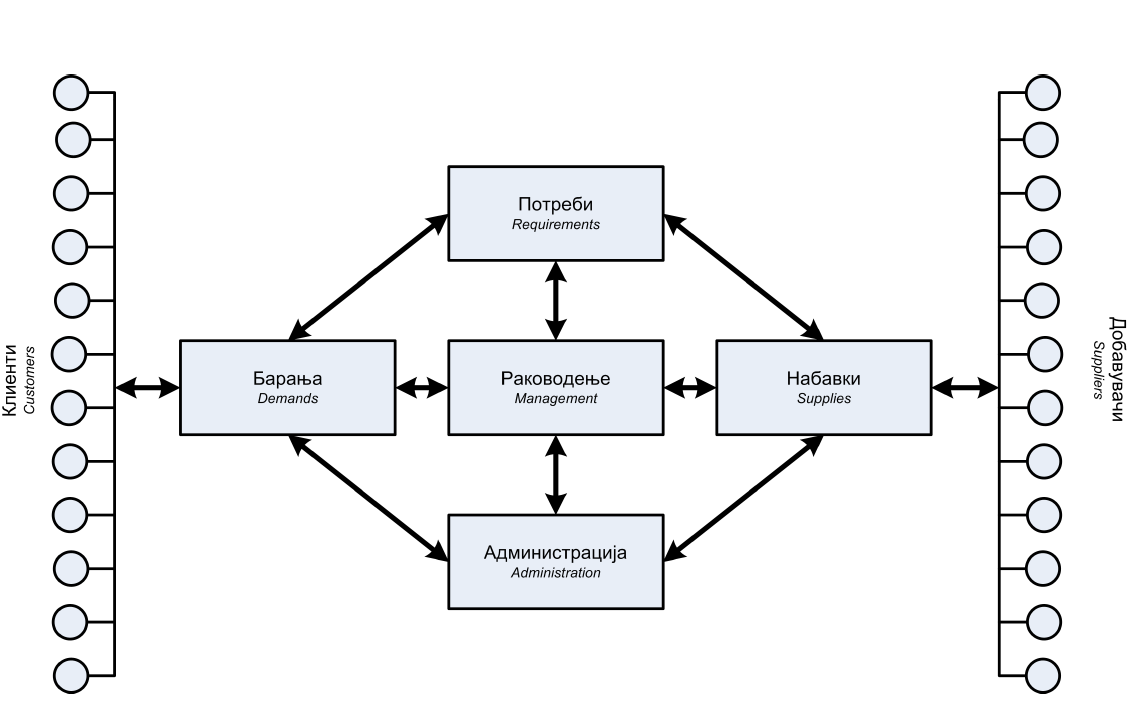
* Сите процеси може да се поделат на три групи:
  + Процеси за планирање и контрола
  + Процеси поврзани со продуктот/услугата што го нуди претпријатието
  + Процеси поврзани со ресурсите за поддршка
* Животен пат на продукт/услуга:
  + Утврдување на потреби, планирање, мерење и контрола (*Requirements*)
  + Набавка или развој – раѓање (*Acquisition*)
  + Манипулација (*Stewardship*)
  + Излез или расходување (*Retirement*)

**Начин на IBM**

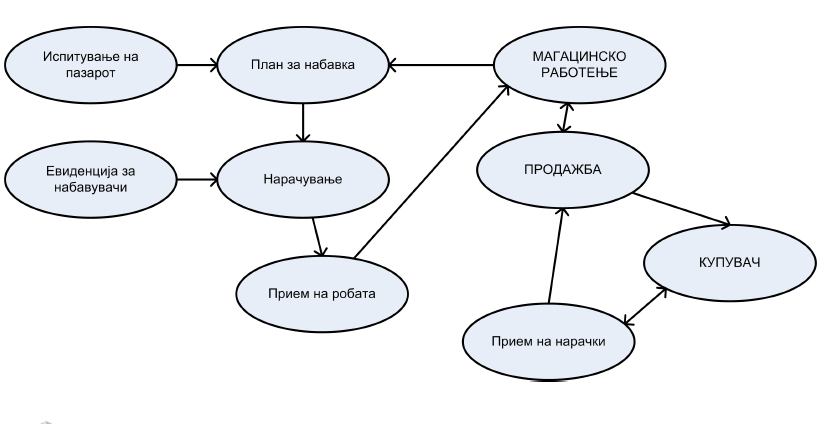


* **Пример:** Ќе ги идентификуваме процесите поврзани со вработените, материјалите и основните средства следејќи го нивниот животен циклус.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Requirements | Acquisition | Stewardship | Retirement |
| Вработени | Кадровско планирање | Вработување | -Исплата  -Усовршување | -Отказ  -Пензионирање |
| Материјали | Планирање на потреби | -Купување  -Прием | Контрола на инвентар | Издавање на материјали |
| Основни средства | План за основни средства | -Купување  -Изградба | Одржување | -Расход  -Продажба |



**Дијаграм на тек на процеси**



**Ентитети и класи на податоци**

* **Ентитети,** нештата што се од траен интерес за претпријатието, околу кои се чуваат и контролираат податоци
* **Класи на податоци,** логичка група на податоци неопходни за поддршка на бизнисот

