# II. Архитектура на облачна система за управление на поръчките от клиенти

В този раздел са представени основни елементи, които да положат фундамент на софтуерно решение. Първа подточка ще разгледа общи принципи на софтуерната архитектура. След това втора и трета ще дадат детайлна характеристика на концептуалните и функционални модели на подсистемите.

## 2.1 Същност, цел и обхват на софтуерната архитектура

В общ смисъл, софтуерната архитектура е структурирано решение, което може да оптимизира общи атрибути на качеството като: висока производителност, сигурност, контрол, управляемост, мащабируемост, достъпност (Ali Babar et al., 2009). Думата „**архитектура**“ често се използва в контекста на нещо от високо ниво, което е отделено от детайлите на по-ниско ниво. За разлика от него, „**дизайнът**“ предполага структури и решения на по-ниско ниво. Основната цел на архитектурата е да поддържа жизнения цикъл на системата. Добра архитектура прави системата лесна за разбиране, разработване, поддръжка и внедряване. Kрайната цел е да се минимизират разходите за целия живот на системата и да се увеличи максимално продуктивността на програмиста. (Martin et al., 2017)

Кларк и Уокър (2001) въвеждат концепцията за композиционни модели като начин за разработка. Структурни елементи и интерфейси, съставящи системата са:

* Обекти – градивни елементи от ниско ниво;
* Комуникационните канали в архитектурата;
* Картографиране структурата на данните от високо ниво;
* Архитектурни стилове (Model-View-Controller, Ориентираната към микроуслуги и други), които ръководят композицията.

В архитектурата на сградите обосновката експлицира основната философска естетика, която мотивира архитекта. В софтуерната архитектура, обосновката вместо това обяснява удовлетворението на системните ограничения. Тези ограничения се определят от съображения, вариращи от основни функционални аспекти до различни нефункционални аспекти като икономика, производителност и надеждност (DE Perry et al., · 1992) .По тази причина дизайнът трябва да обхваща икономически, технологични ограничения и естетически проблеми.

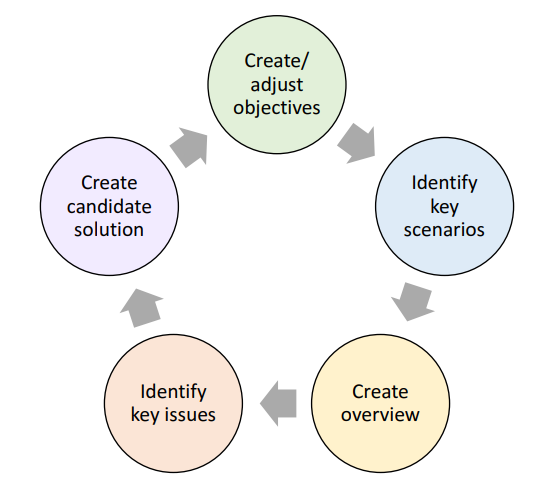
Крайната цел е да се структурира документ от високо ниво, който да не навлиза в подробности за изпълнението. Сложността на операциите трябва да е сведена до минимум, като да обръща внимание на всички изисквания. Приложенията трябва да се съвместими с всички случаи на употреба и бизнес сценарии. Критичните точки трябва да се идентифицират и проучат. Добра практика при проектиране на софтуерна архитектура е да бъде ориентирана към модулност, приемайки лесно новите промени.

