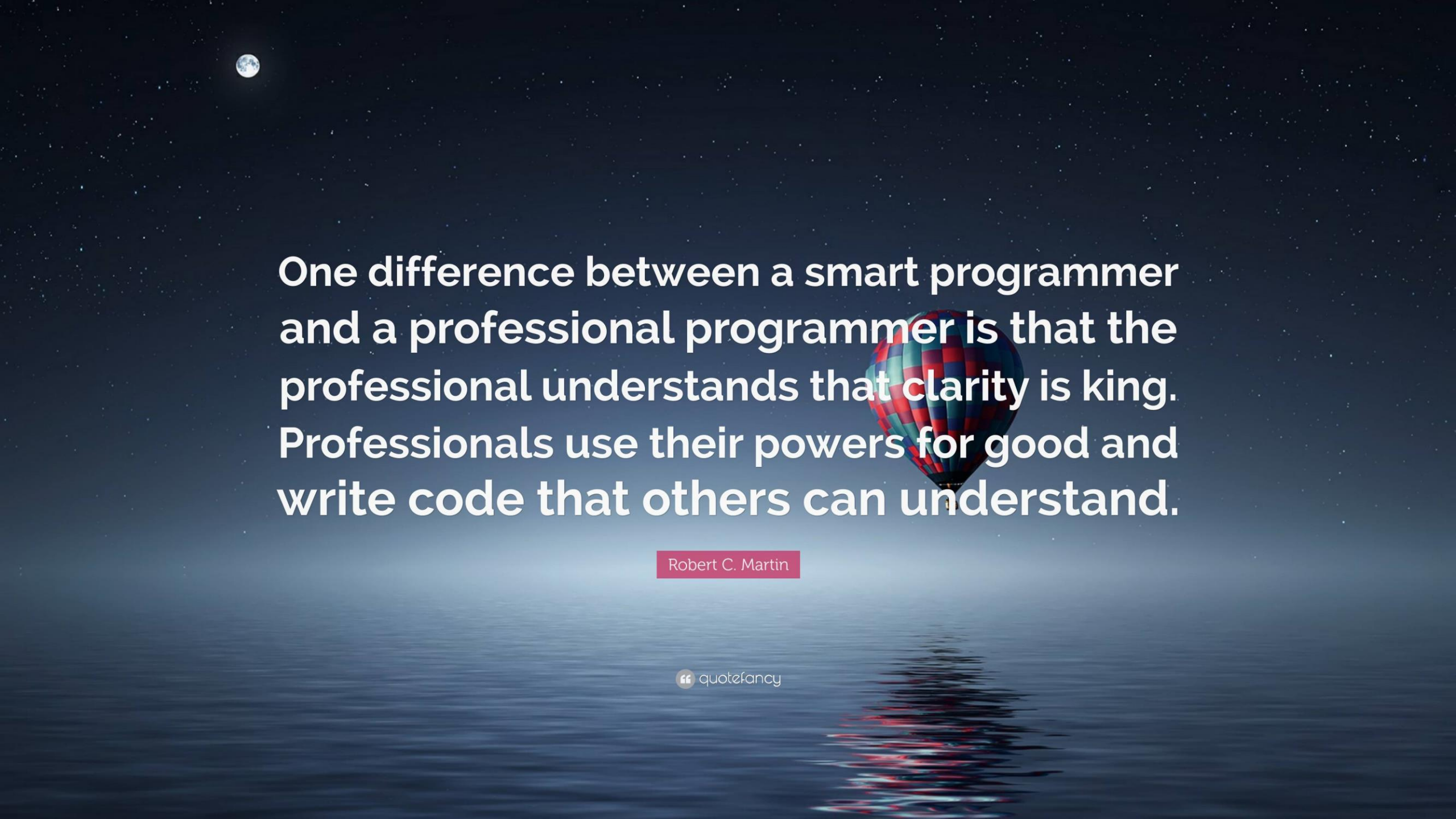


Clean Code

Niklas Hjelm



IT-HÖGSKOLAN
GÖTEBORG



One difference between a smart programmer
and a professional programmer is that the
professional understands that clarity is king.
Professionals use their powers for good and
write code that others can understand.

Robert C. Martin

Vad är Clean Code?