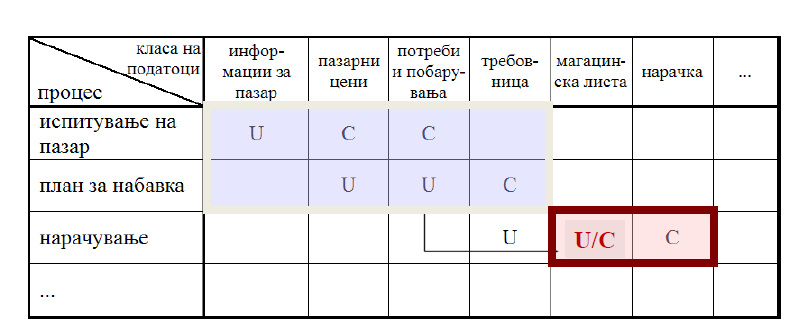
|  |  |  |  |
| --- | --- | --- | --- |
| **Ентитет** | Човек | Добавувач | Производ |
| **Класа на податоци** | -Вработен  -Клиент | Што нуди | -Лагер листа  - Цена |

* Секоја класа на податоци се придружува кон соодветниот бизнис процес и се креира влез-процес-излез табела

|  |  |  |
| --- | --- | --- |
| Користи  U | Процес  P | Креира  C |
| -Информации за пазар (6) | Испитување на пазар | -Потреби и побарувања на пазарот (1)  -Класа пазарни цени (2) |
| -Потреби и побарувања на пазарот (1)  -Пазарни цени (2) | Планирање на набавка | - Требовница (3) |
| - Магацинска листа (5)  - Информации од добавувачи (7)  -Требовница (3) | Нарачување | -Нарачки (4) -Магацинска листа (5) |

**Правила при поврзувањето на податочните класи во процесите**

* Една класа на податоци се креира од еден процес
* Еден процес може да креира повеќе класи на податоци
* Ако ова не е случај, тогаш:
  + Класите се различни ентитети
  + Процесите не се добро групирани
  + Треба да се издвои заеднички дел од повеќе процеси

**Матрица на податоци процеси**

***Пример задачи x2***

## 4. DFD – Data Flow Diagram

**DFD дијаграми општо**

* BSP
  + Идентификација на бизнис процесите
  + Не навлегуваат во нивната комуникација, рамената и чувањето на податоците
* DFD
  + Тек на податоци помеѓу бизнис процесите во еден систем
  + Не нудат контрола на процесите

**Типови на DFD дијаграми**

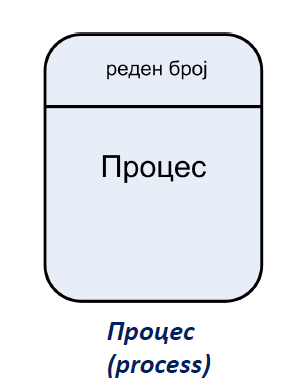
* DFD дијаграми на **физичкиот тек** на податоците
  + Зависат од конкретната имплементација на системот. Тие ги покажуваат конкретните ресурси (уреди, луѓе, организациски едници) и текот на податоци помеѓу нив како што постојат во некој постоечки систем
* DFD дијаграми на **логичкиот тек** на податоците
  + Не зависат од конкретна реализација на системот. Тие покажуваат како би требало да течат податоците во рамките на визијата на некој систем

**Анализа со користење на DFD дијаграми**

* Двата типа на DFD дијаграми (но посебно логичките) користат bottom-up пристап во опишување на системот
* Процесот на анализа, може (не мора) да започне со креирање на физичкиот DFD, со цел да се воочат слабостите кои постојат во претходниот ИС
* Потоа се креира логички DFD на саканиот информациски систем
* Се согледува како саканиот систем функционира концептуално и потоа постепени се оди во поголеми детали на описот л на текот на податоците во системот

**Елементи на DFD нотација**

**Надворешниот ентитет**

* **Надворешниот ентитет** е или извор или одредиште на податоците во/од системот
* Само тие ентитети кои примаат/даваат информации за бизнис процесите се претставуваат со DFD дијаграмот
* Надворешниот ентитет е дефиниран со личност (корисник), организациона едница или организација што е НАДВОРЕШНА во однос на делот од системот што се претставува со дијаграмот, но има некаква интеракција со него
* Со помош на овие ентитети се дефинираат „*границите*“ на системот.
* Симболот за надворешните ентитети е правоаголник во кој се пишува името на ентитетот (обично се именуваат со именки и се означуваат со голем букви)

**Процес**

* **Процесот** врши некаква трансформација на податоците во системот
* Процесот е активност или функција која се случува поради некоја специфична причина
* Идеално, еден процес би требало да се однесува на една активност
* Симболот за процесите во DFD е заоблен правоаголник поделен со линија. Во горниот дел се пишува редниот број на процесот, додека долу се пишува името на активноста што ја прави (се именуваат со глагол после кој следи именка)



