# Тема 27. Архитектура на софтуерни системи

1. Софтуерни технологии центрирани около софтуерната архитектура. Качествени атрибути
2. Компоненти и конектори. Типове конектори и техните променливи характеристики
3. Критерий за избор на подходящи конектори. Архитектурни стилове и дизайн за постигане на ефективност, сложност, скалируемост, хетерогенност, адаптируемост, надеждност и сигурност
    a. Разпределени, мрежови, децентрализирани архитектури
        i. Архитектури, ориентирани към услуги и уеб услуги
        ii. Клиент – сървър
4. Анализ и визуализация на софтуерна архитектура
    a. ATAM, CBAM
    b. Графични ADLs, UML

# 1. Понятие за софтуерната архитектура. Качествени атрибути

Софт. технологии предполагат приложението на структуриран, добре дефиниран процес, който да води до предвидимост на разработката на продуктите. Важна част от дейностите в процеса е проектирането на софтуера, чийто основен резултат е софтуерната архитектура, която зависи най-силно от изискванията за качество, т.е. **качествените атрибути**.
Различните качествени изисквания налагат различни ограничения, на които отговаряме чрез специфични дизайни на СА.

**def.** Архитектура на дадена софтуерна система е съвкупност от структури(изгледи), показващи различните софтуерни елементи на системата, външно видимите им свойства и връзките между тях.

- Изглед представлява конкретно документирано представяне на дадена структура

Предмет на СА е **поведението и връзките между различните елементи** на системата, разглеждани като **черни кутии.**

# 2. Компоненти и конектори. Типове конектори и техните променливи характеристики.

Логическият изглед на СА има 4 нива на абстракция(от най-ниско към най-високо ниво):

1. **Компоненти и конектори**
2. Техните интерфейси
3. Архитектурни конфигурации - специфична топология на взаимосвързани компоненти и конектори
4. **Архитектурни стилове** - образци за успешни и практически доказани конфигурации

**def.** Софтуерният компонент е изчислителна единица, която има определена функционалност, която е достъпна чрез добре дефинирани интерфейси и има изрично специфицирани зависимости(входен и изходен интерфейс)

**def.** Софтуерният конектор е first class entity, което задава механизма за взаимодействие между компонентите и правилата за комуникация.

Конекторите са независими от приложението, те са правила, докато компонентите осигуряват специфична за приложението функционалност.

Можем да разделим конекторите на няколко типа спрямо ролите им, т.е.:

- Конекторите като *комуникатори*

    - Разделят комуникацията от изчисленията, примери са RPC, message passing, shared data
    - Изборът може да повлияе на производителността, мащабируемостта и сигурността на системата
- Конекторите като *координатори*
    - определят контрола над изчисленията
    - Разделят контрола от изчисленията
- Конекторите като *конвертори*
    - Правят възможно взаимодействието между независимо разработени и несъответстващи си компоненти, напр. Adapters и Wrappers са такива
- Конекторите като *фасилитатори*
    - Правят възможно взаимодействието между компоненти, които са предназначени да работят заедно
    - Улесняват load balancing и механизми за синхронизация
    - Примери са медиатори и оптимизатори в поточни взаимодействия

# 3. Критерий за избор на подходящи конектори. Архитектурни стилове и дизайн за постигане на **ефективност**, **сложност**, **скалируемост**, **хетерогенност**, **адаптируемост**, **надеждност** и **сигурност**

**Критериите за избор на подходящи конектори**, които са следователно определящи и за избора ни на архитектурен стил, са качествените изисквания/атрибути.

**def.** Архитектурният стил определя семейство от системи по отношение на модел(**образец**) на структурната организация.

Определя в частност речника от **компоненти и конектори**, които могат да се използват в екземпляри от този стил, както и описания как те могат да бъдат комбинирани, като семантика на изпълнението(напр. дали процесите могат да се изпълняват паралелно), топология на описанията(напр. без цикли).

