SOLID

Principles

Pietro Balestra 20.10.22

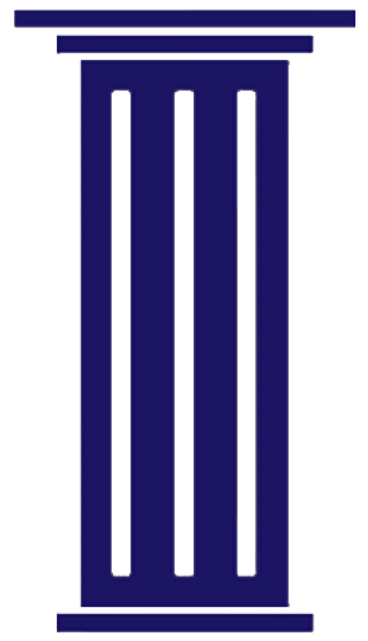
SOLID

Principles

SOLID

Principles

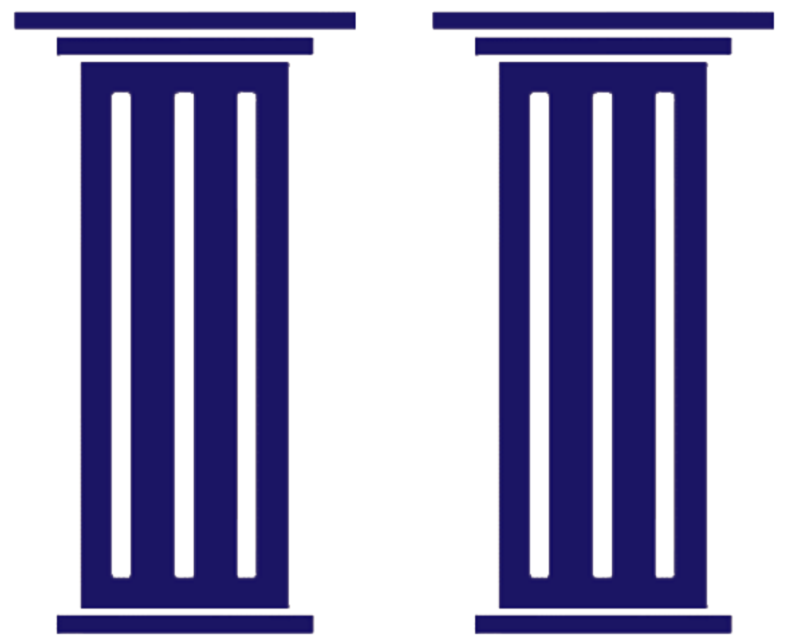
- **S**ingle-responsibility principle



SOLID

Principles

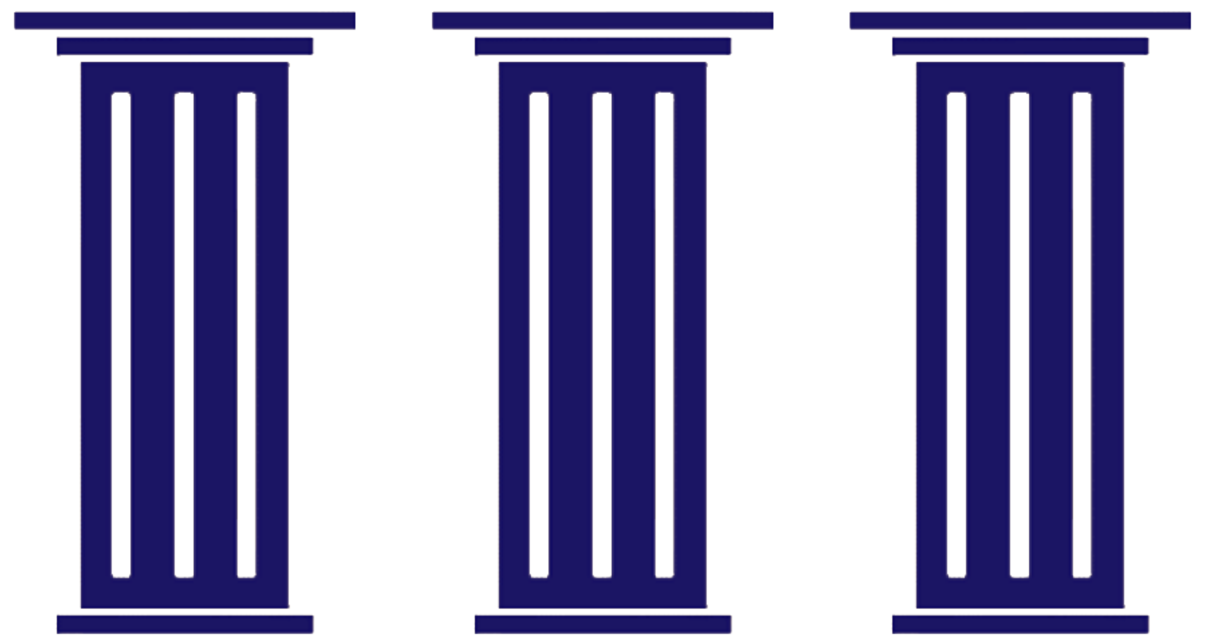
- **S**ingle-responsibility principle
- **O**pen-closed principle



