



12.Тест- резюме


Кохезията (Cohesion) е мярка за това колко добре са свързани и работят заедно вътрешните части на един модул в софтуерната архитектура. Тя показва доколко елементите в един модул споделят обща цел или функционалност.

▶ Какво означава това?


- Висока кохезия = Всички части на модула са силно свързани и работят за постигане на една основна цел.
 -  *Пример:* Модул, който изчислява данък – всички функции в него са свързани с тази задача.
- Ниска кохезия = Частите на модула вършат различни, несвързани задачи.
 -  *Пример:* Модул, който съдържа функции за обработка на данни, изпращане на имейл и валидиране на формуляри.

▶ **От най-лоша към най-добра кохезия:**


1. Случайна кохезия (най-нежелана)

- Частите нямат нищо общо. Просто са сложени заедно без връзка.
-  *Пример:* Функция, която едновременно изтрива файл и изпраща имейл.


2. Логическа кохезия

- Частите вършат различни задачи в зависимост от условие (например параметър).
-  *Пример:* Функция, която прави различни неща в зависимост дали входът е число или текст.


3. Темпорална кохезия

- Частите се изпълняват по едно и също време, но нямат логическа връзка.
-  *Пример:* Всички задачи, които се изпълняват при стартиране на програмата.


4. Процедурна кохезия

- Частите се изпълняват в определен ред, но не винаги са силно свързани.
-  *Пример:* Функция, която първо проверява данни, после ги записва.


5. Комуникационна кохезия

- Всички части работят с едни и същи данни.
-  *Пример:* Модул, който обработва информация за клиент – чете, проверява и записва.




6. Последователна кохезия

- Резултатът от една част се използва в следващата.
-  *Пример:* Вземане на данни → обработване → записване.

7. Функционална кохезия (най-желана)



- Всички части работят за една конкретна цел.
-  *Пример:* Функция, която изчислява ДДС. Това е най-чистият и лесен за поддръжка код.

Запомни лесно:


-  **Случайна кохезия** = хаос
-  **Процедурна/Темпорална кохезия** = частична връзка
-  **Функционална кохезия** = идеален, чист код

Icons for Each Type of Cohesion:


1. Случайна кохезия (Coincidental Cohesion)

- **Icon:**  (*зар*) или 
- **Meaning:** Случайни елементи без връзка, както случайно хвърлени зарове.

2. Логическа кохезия (Logical Cohesion)

- **Icon:**  (*стрелки в различни посоки*)
- **Meaning:** Различни логически операции, насочени според условие.

3. Темпорална кохезия (Temporal Cohesion)

- **Icon:**  (*часовник*)
- **Meaning:** Задачи, които се изпълняват по едно и също време.

4. Процедурна кохезия (Procedural Cohesion)

- **Icon:** → (стрелка напред)
- **Meaning:** Последователни стъпки в процедура.

5. Комуникационна кохезия (Communicational Cohesion)



- **Icon:** 📊 (диаграма) или 📡 (комуникация)
- **Meaning:** Всички операции работят с едни и същи данни.

6. Последователна кохезия (Sequential Cohesion) 🔗

- **Icon:** 🔗 (верига)
- **Meaning:** Резултатът от една стъпка се използва в следващата.

7. Функционална кохезия (Functional Cohesion) 🎯

- **Icon:** 🎯 (мишена)
- **Meaning:** Всички части са насочени към една конкретна цел.

🎯 Функционална кохезия

🔗 Последователна кохезия

📡 Комуникационна кохезия

→ Процедурна кохезия

🕒 Темпорална кохезия



Логическа кохезия



Случайна кохезия

В UML (Unified Modeling Language) свойството **Direction** се използва, за да определи **посоката на данните** за параметър в операция на клас. Това показва как данните се предават – дали влизат в операцията, излизат от нея или и двете.

▶ Възможни стойности на **Direction**:

1. **in** → Данните влизат в операцията.
 - **Пример:** Подаване на входни данни към функция.
 - *Програмен еквивалент:* `function calculate(in value)`
2. **out** → Данните се връщат от операцията, но не влизат като вход.
 - **Пример:** Функция, която връща резултат, без да използва стойността като вход.
 - *Програмен еквивалент:* Функция, която само задава стойност на променлива.
3. **inout** → Данните се използват както за вход, така и за изход.
 - **Пример:** Променлива, която влиза с начална стойност и се променя по време на операцията.
 - *Програмен еквивалент:* Използване на референции или указатели в C/C++.

4. **return** → Специален тип, който показва, че стойността е резултат от операцията (връща се като резултат).

- **Пример:** Стойността, върната от функция с return в програмен код.

Обобщение:

- Ако имаш **входни данни** → in
- Ако връщаш **резултат** → out или return
- Ако данните се използват и за **вход, и за изход** → inout

Когато описваме **сценарий на случай на употреба (use case scenario)** в UML или софтуерно моделиране, е **задължително да зададем входни и изходни условия.**

Какво са входни и изходни условия?

1. **Входни условия (Entry Conditions)**

- Това са условията, които **трябва да са изпълнени**, за да може сценарият да започне.
- **Пример:**
 - За сценарий „Потребител влиза в акаунта си“ входното условие е: „Потребителят има съществуващ акаунт.“

2. **Изходни условия (Exit Conditions)**

- Това са условията, които показват **кога и как завършва успешно** случаят на употреба.
- **Пример:**

- За същия сценарий изходното условие е:
„Потребителят е успешно влязъл в акаунта си и е пренасочен към началната страница.“

▶ **Защо са важни?**

- **Осигуряват яснота:** Всички знаят кога започва и кога приключва процесът.
- **Помагат при тестване:** Лесно е да се проверят дали условията са изпълнени.
- **Намаляват грешките:** Предотвратяват обърквания в сложни системи.

UML изглед на потребителските случаи (Use Case View)

показва как потребителите взаимодействат със системата.

Този изглед е полезен за анализ на функционалните изисквания и за планиране на архитектурата на софтуера.

Какво съдържа Use Case View?

1. Диаграми на потребителските случаи (Use Case Diagrams)

- Показват какви функции (сценарии) предлага системата и кои актьори (потребители или други системи) ги използват.
- **Пример:** Диаграма, която показва как потребител може да „Влезе в акаунт“, „Поръча продукт“ и др.

2. Диаграми на сътрудничеството (Collaboration Diagrams)



- Показват **как обектите в системата си сътрудничат, за да изпълнят даден сценарий.**
- Фокусирани са върху взаимодействията между обекти и как те си изпращат съобщения.

3. **Диаграми на класовете (Class Diagrams)**



- Представят **структурата на системата** чрез класове, техните атрибути, методи и връзки помежду им.
- Те показват **какви данни обработва системата и как е организирана.**

Какво НЕ включва?

- **Диаграми на последователност (Sequence Diagrams)** и
- **Диаграми на състоянието (State Diagrams)**
се използват в други изгледи (например за динамично поведение на системата), но **не са част от основния Use Case View.**

Обобщение:

Use Case View включва:

-  **Диаграми на потребителските случаи** – какво прави системата;
-  **Диаграми на сътрудничеството** – как обектите работят заедно;

-  **Диаграми на класовете** – как е структурирана системата.

Това помага да се види ясно **какво трябва да прави системата** и как работи „отвътре“. 