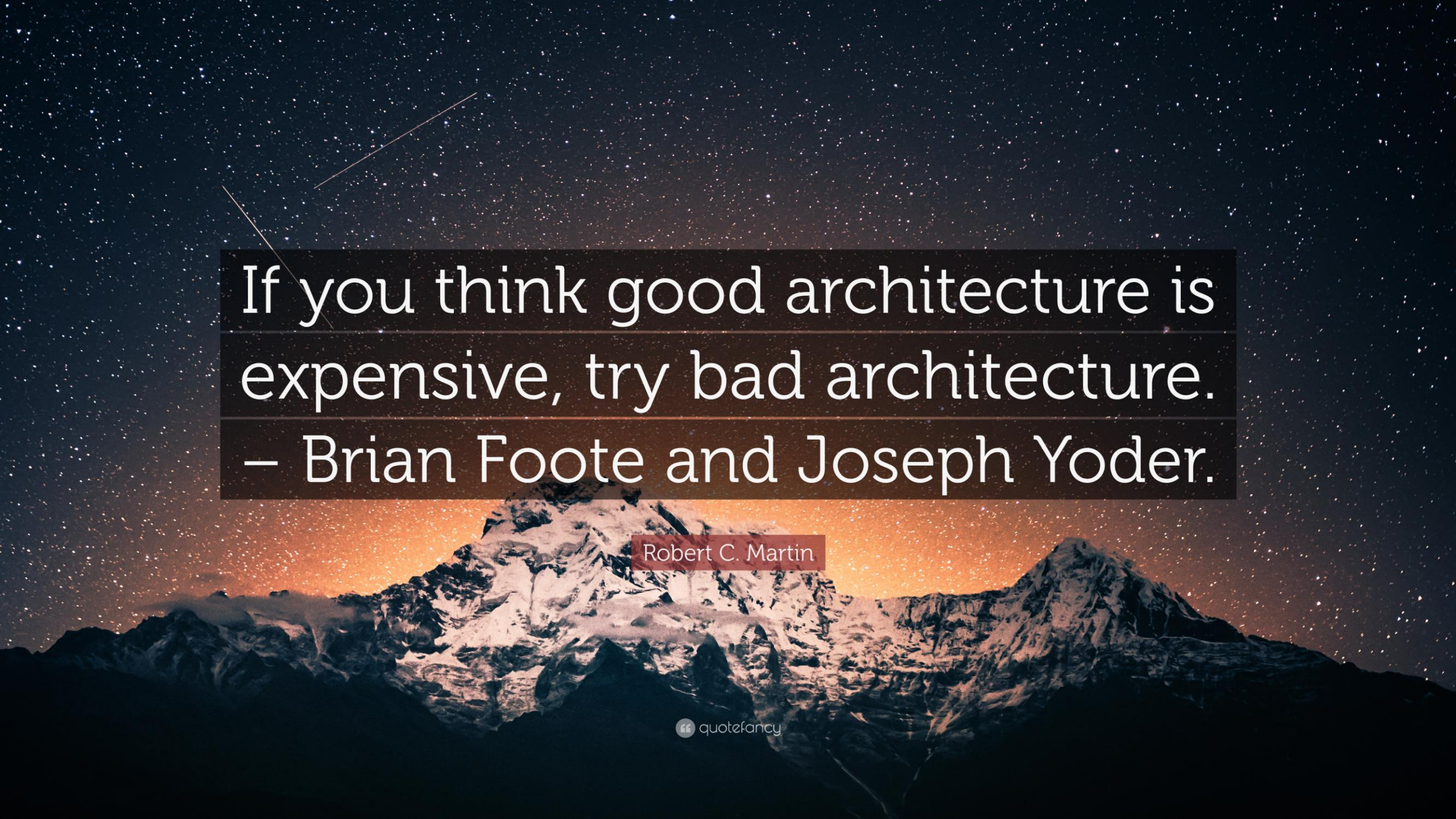
- Kod som är lätt att läsa
- Kod som är lätt att utöka
- Kod som kan återanvändas
- Kod som inte går sönder på oväntade ställen



Vad är smutsig kod? (Code Smell)

- Kod som inte är tydlig
- Kod som är onödigt komplex
- Kod som går sönder på oväntade sätt
- Kod som gör saker den inte ska





If you think good architecture is
expensive, try bad architecture.
– Brian Foote and Joseph Yoder.

Robert C. Martin

Enkla tumregler

- Namngivning är **VIKTIGT!**
 - Välj variabelnamn som beskriver dess innehåll
 - Välj metodnamn som beskriver dess uppgift
- Om du behöver kommentera koden du skrivit för att den ska vara läsbar. Skriv om koden så den går att förstå istället.
- Det är bättre med många små funktioner och metoder än stora oläsliga.

