## Изпитна тема № 10: Разработка на софтуер

**План-тезис:** Трислоен модел и MVC. Концепция за тестване и писане на компонентни тестове. Концепция за дебъгване, откриване и отстраняване на грешки. Концепция за рефакториране и правене на „инкрементални промени“.

**Трислоен модел**

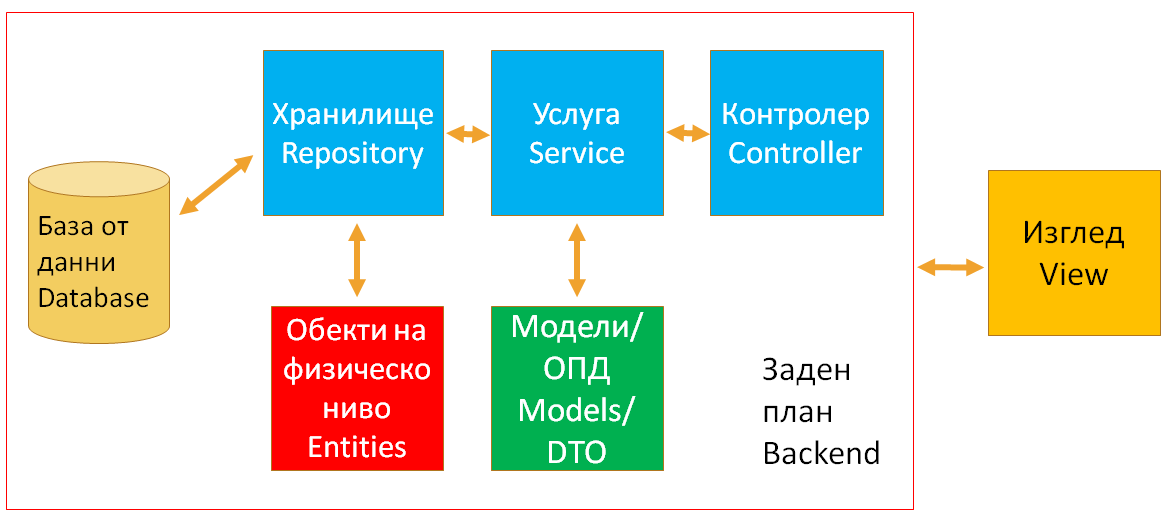
* Разпределя приложението на слоеве
* Всеки слой има строго определена задача
* **Слой за данни (Data Access Layer)**
  + Отговаря за връзката с БД
  + Съхранява данните
  + Изпълнява заявки и команди върху БД
  + Не позволява данните да бъдат достъпвани и манипулирани директно от клиента в презентационния слой
  + Предоставя възможност за управление на информацията без значение от съхраняващия механизъм
  + Носи ползи за мащабируемостта и поддръжката на приложението
* **Слой за услуги/бизнес слой/логически слой (business logic layer/service layer/middle layer/logic layer)**
  + Отговаря за:
    - * Обработка на данните приети от презентационния слой
      * Заявяване на данни от слоя за данни
      * Обработка на получените данни от слоя за данни
      * Подготовка на данните за презентационния слой
* **Презентационен слой (presentation layer)**
  + Отговаря за:

Въвеждане на данни от потребителите

Визуализиране на данни към потребителите

* + Може да бъде графичен потребителски интерфейс, уеб страница и др.

**MVC (Model-View-Controller)** - Шаблон за дизайн (Design Pattern) с три независими компонента

* **Модел (Model)** – управлява данните и логиката на приложението
* **Изглед (View)** – слой за представяне
* **Контролер (Controller)** – преобразува входните данни в команди и ги изпраща към Изгледа или към модела
* Прилики с трислойния модел
  + Разделя на три части
* Разлики с трислойния модел
  + При MVC моделът и изгледът могат да са свързани, постигайки триъгълна структура
  + При трислойния модел задължително се преминава през слоя за услугите, постигайки линейна структура

**Ръчно тестване -** Ръчните тестове са:

* По-малко ефективни
* Неструктурирани
* Неповторяеми
* Не покриват целия ваш код
* Не е лесно да се направят както трябва да бъде

**Компонентно тестване**

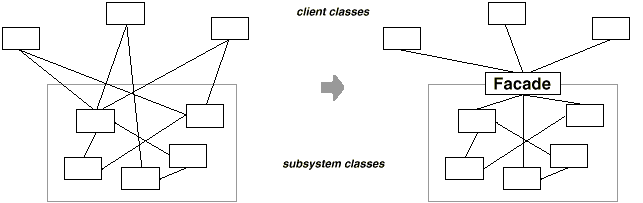
* **Компонент** - код, явяващ се възможно най-малка част от приложението, която може да бъде самостоятелно тествана. В контекстът на C# това обикновено са методите
* Тестовете са парчета код (част от програмата)
* В повечето случаи компонентните тестове са писани от разработчиците (програмистите)
* Компонентните тестове се поставят в хранилището на кода заедно с кода, който тестват. Обикновено се обособяват в собствена папка и проект
* Всички класове трябва да бъдат тествани
* Всички методи трябва да бъдат тествани
* Тривиалният код може да се пропусне –   
  свойства като getters и setters
* Частни методи могат да бъдат пропуснати
* В идеалния случай всички компоненти тестове трябва да минат успешно преди да се качи (commit) кода в хранилището за управление на кода
* Множествена проверка в единичен тестов случай не е препоръчителна
* Ползи:
  + Драматично намаляват броя на дефектите в кода
  + Подобряват дизайна
  + Добра документация са
  + Намаляват цената на промените
  + Позволяват рефакториране
  + Намаляват вероятността от инжектиране на дефекти чрез рефакториране и промени
* **Името на теста** трябва да изразява специфичните изисквания, които се тестват. То трябва да включва:
  + Очакваното входното състояние
  + Очаквания резултат или състояние
  + Името на тествания метод или клас
  + Започва с Test
* Успешните тестове почти никога не се премахват
  + Те подсигуряват това, че промените в кода не повреждат работещия код
  + Успешен тест трябва да се променя само ако се цели да се подобри четимостта му
* Когато тестовете не минават - това обикновено значи, че има конфликтни изисквания – или самият тест е написан дефектно или в кода, който тества има дефект
* Събщенията в тестовете
  + Трябва да казват какво се очаква да се случи, но не и какво е станало вместо това
  + Доброто съобщение от проверката ни помага да проследим грешките и да разберем по-лесно компонентното тестване
  + Изразяват какво трябва да да се случи и какво не трябва да се случи
  + Не повтарят името на тестовия случай
* Нови методологии, в частност Scrum и XP, са превърнали компонентното тестване в основа на разработка на софтуер
* **Code Coverage** – каква част от кода е покрит с тестове
  + Оптималният резултат за един софтуерен проект е около 80%
  + Във VisualStudio:

От меню [Test] -> Analyze Code Coverage -> All tests

**Дебъгване** - процесът на откриване и поправяне или изолиране на грешки в компютърен програмен код

* Тестване - първоначално откриване на грешки
* Дебъгване - диагностициране и коригиране на коренните причини за грешки, които вече са били открити
* Перфектният код е илюзия
* То е цялостно наблюдение на състоянието на процеса
* Намиране на дефект:
  1. Установяване на грешка
  2. Намиране на кода, предизвикващ грешка
     1. Събирането на данни
     2. Анализира данните и формиране на хипотеза
     3. Определяне как да се докаже или опровергае хипотезата
     4. Доказване или опровергаване на хипотезата от 2.c)
  3. Поправяне на дефекта
  4. Тестване на поправката
  5. Търсене на подобни грешки

**Качество на софтуера**

* Външно качество
  + Дали програмата се държи коректно?
  + Дали връща очаквания резултат?
  + Дали софтуерът е бърз?
  + Дали неговия интерфейс е лесен за ползване?
  + Дали програмният код е достатъчно сигурен?
* Вътрешно качество
  + Дали кодът е лесен за четене и разбиране?
  + Дали кодът е добре структуриран?
  + Дали кодът е лесен за промяна?
* **Висококачествен програмен код**
  + Лесен за четене и разбиране
  + Лесен за промяна и поддръжка
  + Има коректно поведение
  + Добре тестван
  + Добре е проектиран и изграден
  + Добре е документиран или е самоописващ се (Self-documenting)
  + **Силна специализация (strong cohesion)** – един елемент отговаря за една-единствена задача
  + **Слаба зависимост (loose coupling)** - елементите са независими един от друг
* **Конвенции в кода** - формализирани напътствия за стила на писане на програмен код
  + Конвенции за форматиране на кода
  + Конвенции за именуване
  + Най-добри практики
* **Преработка (refactoring)** - подобряване на организацията и качеството на наличния програмен код, без да се променя неговото поведение
  + Постепенен процес, превръщащ лошо написания код в качествен
  + Основни пронципи
    - **KISS (Keep It Simple, Stupid!)** - придържане към простите решения и избягване на излишното усложняване на проектите
    - **DRY (don't repeat yourself)** - намаляване на повторения от всякакъв вид
      * WET (write every time, write everything twice, we enjoy typing, waste everyone's time)– нарушенията на DRY принципа
    - **YAGNI (You Aren’t Gonna Need It)** – не имплементирай нещо, ако не е необходимо
    - **SOLID**
      * **Single responsibility** – Всеки клас трябва да има една-единствена отговорност
      * **Open-closed** - Софтуерните единици трябва да са отворени за разширение, но затворени за промяна
      * **Liskov substitution** - Всеки наследник (подтип) трябва лесно да заменя всичките си базови типове
      * **Interface Segregation** - Много на брой малки интерфейси е по-добре от един голям общ интерфейс
      * **Dependency inversion** - Всички класове трябва да зависят от абстракции и нито един не трябва да зависи от конкретен клас
    - **SoC (Separation of Concerns)** – разделяне на сложно приложение на отделни независими секции, всяка с отделна специфично предназначение
    - **The Boy Scout Rule** - оставете кода в по-добро състояние от това, в което сте го заварили
* **Проектна документация**
  + Външна документация
    - На по-високо ниво от кода
    - Описания на проблема, изисквания, проектиране, дизайн, планове за проекта, планове за тестване и т.н.
  + Вътрешна документация
    - На по-ниско ниво
    - Обяснява клас, метод или част от кода
* **Шаблон Façade**
  + Осигурява опростен интерфейс към по-голям програмен код
  + Интерфейс от по-висок ред скрива сложността на подсистемите
  + Пример:

По-сложния начин:

speakers.On();

speakers.SetSurroundSound(true);

speakers.SetVolume(25/100);

speakers.SetOptions(SoundOptions.BlueRay);

environment.DimLights();

projector.On();

projector.SetMode(Modes.WideScreen);

dvd.On();

dvd.Play(movieName);

С Façade:

homeTheater.WatchMovie(movieName);

* Нива на преработка:
  + Преработка на ниво данни
  + Преработка на ниво израз
  + Преработка на ниво метод
  + Преработка на ниво клас
  + Преработка на ниво интерфейс на класа
  + Преработка на системно ниво