**Податочен склад**

* **Податочниот склад** е место за чување на информации внатре во системот
* Во нив мора да влегува барем еден податочен тек
* Можат да чуваат трајна архива на податоци или привремени податоци, кои по одредена манипулација се бришат
* Обично се однесуваат на ресурси, а се имплементираат во рамките на некој систем за управување на бази на податоци како датотеки.
* Симболот е отворен правоаголник кој се нумерира со одреден број, а се идентификува преку името на податочниот склад

**Податочен Тек**

* Податочниот тек го претставува текот на информацијата од нејзиното извориште до нејзиното одредиште
* Податочните текови се претставуваат со линии со стрелки над/под која стои името на текот на информацијата (кое обично се претставува со описни именки)

**Правила за градба на DFD**

* За да се избегне преклопување на линии податочните складишта и надворешните ентитети може да се дуплираат
* Симболите за процесите не е препорачливо да се дуплираат
* Принцип на **промена**
* Принцип на **итерација**

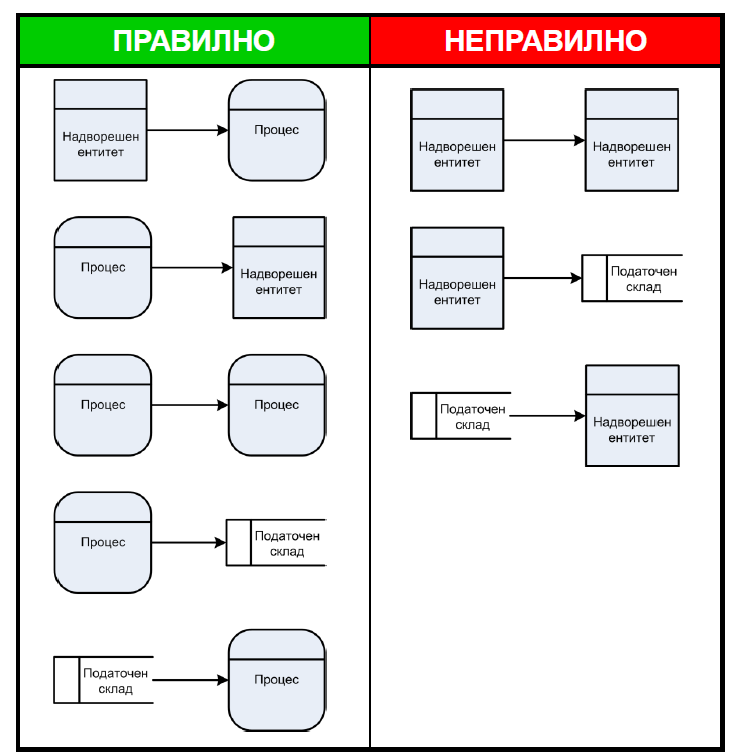
**Дуплирање на елементи**

* Дуплиран надворешен ентитет
* Дуплиран податочен склад
* Дуплиран процес

**Правила за градба на DFD**

**Принцип на промена**

* Не постојат „црни дупки“ и „чуда“. Не постои процес што губи или создава податоци. Секоја податочна структура треба да биде именувана. Имињата треба да ги рефлектираат податоците што се разменуваат помеѓу процесите, надворешните ентитети и податочните складови. Само податоците што се неопходни за извршување на некој процес се влез на тој процес, тој потоа ги користи или менува. При евентуална промена, влезните податоци со помош на некаков алгоритам се менуваат и се креираат излезните податоци кои се праќаат во некој склад на податоци или надворешен ентитет.
* Процесите се секогаш активни (тие ниту започнуваат ниту завршуваат). Доколку постојат влезни податоци, тие ја реализираат функцијата која треба да ја реализираат со нив и креираат излезени податоци. Процесите може да се опишат на високо или на ниско ниво. Процесите опишани на пониско ниво ги детализираат процесите напишано на повисоко ниво. Процесите опишани на најниско ниво опишуваат една функција. Податоците што влегуваат во податоците на високо ниво мора да постојат и во процесите на пониско ниво што ги детализираат.



