Изпитна тема № 11: Разработка на софтуер

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“.

Трислойният модел е софтуерна архитектура, която разделя приложенията на три логически слоя – презентационен, слой за бизнес логика и слой за данни. Предимството на това разделение, е че всеки слой може да работи самостоятелно, т.е. не е зависим от другите. Това позволява той да бъде разработен на различна технология и да бъде внедрен в среда различна от тази на другите слоеве, което дава повече възможности за надграждане на отделните слоеве и по-ефективна разработка на приложенията, тъй като разработката на единия слой е напълно независима от разработката на другите слоеве. Например, смяна на инструментите, с които се разработва презентационния слой, ще доведе единствено до промени в него. Приложенията използващи тази архитектура също са по-надеждни и сигурни, тъй като евентуален проблем в някой от слоевете не би довел до проблем и в останалите.

Презентационния слой представлява интерфейсът на приложението. Там се случва комуникацията с потребителите. Основната му цел е да показва и събира информация от потребителя, която след това да бъде обработена и записана. Този слой може да работи както в браузър, така и в настолно приложение, така и всеки друг вид интерфейс.

Бизнес слоят се използва за валидиране и обработка на информацията, която е получена от презентационния слой, с различни правила, по които е необходимо да работи приложението. В този слой се съдържа по-голямата част от логиката на програмата. За комуникация между бизнес и презентационния слой обикновено се използват програмни интерфейси (API).

В слоя за данни, обработената от бизнес слоя информация се управлява и запазва. За този слой често се използват сървъри за бази от данни като MySQL, Microsoft SQL Server и други. Съхранението на данните в отделен слой увеличава възможностите за надграждане и подобрява производителността.

Model View Controller (MVC) е архитектурен шаблон, който се използва за разделяне на потребителския интерфейс от данните и логиката и правилата за работа на едно софтуерно приложение. MVC може да се използва както за настолни, така и за уеб приложения. Ако го използваме за уеб приложения заявките се изпращат в контролер, който е отговорен за работата с моделите и това какво точно трябва да бъде върнато към потребителя.

В моделите (models) се дефинира каква информация трябва да съдържа даденото приложение. Те представляват състоянието, в което се намира приложението и в тях и са независими от интерфейса. Моделът управлява данните, логиката и правилата на приложението.

Изгледите (views) се използват за извеждане на данните от модела, чрез потребителския интерфейс.

Контролерите (controllers) се използват за обработване на заявките от потребителя и след това работят с моделите и избират кой изглед трябва да бъде рендериран.

Компонентното тестване (unit testing) е процес в програмирането, при който се тестват отделни части (компоненти) от изходния код. Целта на тестването е да се изолират части от кода, който да бъде тестван дали работят както е очаквано. Това е важно, понеже по този начин могат от рано да бъдат намерени грешки, които биха били трудни за откриване и премахване в по-напреднал етап от разработката. Компонентите, които могат да бъдат тествани са отделни функции/методи или процедури, но обикновено при обектно ориентираното програмиране се тества цял клас. Тестовете се пишат от програмистите и тяхното успешно изпълнение гарантира, че софтуера покрива поставените изисквания. Основните подходи за тестване на кода са ръчен и автоматичен. При ръчното тестване се следват набор от инструкции, които трябва да се изпълнят, за да е сигурно, че компонента работи, както е описано. Автоматизираното тестване се смята за по-ефективно, а за да се извърши то е необходимо да се напише код, който да извърши съответните проверки. В следния пример е създаден компонентен тест на метода Add от класа BasicMaths.