***Таблица 2.1****: Принципи на проектиране.*

|  |  |  |
| --- | --- | --- |
| име на български | име на английски | описание |
| Разделяне на грижите | Separation of Concerns | всеки обект и модул трябва да бъде в своя собствена грижа и контекст |
| Капсулиране | Encapsulation |  |
| Инверсия на зависимостта | Dependency Inversion |  |
| Изрични компоненти | Explicit Components |  |
| Единична отговорност | Single Responsibility |  |
| Не се повтаряйте | Don’t Repeat Yourself |  |
| Устойчивост и невежество относно инфраструктурата | Presentation Ignorance |  |
| Ограничени контексти | Bounded Contexts |  |

+ TODO: Препоръки при проектиране

Процесът за проектиране на архитектурa:



* + Създаване или коригиране на цели (определяне обхвата): в тaза част от процеса се включват целите и изискванията на високо ниво, технологии, времето за прекарване, идентифициране на екипа за разработка (ДИРЕКТОР, Разработчици, Функционален анализатор), идентифициране на технически ограничения, (ограничения за използване и внедряване), анализ на технологиите за разработка, идентифициране на потребителите, използващи системата едновременно, трафика, времето за отговор и др.
  + Идентифициране на ключови сценарии: вкючва съществени, но неизвестни към момента, рискове. Пример за това биха били протомотиране на кодове за отстъпка. Също така важни случаи на употреба, критични за бизнеса, които са част от основна бизнес област като например интеграция на плащания. Свързва се с UML диаграми на случаи на използване, които ше бъдат разгледани в точка 2.1.1;
  + Общ преглед – определяне на вида на приложението(ята), идентифициране на ограниченията по внедряване и разгръщане, архитектурните стилове, определяне на технологии, библиотеки и инструменти за разработка;
  + Идентифициране на ключовите проблеми - време на изпълнение, дизайн, производителност, стабилност, управляемост, конфигурируемост и други системни проблеми като удостоверяване и оторизация, кеширане, комуникация, управление на конфигурацията, регистриране и управление на изключения, валидиране.
  + Етап но финалния кандидат за разгръщане на нова версия – ЙЙ: свързан е със всичко напред: (създаване на диаграми).

Унифицираният език за моделиране (Unified Modeling Language) e графичен език за визуализиране, специфициране, конструиране и документиране на елементите на една софтуерна система.

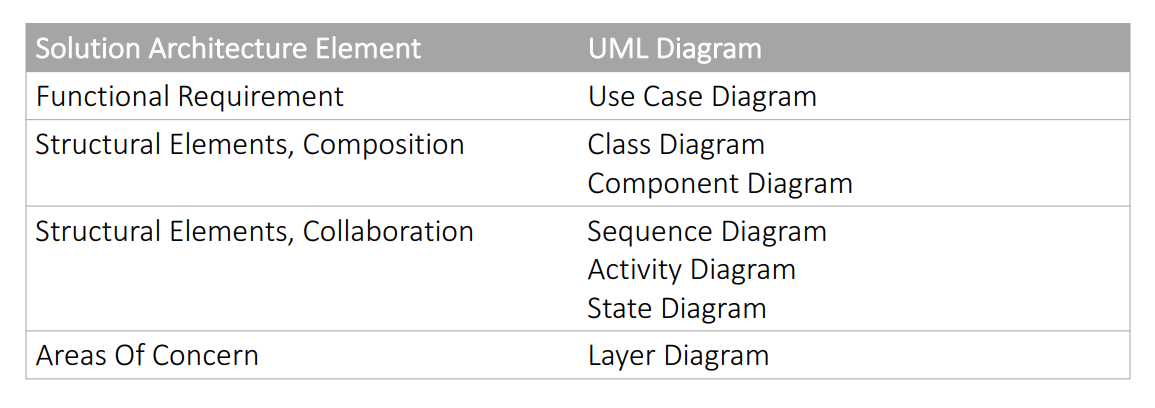
Основни атрибути на UML:

* + Визуално – лесно се вижда представянето на архитектурата;
  + Абстрактно – стои далеч от детайлите на изпълнението;
  + Описателен – показва пълното представяне;
  + Стандарт – UML е отвърден световен стандарт;
  + Поддържа генериране на код – определени секции могат да бъдат конвертирани в код;

Типове UML модели:

* + Бизнес модел – нетехнически, детайлен, приемайки системата като черна кутия.
  + ИТ модел – разделя се на статични (структурни), които изобразяват как се съчетават различните елементи, и динамични (Поведенчески), които се фокусират върху взаимоотношенията.