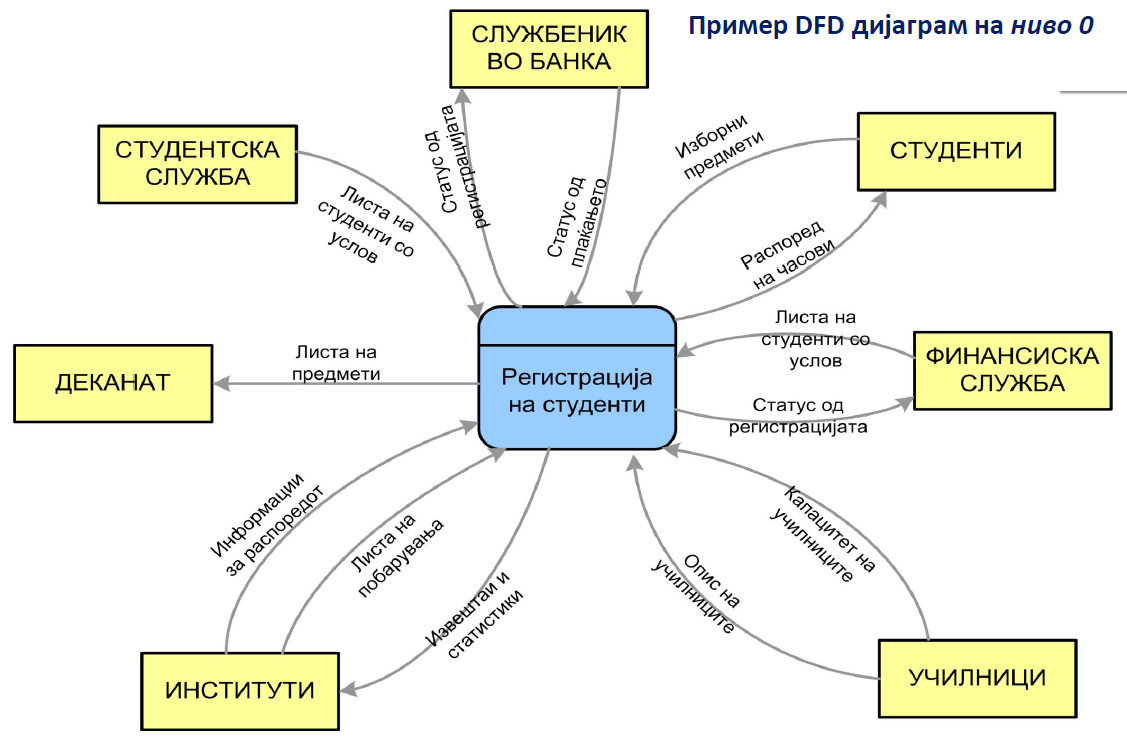
**Постапка**

* Се почнува од најголемите детали на системот – контекстуален дијаграм (ниво 0)
* Се детализира претходниот дијаграм до потребните детали – дијаграми на ниво 1, ниво 2, ...
* Се застанува кога ќе почне да се навлегува во имплементација на одредениот процес

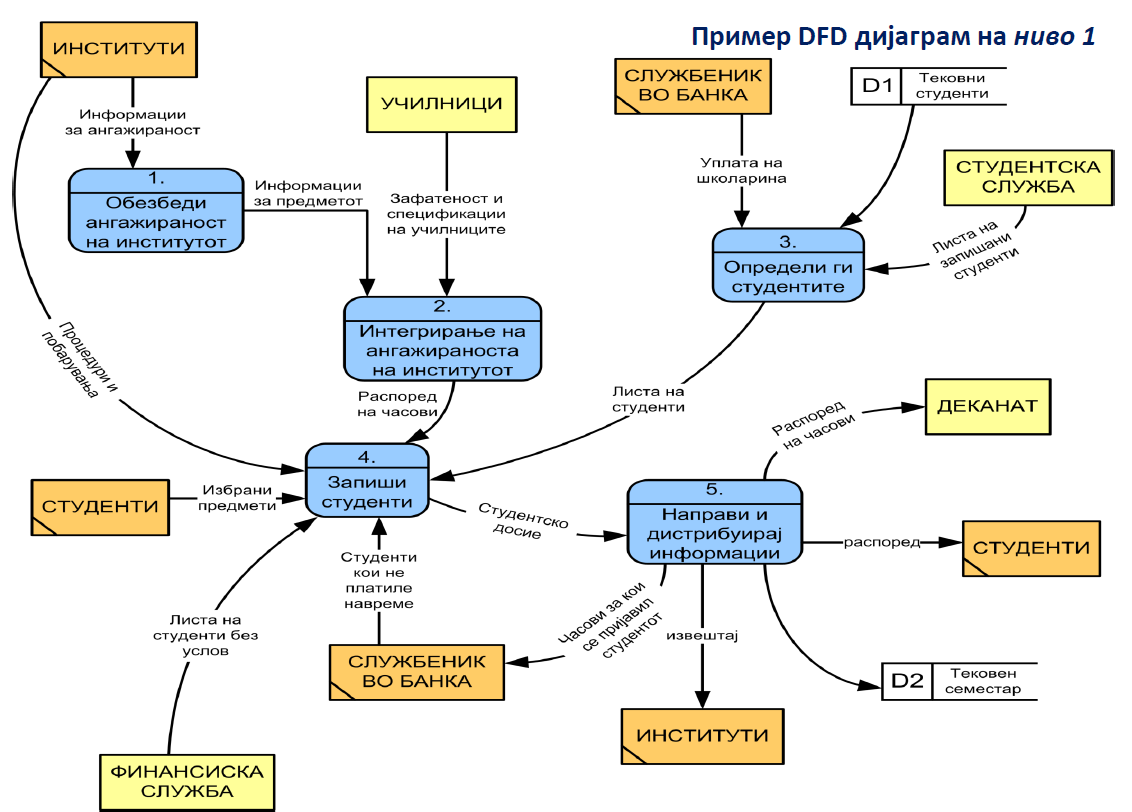
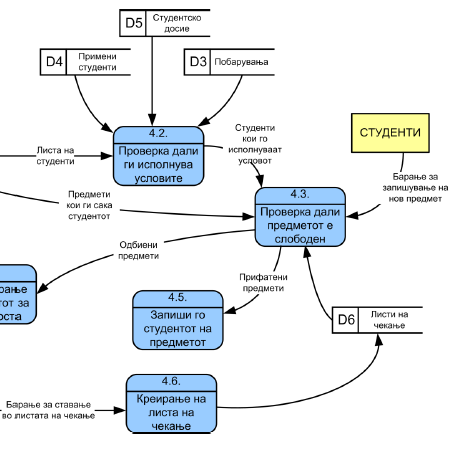
**DFD дијаграм на ниво 0**

* Ги сумираат сите процеси од интерес
* Го опишуваат системот на највисоко ниво
* Ги опишуваат „границите“ на системот од интерес, утврдувајќи ги надворешните ентитети за целиот систем, како и податочните структури што доаѓаат од нив и одат во нив.