Designprinciper för OOP

- KISS
 - Keep It Simple Stupid
- DRY
 - Don't Repeat Yourself
- YAGNI
 - You Aren't Gonna Need It
- SOLID
 - Mer om detta....
 - NU!

SOLID

- Single Responsibility Principle – Robert C. Martin
- Open/Closed Principle – Bertrand Meyer
- Liskov's Substitution Principle – Barbara Liskov
- Interface Segregation Principle – Robert C. Martin
- Dependency Inversion Principle – Robert C. Martin

Single Responsibility Principle - SRP

- *“A class should have one, and only one, reason to change.”*
– Robert C Martin
- Varje klass och i förlängningen metod eller funktion ska ha bara ett syfte.
- Tips: Om en metod måste beskrivas med ord som och/eller/sen så ska den brytas isär
- Klasser ska inte känna till saker de inte använder

Open/Closed Principle – OCP

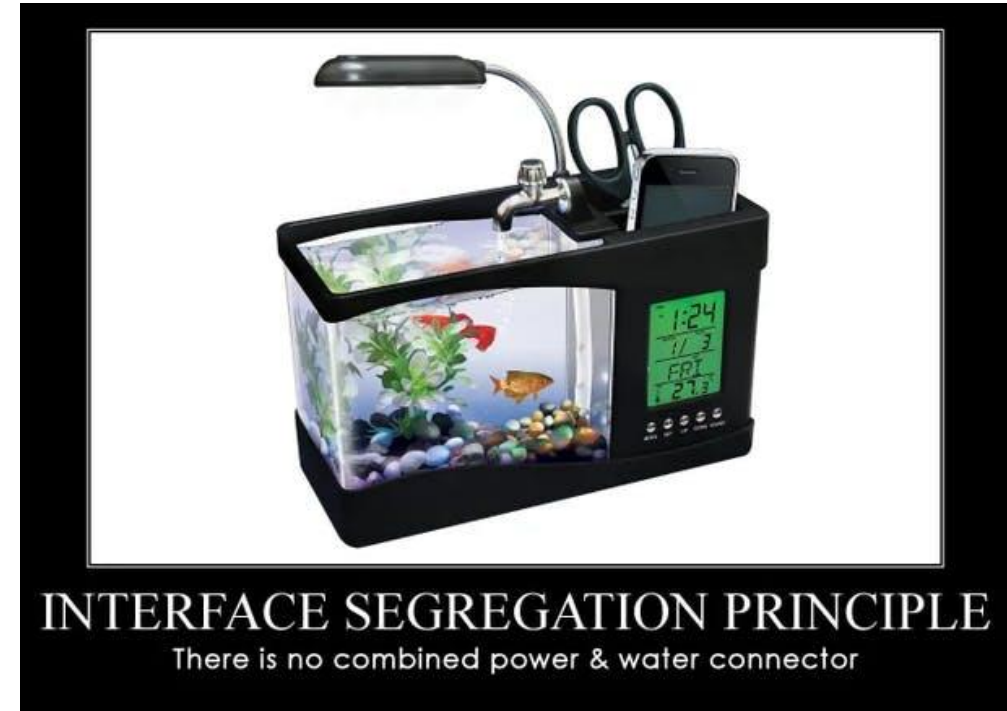
- *“Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”* – Bertrand Meyer
- Denna princip syftar till att man aldrig ska skriva kod som man bara har ”tills vidare”.
- För att förändra eller utöka beteende ska vi bara behöva lägga till eller ta bort kod. Inte skriva om.

Liskov's Substitution – LSP

- *“Let $\Phi(x)$ be a property provable about objects x of type T . Then $\Phi(y)$ should be true for objects y of type S where S is a subtype of T .”* – Barbara Liskov
- Om det ser ut som en anka, later som en anka men behöver batterier så har du troligtvis fel abstraktion.

Interface Segregation Principle – ISP

- *“Clients should not be forced to depend upon interfaces that they do not use.”* – Robert C. Martin
- Bryt isär interfaces så att de går att återanvända.
- Kombinerar flera interfaces istället för att bygga stora interfaces.



Dependency Inversion Principle – DCP

- *High-level modules should not depend on low-level modules. Both should depend on abstractions.*
- *Abstractions should not depend on details. Details should depend on abstractions.*
- En platt arkitektur är lättare att förändra och utöka.
- Inversion of Control med Dependency Injection är ett sätt att göra detta.

Tack!



Referenser och rekommenderad läsning

- Clean Code - Robert Martin
- Clean Architecture - Robert Martin
- Design Patterns – Gang of Four

- Design Patterns - Refactoring.Guru
- Solid Design Principles - Stackify.com