### 3.1. Shared Data стил

Активно се използва в системи, където компонентите трябва да прехвърлят големи количества данни
*Споделените данни могат да се разглеждат като конектор между компонентите.*
Има 2 вариации:

- Blackboard(черна дъска) - когато някакви данни се изпращат към конектора за споделяни данни, всички компоненти трябва да бъдат информирани за това
- Repository(хранилище) - споделените данни са пасивни, до компонентите не се изпращат известия

**Предимства**

- **Скалируемост** - могат лесно да се добавят нови компоненти
- Високо **ефективен** при обмен на големи количества данни
- **Сигурност** - централизираното управление ни дава по-добри условия за сигурност и архивиране на данните

**Недостатъци**

- Трудно приложим в разпределена среда
- Споделените данни трябва да поддържат единен модел на данни
- Промените в модела могат да доведат до ненужни разходи
- Тясна зависимост между blackboard и източника на данни
- Може да се превърне в тясно място(bottleneck) в случай на твърде много клиенти

3.2. Разпределени, мрежови, децентрализирани архитектури

### 3.2.1. Клиент – сървър

Системата е проектирана като набор от сървъри, които предлагат услуги и редица клиенти, които използват тези услуги, като за сървърите не е необходимо да имат информация за своите клиенти.

В зависимост от това каква част от функционалността се реализира от клиентската или сървърната страна, имаме две вариации на стила:

- ***Thin*** client - клиентът реализира функционалността на потребителския интерфейс, а сървърът реализира функцията за управлението на данни и приложната обработка
- ***Fat*** client - клиентите могат да внедрят част от функционалността за обработка на приложения

***Трислойният клиент-сървър*** е модел на многослойна компютърна архитектура, в която цялото приложение се разделя в три различни изчислителни слоя или нива. Разделя слоевете на презентация, логика на приложение и обработка на данни на клиентски и сървърни устройства.

**Предимства**

- По-добра **производителност**, като следствие от централизацията
- **Сигурност** - при клиента и при сървърите, лесно архивиране и възстановяване

От гледна точка на **надеждност**, в смисъл на **отказоустойчивост** - има необходимост от излишък(напр. допълнителен сървър) като тактика за постигане на отказоустойчивост.

### 3.2.2. Архитектури, ориентирани към услуги и уеб услуги. Микроуслуги

Преди архитектурите, ориентирани към услуги, архитектурите се характеризират с висока свързаност, т.нар. Monolith. Тенденцията за децентрализация на управлението и дистрибутираност води до стила на Микроуслугите.
Добавянето на нови функционалности, с течение на времето увеличава значимо **сложността** при Монолита, нарастват усилията за въвеждане на функции, така че връзките между компонентите да бъдат безконфликтни.
За разлика, характерно за Микроуслугите е, че те в контекста си са малки, независими една от друга и от централната система, независими от конкретна технология, и си комуникират стриктно чрез своите интерфейси.

Това **намалява сложността** от гледна точка на комуникация и дава възможност за по-голяма **скалируемост**, което се дължи на улесненото добавяне на функции и по-простия начин на комуникация.
Също така осигурява **хетерогенност** на услугите(те са независими), както и по-добра **адаптируемост**, компонентите се заменят лесно.

Заради силната зависимост от външни услуги и фактът, че всяка комуникация е мрежова, тук нивото на **надеждност** намалява.

# 4. Анализ и визуализация на софтуерната архитектура

## 4.1. Анализ

Представлява начини да оценим дали вече така проектираната и документирана архитектура ще доведе до система, която удовлетворява изискванията.

- *ATAM*(Architecture Tradeoff Analysis Method)
  Разкрива до каква степен архитектурата удовлетворява индивидуалните качествени изисквания и как архитектурните решения си взаимодействат и съотв. какви компромиси се правят за това.

- *CBAM*(Cost Benefit Analysis Method)
  В ATAM става въпрос за анализ на компромиси, но реално най-важните компромиси се правят по икономически съображения, а ATAM не засяга въпроса как дадената организация да увеличи печалбите си и да намали риска от даден софтуерен проект.
  CBAM започва там, където ATAM свършва и се основава на резултатите му; т.е. CBAM дава оценка на технико-икономическите аспекти на архитектурните решения.

## 4.2. Визуализация

Някои от езиците за описание на архитектурата имат възможност за генериране на графични изображения на архитектурата, такива са **xADL, ACME**.

След 2000-та година втората версия на UML се разширява така че да обхваща елементи на тези езици на описание.

UML диаграмите представят решенията след ОО анализа и проектирането.
Според йерархията им, някои видове са:

- диаграми на поведението - Activity, State Machine Diagrams, Use Case Diagrams, Interaction Diagrams и др.
- структурни диаграми - Class Diagrams, Object Diagrams, Deployment Diagrams и др.

Например, за следните изгледи, можем да използваме за моделиране следните UML диаграми:

- class диаграми и package диаграми - модулни
- sequence, activity, state charts - структури на процесите
- deployment диаграма - за физическия(на внедряването) изглед