Пример



**DFD дијаграм на ниво 1**

* Овие дијаграми ги прикажуваат:
  + Примарните процеси
  + Примарните податочни складови
  + Изворишта и одредишта на податоци (опишани со помош на одредени податочни структури)
* Процесите се нумерираат со броеви
* Податочните складови може да се нумерираат при што се додава D пред бројката

**DFD дијаграм на ниво X**

* Опишување на под-процеси содржани во процесите од погорното ниво
* Процесите се нумерираат со две или повеќе бројки

**Препораки за одредување на системските побарувања со помош на DFD**

* DFD дијаграмите претставуваат хиерархиски организирана нотација која го опишува текот на податоците во одреден систем
* Во тој контекст, логичките DFD дијаграми можат да послужат како основа за екстрахирање (извлекување) на функционалности на бараниот систем.
* За таа цел може да се искористи едноставно и еднозначни пресликување
* **Влезови од надворешните ентитети:** 
  + Системот треба да ги прими податоците за (има на податочен тек) од (име на надворешен ентитет)
* **Излези кон надворешните ентитети:** 
  + Системот треба да ги дава податоците за (име на податочен тек) до (име на надворешен ентитет)
* **Процеси:**
  + Системот треба да: (име на процеси)
* **Внатрешните процеси:**
  + Системот треба да ги дава податоците (има на податочен тек) до (име на ентитет од ниво 2 или пониско)
* **Податоци:** 
  + Системот треба да ги преземе податоците за (име на податочен тек) од (име податочен склад)
  + Системот треба да ги зачува податоците за (име на податочен тек) во (име податочен склад)

***Пример задачи x 2***

## 5. UML – Класен дијаграм

**Историски развој**

* **Simula 1** и **Simula 67** се сметаат за првите ОО програмски јазици
  + Се појавуваат околу 60-тите години од XX век
  + Вклучуваат неколку ОО карактеристики како што се класите и наследството
* Првиот експлицитен ОО програмски јазик е **Smalltalk**
  + Развиен од страна на Alan Klay
  + Ги вклучува механизмите на енкапсулација на атрибутите и размена на пораки помеѓу објектите
* Следејќи го Smalltalk, се појавуваат и останатите ОО програмски јазици како Objective C, C++, Eiffel, CLOS, Java, C#...

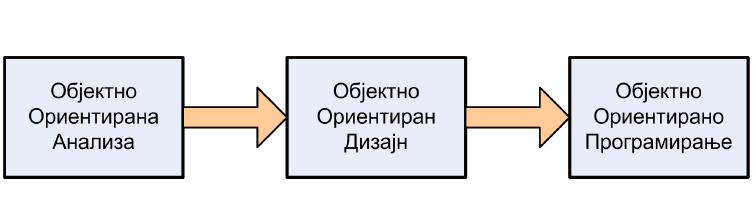
**Недостатоци на ОО програмски јазици**

* Нема доволно ефикасни средства за спецификација и документирање на сложени ООС
* Концепти кои ги нуди ООП се добри, ама недоволно апстрактни
* ООП не нуди поддршка за сите фази на развој на еден ООС
* Текстуалниот начин на документација е помалку ефикасен од визуелниот начин на документација

**Модели и моделирање**

* **Модел** е поедноставен приказ на реалноста
* Моделот се прави со цел подобро да се разбере системот кој се гради
* Моделот на комплексниот систем се прави затоа што таквиот систем многу тешко може да се разбере како компактна целина

**Објектно Ориентирано моделирање**

* Развој на моделот на софтвер на повеќе нивоа на апстракција, преку користење на по апстрактни концепти од оние кои ги нуди ООП
* Спецификација на моделот преку визуелни, графички симболи
* Трансформација на високо-апстрактните, визуелни модели во имплементирачки форми

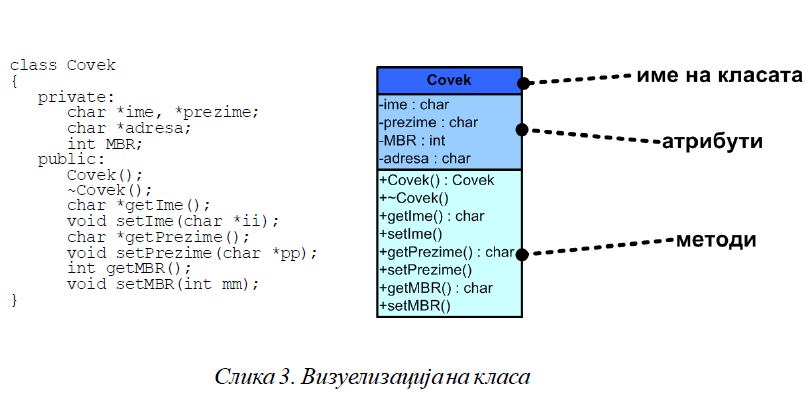
**Историски развој**

* Кон крајот на 80-тите и почетокот на 90-тите години од XX век се појавуваат првите методи за ОО моделирање
* Во 1995 година се припрема првата драфт верзија на UML стандардот, а во 1997 се појавува UML како стандард за ОО моделирање

