1. Niedrige Kopplung

Kopplung beschreibt den Grad der **Abhängigkeit zwischen Klassen oder Modulen** in einem Software-System. Eine **niedrige Kopplung** bedeutet, dass die Klassen und Module **möglichst wenig voneinander abhängig** sind und nur notwendige Informationen austauschen. Ziel ist es, Klassen so zu gestalten, dass sie unabhängig voneinander geändert oder wiederverwendet werden können, ohne dass Änderungen an anderen Klassen notwendig sind.

- **Niedrige Kopplung** ist wünschenswert, weil:
    - **Änderungen an einer Klasse** keine oder nur geringe Auswirkungen auf andere Klassen haben.
    - **Wartung und Erweiterung** des Systems einfacher sind, weil weniger Abhängigkeiten bestehen.
    - **Wiederverwendbarkeit** von Klassen und Modulen gefördert wird, da sie nicht stark von anderen Teilen des Systems abhängen.

Beispiel für niedrige Kopplung: Wenn du eine Klasse für eine Benutzeroberfläche und eine Klasse für die Datenbank hast, sollten diese so gestaltet sein, dass die Benutzeroberfläche nicht direkt von den internen Implementierungsdetails der Datenbank abhängig ist. Stattdessen könnten sie über Schnittstellen oder APIs miteinander kommunizieren.

2. Hohe Kohäsion

Kohäsion bezieht sich darauf, **wie eng die Funktionen innerhalb einer Klasse oder eines Moduls zusammenhängen**. Eine **hohe Kohäsion** bedeutet, dass alle Methoden und Attribute innerhalb einer Klasse oder eines Moduls **stark miteinander verbunden** sind und **eine gemeinsame Verantwortung** haben.

- **Hohe Kohäsion** ist wünschenswert, weil:
    - Eine Klasse, die eine **klare und fokussierte Aufgabe** hat, leichter verständlich und wartbar ist.
    - Es einfacher ist, eine Klasse zu erweitern oder zu ändern, weil alle Funktionen eng miteinander verbunden sind.
    - Der **Code logisch strukturiert** ist, was die Verständlichkeit und Testbarkeit verbessert.

Beispiel für hohe Kohäsion: Eine Klasse `Rechnung` könnte Methoden haben wie `berechneGesamtsumme()`, `fügeArtikelHinzu()` und `entferneArtikel()`. Diese Methoden gehören alle zur Kernverantwortung der Klasse, nämlich die Verwaltung von Rechnungen, und sind damit kohärent.

Hohe vs. Niedrige Kohäsion

- **Hohe Kohäsion**: Alle Teile einer Klasse arbeiten auf ein gemeinsames Ziel hin. Die Klasse hat eine klare, fokussierte Verantwortung.
- **Niedrige Kohäsion**: Die Klasse enthält Funktionen, die **nicht eng zusammenhängen**. Sie erfüllt zu viele unterschiedliche Aufgaben, was es schwieriger macht, sie zu verstehen und zu warten.

Niedrige Kohäsion kann auftreten, wenn eine Klasse zu viele verschiedene Aufgaben erfüllt. Das ist ähnlich wie bei der **Gott-Klasse**, die für zu viele Dinge verantwortlich ist und dadurch schwer verständlich und wartbar wird. Bei niedriger Kohäsion kann es sein, dass die Methoden und Daten einer Klasse nicht gut zusammenpassen oder zu viele unterschiedliche Verantwortlichkeiten in einer Klasse gebündelt sind.

Zusammenhang zwischen niedriger Kopplung und hoher Kohäsion

- **Niedrige Kopplung** sorgt dafür, dass Klassen oder Module **wenig voneinander abhängig** sind, was die Wartbarkeit und Erweiterbarkeit des Systems erleichtert.
- **Hohe Kohäsion** stellt sicher, dass jede Klasse oder jedes Modul **gut organisiert** und fokussiert ist, also nur eine klare Verantwortung hat.

Diese beiden Prinzipien arbeiten oft **Hand in Hand**: Wenn du Klassen mit **hoher Kohäsion** entwirfst, führt dies in der Regel zu **niedriger Kopplung**. Eine gut strukturierte Klasse, die nur eine Aufgabe hat, neigt dazu, weniger Abhängigkeiten zu anderen Klassen zu haben, was die Kopplung verringert.

Fazit:

- **Niedrige Kopplung** bedeutet, dass Klassen und Module möglichst **wenig voneinander abhängig** sind, was die Wartbarkeit und Erweiterbarkeit erhöht.
- **Hohe Kohäsion** bedeutet, dass die **Funktionen innerhalb einer Klasse eng zusammenhängen** und auf ein gemeinsames Ziel hinarbeiten, was den Code verständlicher und leichter wartbar macht.
- **Niedrige Kohäsion** tritt auf, wenn eine Klasse zu viele verschiedene Verantwortlichkeiten übernimmt, was zu unübersichtlichem und schwer wartbarem Code führt.

Das Ziel ist, **niedrige Kopplung und hohe Kohäsion** zu erreichen, um sauberen, modularen und leicht wartbaren Code zu schreiben.