SOLID

Principles

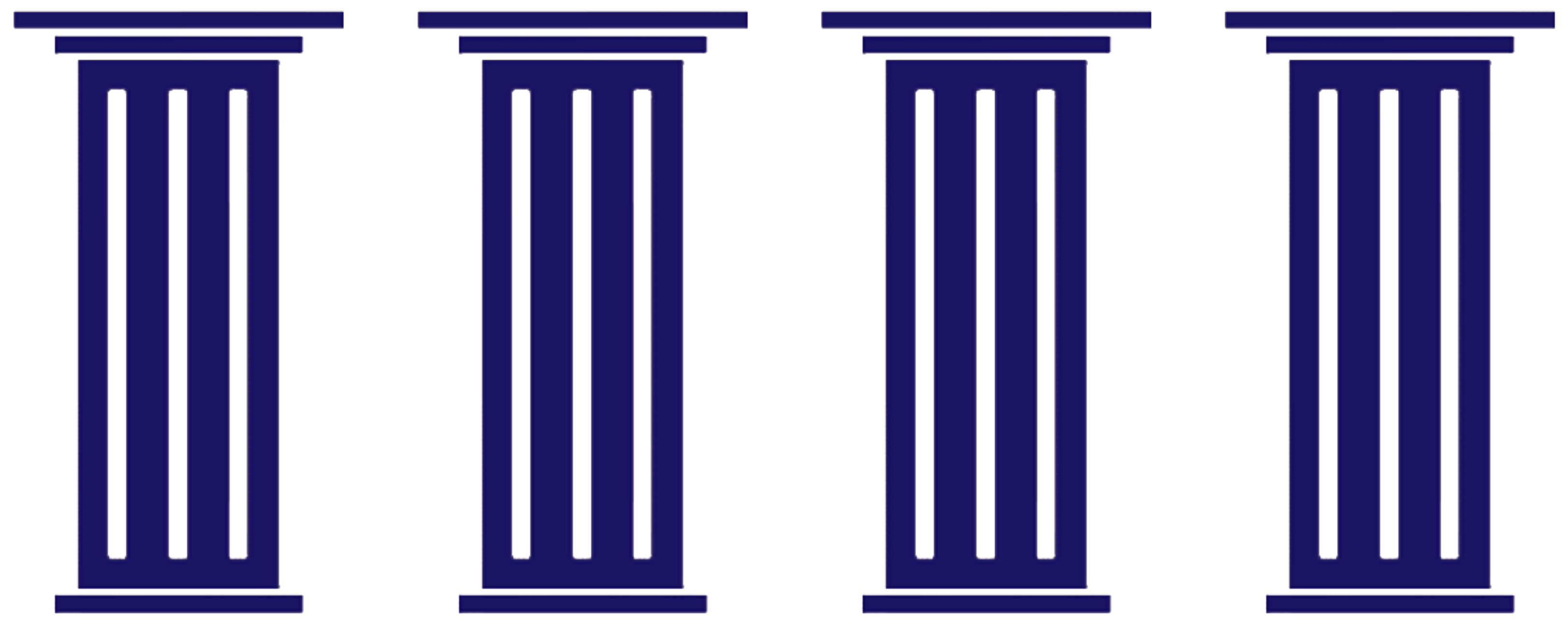
- **S**ingle-responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle



