Designprinciper

1. SOLID
2. DRY (Don’t Repeat Yourself)
3. KISS (Keep It Simple, Stupid)
4. YAGNI (You Aren’t Gonna Need it)
5. Separation of Concerns (SoC)
6. Composition over Inheritance (CoI)
7. Favor Immutability (IMM, FI)
8. Encapsulation
9. Law of Demeter (LoD)
10. Principle of Least Astonishment (POLA)

Designprinciper_**SOLID**

En uppsättning på fem principer som främjar bra objektorienterad design och lätt underhållbar kod:

- **S - Single Responsibility Principle (SRP)**
  En klass bör ha endast ett ansvar, det vill säga bara en anledning att ändras. Detta innebär att varje klass ska ha ett tydligt och specifikt syfte.
- **O - Open/Closed Principle (OCP)**
  En klass bör vara **öppen för utökning men stängd för förändring**. Man kan lägga till ny funktionalitet utan att ändra befintlig kod, vilket minskar risken för buggar och problem.
- **L - Liskov Substitution Principle (LSP)**
  En instans av en subklass ska kunna ersätta en instans av dess basklass utan att ändra programmets korrekta beteende.
- **I - Interface Segregation Principle (ISP)**
  Gränssnitt ska vara specifika för varje användning och inte ha onödiga metoder. Klasser ska inte tvingas att implementera metoder de inte använder.
- **D - Dependency Inversion Principle (DIP)**
  Hög nivå-moduler ska inte bero på låg nivå-moduler, utan båda ska istället bero på abstraktioner (interfaces). Abstraktioner bör inte bero på detaljer; detaljer bör bero på abstraktioner.

Designprinciper_**DRY**

Don't Repeat Yourself

Koden ska inte innehålla onödiga upprepningar av logik eller funktionalitet. Genom att hålla koden DRY minskar man risken för buggar och förenklar underhåll.

Designprinciper_**KISS**

Keep It Simple, Stupid

KISS-principen handlar om att hålla lösningar enkla och undvika komplexa mönster om det inte behövs. Enkla lösningar är ofta mer robusta och lättare att felsöka.

Designprinciper_**YAGNI**

You Aren't Gonna Need It

YAGNI påminner oss om att inte implementera funktioner "för säkerhets skull". Bygg endast vad som krävs just nu, för att undvika onödig komplexitet och hålla systemet lätt att underhålla.

Designprinciper_**SoC**

Separation of Concerns

Denna princip handlar om att separera olika ansvarsområden i olika komponenter, så att varje komponent ansvarar för en tydlig del av systemets funktionalitet. Till exempel bör affärslogik, presentation och dataåtkomst hållas separata.

Designprinciper_**CoI**

Composition over Inheritance

Denna princip förespråkar att man bör föredra **sammansättning** över **arv**. Genom att bygga objekt genom att kombinera mindre, återanvändbara delar (komponenter) istället för arv, kan man skapa flexiblare och mer anpassningsbar kod.

Designprinciper_**IMM_FI**

Favor Immutability

Att arbeta med immutabla objekt (objekt vars tillstånd inte kan förändras efter skapande) ökar förutsägbarheten och reducerar buggar som kan uppstå när data oavsiktligt ändras.

Designprinciper_**Encapsulation**

En grundläggande objektorienterad princip som säger att man ska skydda data inom en klass och endast exponera det som behövs. Detta ökar säkerheten och modulariteten i koden.

Designprinciper_**LoD**

Law of Demeter

Denna princip säger att en metod bör endast prata med sin närmaste omgivning (sina direkta beroenden) och undvika att anropa metoder hos objekt som är flera steg bort.

Designprinciper_**POLA**

Principle of Least Astonishment

Koden ska bete sig på ett förutsägbart sätt och inte överraska utvecklare eller användare. Det ökar kodens läsbarhet och underhållbarhet.

Varför är designprinciper viktiga?

Designprinciper gör det möjligt att bygga robusta, underhållbara och skalbara system. När utvecklare följer dessa riktlinjer minskar risken för buggar, koden blir lättare att läsa och förstå, och systemet blir mer anpassningsbart för förändringar. Designprinciper är en grundläggande del av "Clean Code"-tänkande och gör system mer motståndskraftiga och hållbara över tid.