A screenshot of a computer program

Description automatically generated with medium confidence

Дебъгер (debugger) е компютърна програма, която се използва за проследяване на процеса на изпълнение на компютърните програми. Обикновено целта на това проследяване е намирането на грешки. Този тип програми се използват за тестване на програми, за да се предотвратят възможни неизправности в процеса на изпълнение. Дебъгване (debugging) се нарича процеса по откриване и отстраняване на грешките. В Visual Studio за стартиране на програмата в режим на дебъг (debug mode) се натиска клавиша F5. Разработчиците имат възможност да добавят и т.нар. точни на прекъсване (breakpoints), които указват, къде точно да се паузира изпълнението на програмата, за да се открие грешка. В режим на дебъг може да се наблюдават паметта и да се преглеждат и променят моментните стойности на променливите.

Рефакториране (refactoring) е процес на изменение на вече съществуващи структура и формат на изходния код, но без да се променя начина, по който той работа. Това се прави с цел премахване и подобряване на нефункционални части от кода. Основните предимства, които рефакторирането носи със себе си са по-лесна четимост на кода, намаляване на сложността му и повече възможности за надграждане. Обикновено рефакториране се прави, ако кодът има някои смислови и логически проблеми като например дублиране, в различни части на програмата. Това може да бъде подобрено, като съответния код бъде преместен в метод, който да се извиква многократно. Много често основното рефакториране представлява изчистване на кода от всички тези неизправности, като излишни временни променливи, които разработчиците може да са използвали за дебъгване, прекалено дълги списъци с параметри, не групирани данни и т.н.

Обобщен план (Опорни думи)

1. **Трислоен модел и MVC**
   * **Трислоен модел**– три логически слоя: презентационен, слой за данни, слой за бизнес логика. Независими един от друг. Внедрени в различни среди, разработени на различни технологии и т.н. Ефективни, по-сигурни.
     + **Презентационен** – интерфейс, браузър, десктоп, и т.н.
     + **Бизнес слой** – приемане, обработка, изпращане на данни, API.
     + **Слой за данни** – Запазване и управление на данните. SQL Server. Подобрява производителността и възможности за надграждане.
   * **MVC** – Model, View, Controller, разделяне на потребителския интерфейс от данните и логиката и правилата за работа на едно софтуерно приложение.
     + **Model** – какво съдържа приложението, моментното състояние, управлява данните, независими от интерфейса
     + **View** – данните от модела, изведени, чрез UI.
     + **Controller** – Обработват заявките на потребителя. Работят с моделите. Избират коя страница да рендерират.
2. **Концепция за тестване и писане на компонентни тестове -** Тестване на отделни части от изходния код, изолиране, откриване на грешки от рано. Тестват се методи/функции, при ООП обикновено цял клас.
   * **Ръчно тестване** – следване на набор от инструкции
   * **Автоматично** **тестване** – пише се код, който извършва съответните проверки
   * **Пример** – class BasicMath, един метод Sum(a,b). Клас UnitTest1, атрибут [TestClass], два метода Test1, Test2, атрибути [TestMethod], Assert.AreEqual
3. **Концепция за дебъгване, откриване и отстраняване на грешки.**
   * **Дебъгер –** компютърна програма, проследява процеса на изпълнение на програмата. Отстраняване на неизправности.
   * **Дебъгване –** процеса по използване на дебъгер. Visual Studio – F5, точки на прекъсване (breakpoints) – паузират програмите на даден ред, преглед на променливи, памет.
4. **Концепция за рефакториране и правене на „инкрементални промени“**.– промяна на структура, формат на кода, без промяна на целта му.
   * По-лесна четимост, възможност за надграждане.
   * Признаци за лош код.
   * Изчистване на кода от неизправности.

Казус

Даден ви е класа SentenceUtils. Напишете компонентни тестове, които да проверят работата на класа и да помогнат в откриването и отстраняването на евентуални проблеми.

За класа SentenceUtils създайте следните тестове:

* Проверка дали има нужда един символ да се преобрази от главна на малка буква или обратно;
* Проверка дали класа SentenceUtils ще преобразува дадените изрази: „Abc”, “ABC”, “abc”, “aBC”;
* Проверка дали се елиминират двойни празни места.