SOLID

Principles

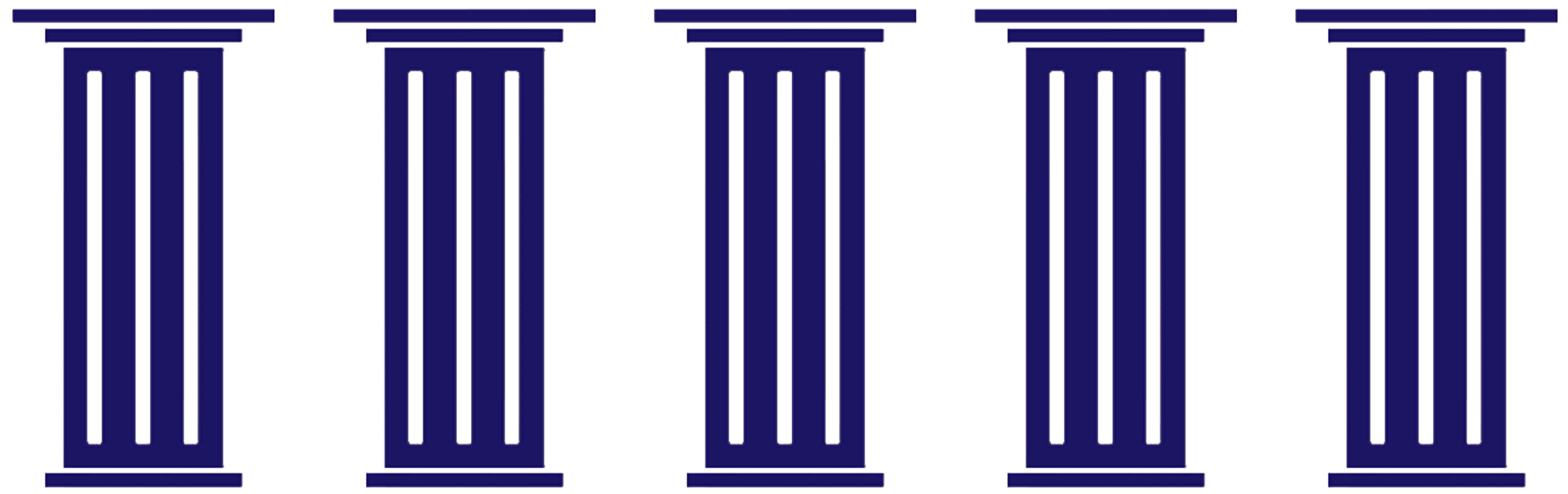
- **S**ingle-responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle



SOLID

Principles

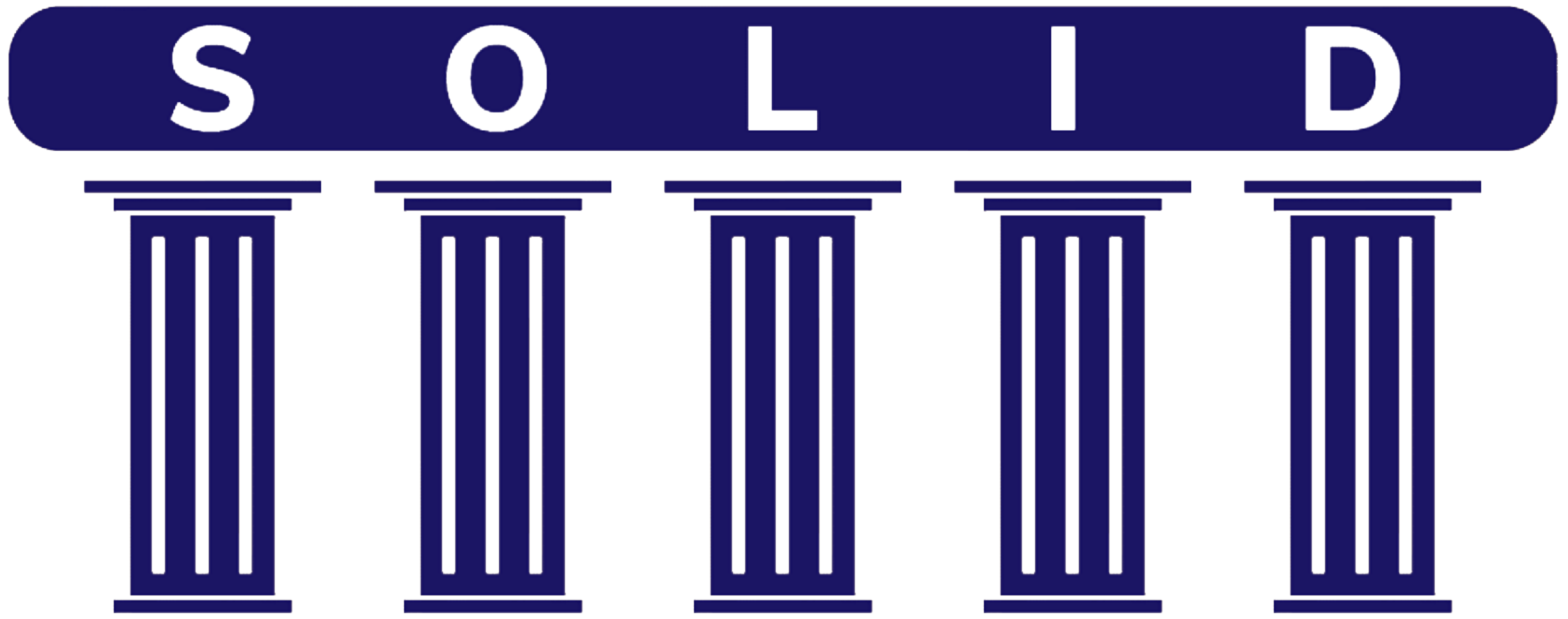
- **S**ingle-responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle



SOLID

Principles

- **S**ingle-responsibility principle
- **O**pen-closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle



Single responsibility

SOLID

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Single responsibility
SOLID

- One responsibility

WENGER
SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Single responsibility
SOLID

- One responsibility
- One reason to change

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should
WENGER

Single responsibility
SOLID

- One responsibility
- One reason to change
- Small modules
  - easy to combine
  - disjointed

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Single responsibility

SOLID

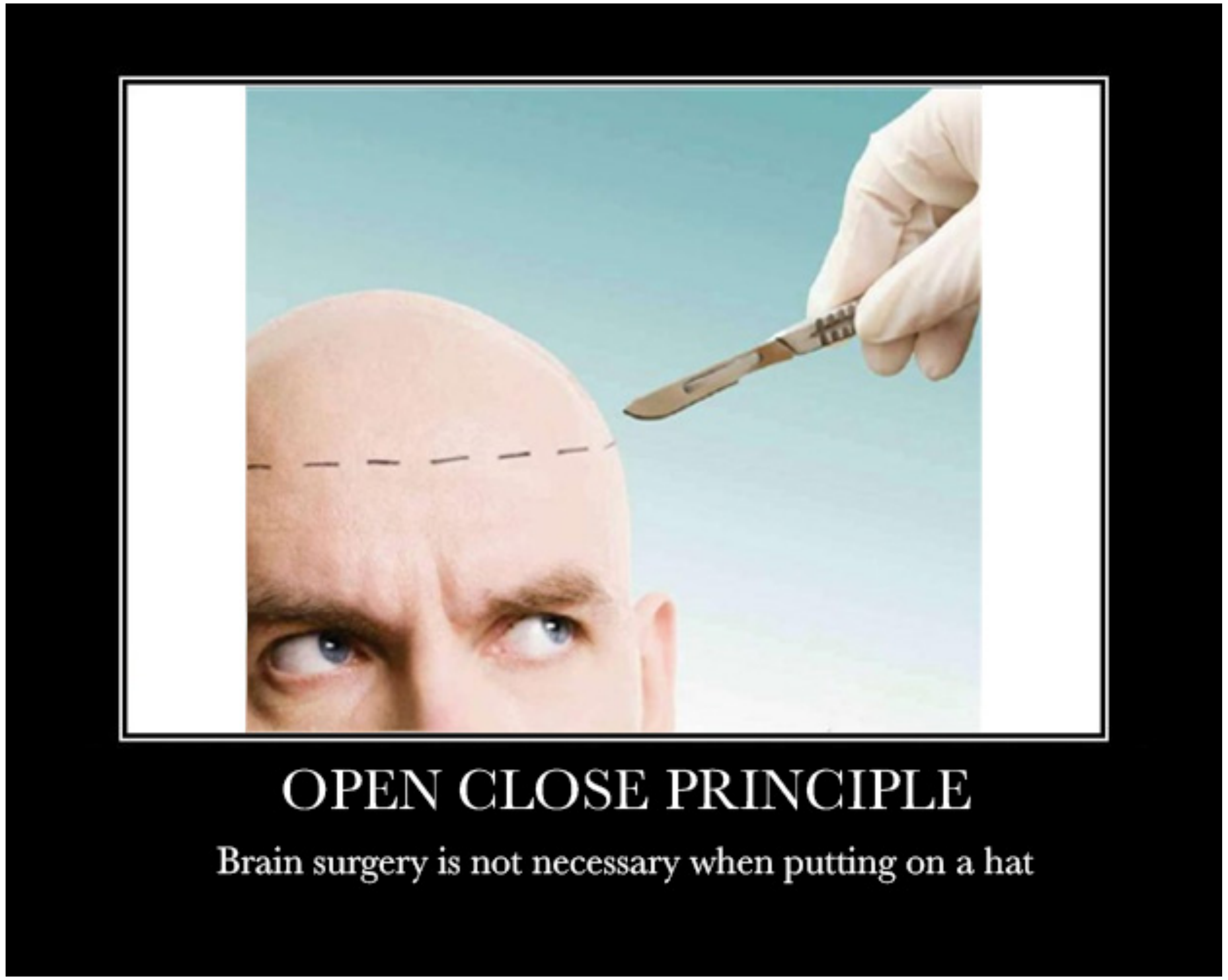
- One responsibility
- One reason to change
- Small modules
  - easy to combine
  - disjointed
- Facilitates
  - naming and reading
  - understanding and editing