**Unified Modeling Language – UML**

* UML е визуелен јазик кој се користи во процесот на ОО моделирање на некој ОО систем
* UML се состои од множество на елементи и множество на правила за начинот на комбинирање на елементите
* **Дијаграмот** е графичка презентација составена од множеството елементи. Дијаграмот не ја носи семантиката на системот, туку претставува приказ на сегмент(и) од моделот на ОО системот
* **Составени елементи на UML нотацијата:** 
  + Класен дијаграм
  + Објектен дијаграм
  + Use-case дијаграм
  + Секвенцен дијаграм
  + Колаборациски дијаграм
  + Дијаграм на состојби
  + Дијаграм на активности
  + Компонентен дијаграм
  + Deployment дијаграм

**Класни дијаграми**

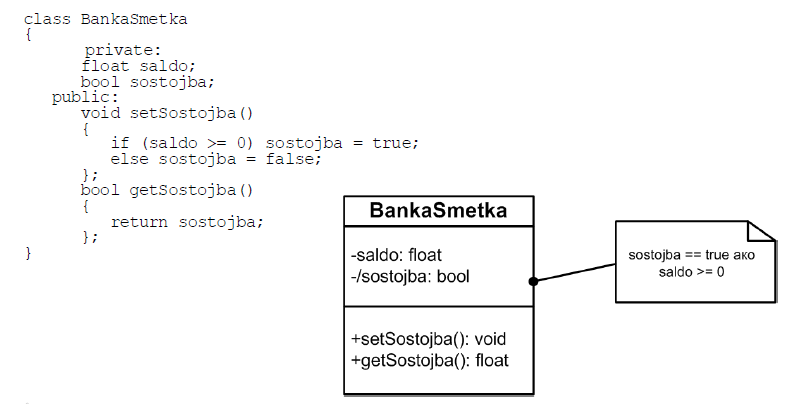
* При визуелизацијата на класите се користат правоаголници поделени на три дела
* Во најгорниот дел се испишува името на класата, во средишниот се дава листа од атрибутите, а во најдолниот дел се дава листата на методи кои го дефинираат однесувањето на класата

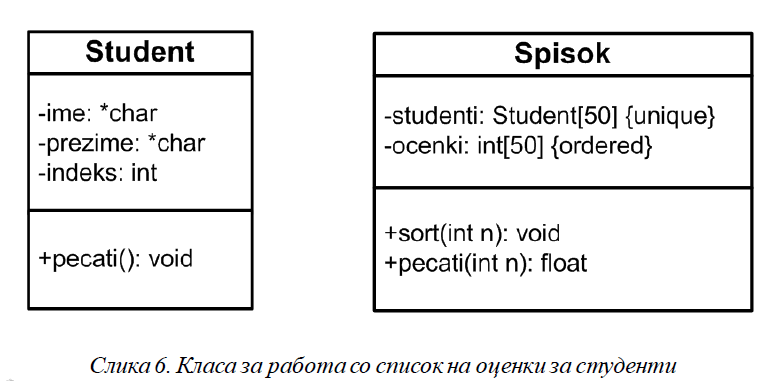
**Атрибутите во класите**

Правила за дефинирање на атрибутите:

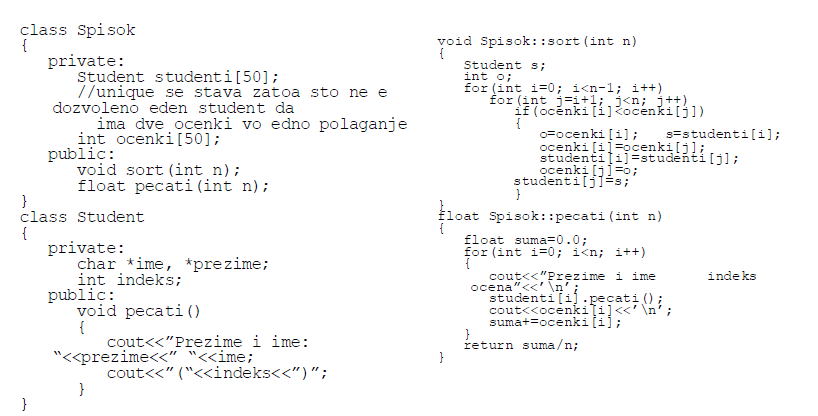
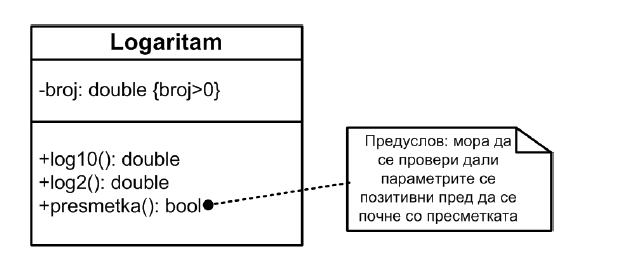
**Visibility / name: type multiplicity = default   
{property strings and constraints}**