UML диаграми в архитектурата:



Стратегии за проектиране на UML:

* + UML като скица –предназначен към общи насоки;
  + UML като план - много подробен, може да се пишете код въз основа на диаграмите;
  + UML като валидиране – валидиране на изпълнението спрямо диаграматал;

## 2.2 Концептуален модел на системата

Концептуалните модели са абстрактни представяния за това как трябва да протича изпълнението на задачите. Хората използват концептуалните модели подсъзнателно и интуитивно като начин за систематизиране на процесите (Molina et al., 2000).

+ тодо : https://www.linkedin.com/pulse/conceptual-models-business-dilan-moragolle

### 2.2.1. Структурни диаграми

Структурните диаграми помагат за дефиниране цялостната структура системата, подобно на плана, който определя как изглежда една къща. Структурните диаграми моделират как ще изглежда нашата система в архитектурно отношение. Те ни помагат да дефинираме „речника“ на системата, гарантират съгласуваност от заинтересовани страни в проекта. Идентифицират различни връзки между различните частти на системата.

Структурните диаграми на UML изобразяват елементите на система, които са независими от времето и които предават концепциите на системата и как те се свързват помежду си. Елементите в тези диаграми приличат на съществителните в естествения език и връзките, които ги свързват, винаги показват структурни или семантични връзки. Например, структурна диаграма на система за резервация на превозно средство може да съдържа елементи като кола, резервация, шофьорска книжка и кредитна карта и да съдържа съединители, свързващи тези елементи. Опитните моделисти също ще покажат връзките с поведенческите елементи на тези диаграми.

https://www.sciencedirect.com/topics/computer-science/structure-diagram

#### 2.2.1.1. Диаграма на класовете UML

Kласове, методи и полета, асоциации, обобщения и кардиналност

#### 2.2.1.2. Диаграмата на компонентите UML

Модулни градивни блокове, внедрени и необходими интерфейси

### 2.2.2. Поведенчески диаграми

#### 2.2.2.1. Диаграмата на последователността UML

Последователност от повиквания: извикващ клас, извикан метод и тип на връщаните данни.

#### 2.2.2.2. Диаграмата на състоянията UML

Състояния или дейности, преходи

#### 2.2.2.3. Диаграма на дейността UML

работен процес, едновременни действия, декларации

## 2.3 Функционална структура на системата

### 2.3.1. Подсистема за обслужване на потребителите

Функции на подсистемата, потоци на удостоверяване и оторизация, токени за сигурност, OAUTH 2 и Openid

### 2.3.2. Подсистема за управление на поръчките

осъществяване чрез шаблони:

управляван от домейн дизайн,

разделяне на отговорностите за запитване и за команди

### 2.3.3. Подсистема за управление на доставките

Интернет на нещата: устройства използвани всекидневно и винаги свързани

## 2.4 Характеристика на комуникационните модели - (реферат по дисциплината Интернет технологии и комуникации)

### 2.4.1. Синхронна комуникация

### 2.4.2. Асинхронна комуникация

### 2.4.3. Достъп до подсистемите

# Литература:

1. <https://www.sciencedirect.com/topics/computer-science/structured-solution> (Anil Kumar Thurimella, T. Maruthi Padmaja, in Economics-Driven Software Architecture, 2014)
2. Clean code
3. Clean architecture
4. Foundations for the Study of Software Architecture Dewayne E. Perry AT&T Bell Laboratories 600 Mountain Avenue Murray Hill, New Jersey 07974 dep@research.att.com Alexander L. Wolf
5. JG Molina, MJ Ortín, B Moros, J Nicolás. (2000) *Towards use case and conceptual models through business modeling*. Conceptual Modeling, 2000 - Springer.