**Пример 1:** Класни дијаграми – коментари



**Пример 2:** Класни дијаграми – ограничувања

Имплементација во C++ на пример 2

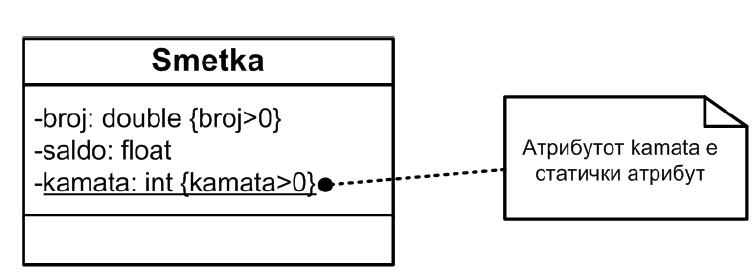


**Пример 3:** Ограничувања на доменот на променливите

**Класни дијаграми – статички атрибути**

**Пример 4:** Статички атрибути

* Статичките атрибути во UML се прикажуваат со повлекување на линија под атрибутот на местото каде што се појавува класата



**Методите во класите**

Правила за дефинирање на методите:

**Visibility name (parameters): return – type {properties}**

Каде што parameters листата се пишува во следниот формат:

**Direction parameter\_name: type [multiplicity]  
=default\_value {properties}**

Секој параметар од листата на параметри на методата може да биде од следниот облик (зависност од direction):

**In** – влезен параметар (по вредност)

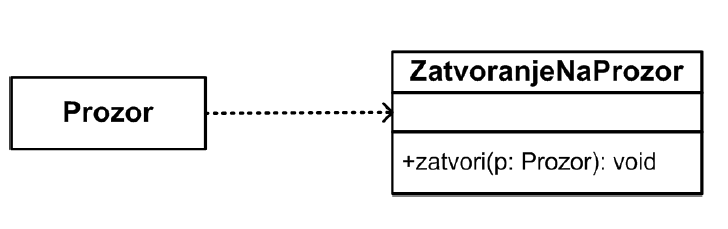
**In out** – влезно излезен параметар (по референца)

**Out** – излезен параметар (по референца)

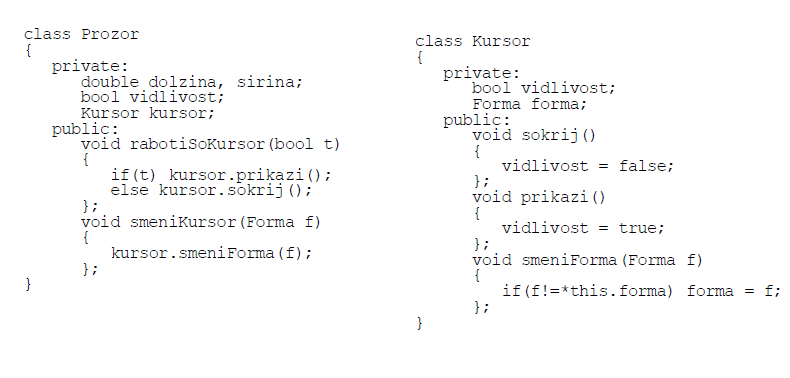
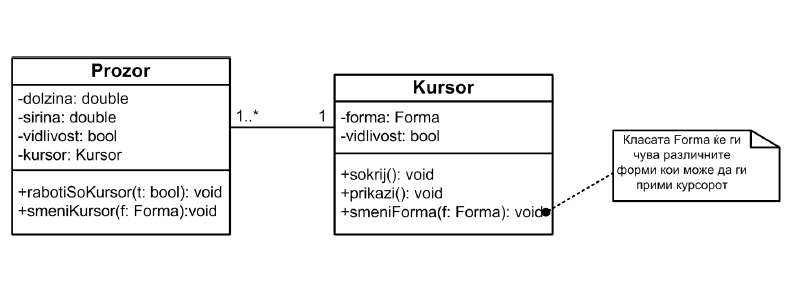
**Врски меѓу класите**

UML стандардот дозволува приказ на врски, релации, помеѓу класите во класниот дијаграм. Постојат следниве начини на поврзување на класите помеѓу себе:

* **Зависност** (dependency)
* **Асоцијација** (association)
* **Генерализација** (generalization)
* **Агрегација** (aggregation)

**Зависност**

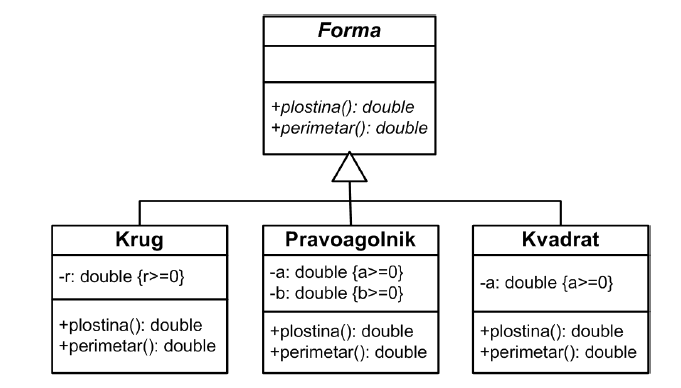
**Асоцијација**

* Асоцијацијата е посилна врска од зависноста и претставува врска помеѓу класите која се одржува со тек на времето
* Објектите не се толку силно врзани, така уништувањето не еден објект не влијае врз постоењето на другите објекти
*  **Се прикажува со полна линија помеѓу двете класи**

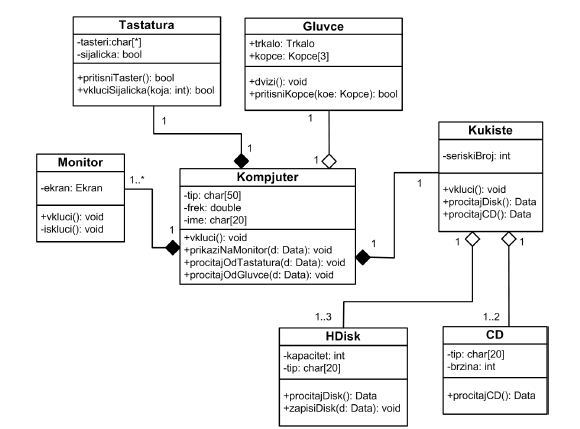
**Наследување**

* **Генерализацијата** е релација за визуелизација на наследството на класите
* Се претставува со полна линија со непополнета триаголна стрелка насочена кон родителската класа

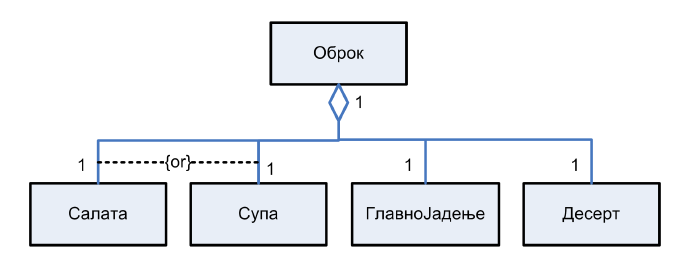
**Пример 5:** Наследување на класа Форма



**Композиција и агрегација**

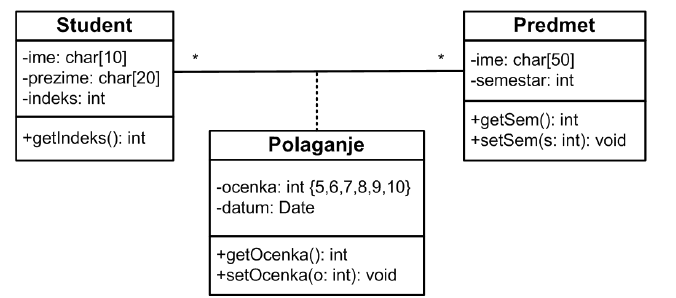
* **Агрегацијата** е појака верзија на асоцијацијата (типот на врската е поседувањето на класата)
* Посилна варијанта на агрегацијата е **композицијата** преку која се прикажуваат составните делови на класата
* И двете врски се прикажуваат со линија која на едната страна има ромб

**Класни дијаграми**

* Понекогаш може да се стават ограничувања во припадноста на составните делови во целината. Ограничувањата се претставуваат со испрекинати линии помеѓу асоцијациите, а во големи згради се става типот на ограничувањето {and, or, not..}

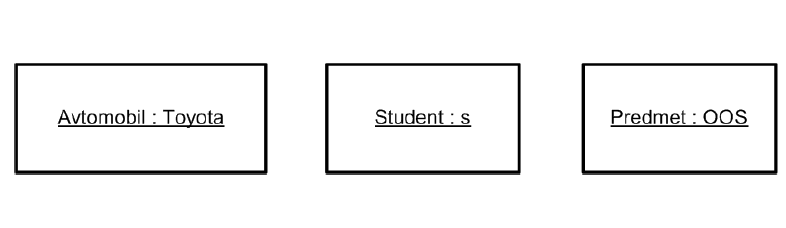
**Асоцијативни класи**

* Асоцијативните класи се асоцијации кои имаат име и атрибути, како и нормалните класи. Асоцијативната класа се прикажува исто како и обичните класи, само што со испрекината линија се поврзува со асоцијацијата на која се однесува.

**Пример** за асоцијативна класа

**Објектни дијаграми**

* Објектите се визуелизираат слично како и класите, само што кај нив се пополнуваат вредностите на атрибутите, и се става името на објектот заедно со класата од која е инстациран. Најчестата претстава е преку правоаголник, без атрибути и методи, во кој се пишува името на објектот, предводено од името на класата и две точки. Целиот текст кој се користи за именување на објектот се подвлекува.

**Пример** за визуелизација на објекти