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should
WENGER

Open closed

SOLID

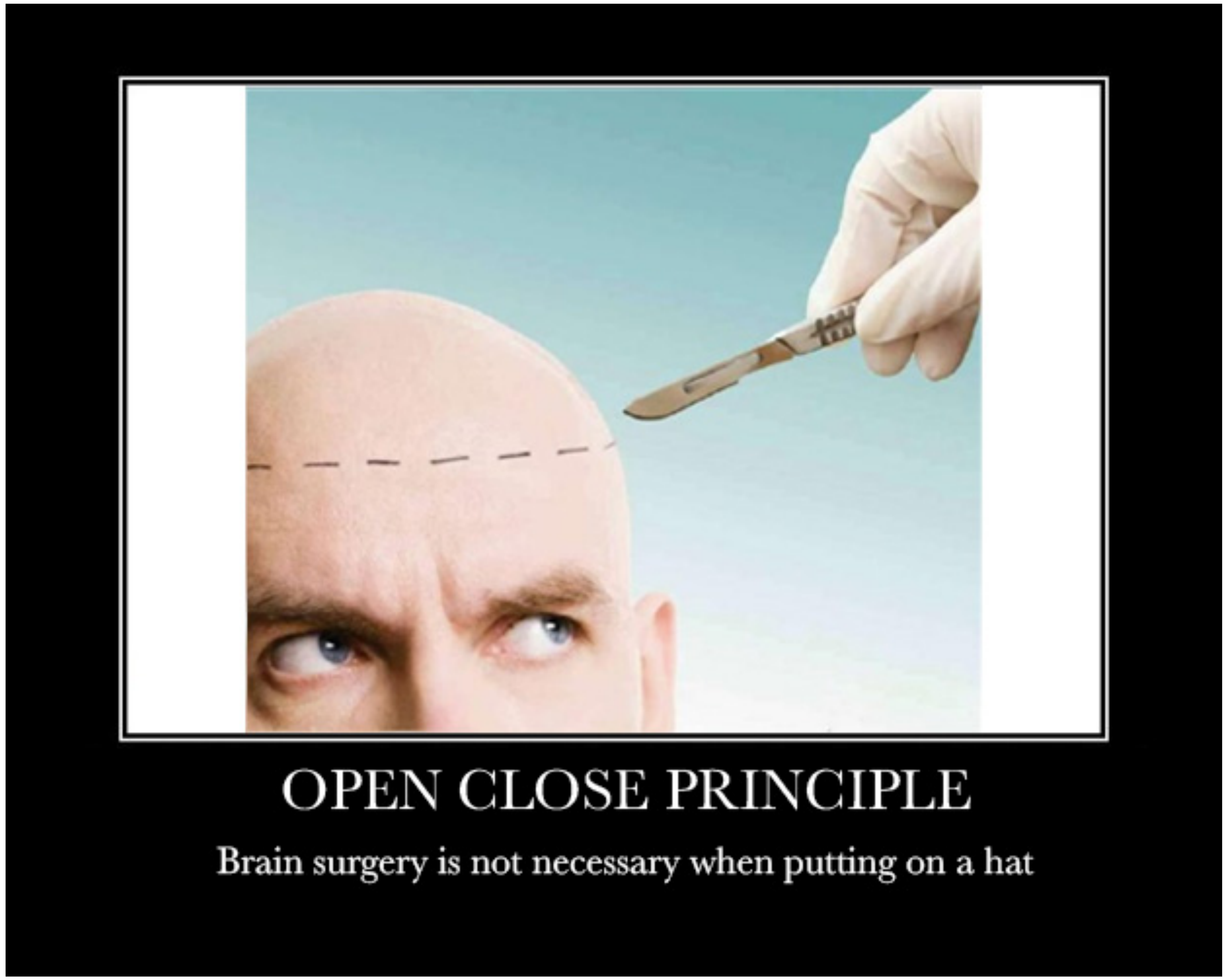
- Open for extension



Open closed

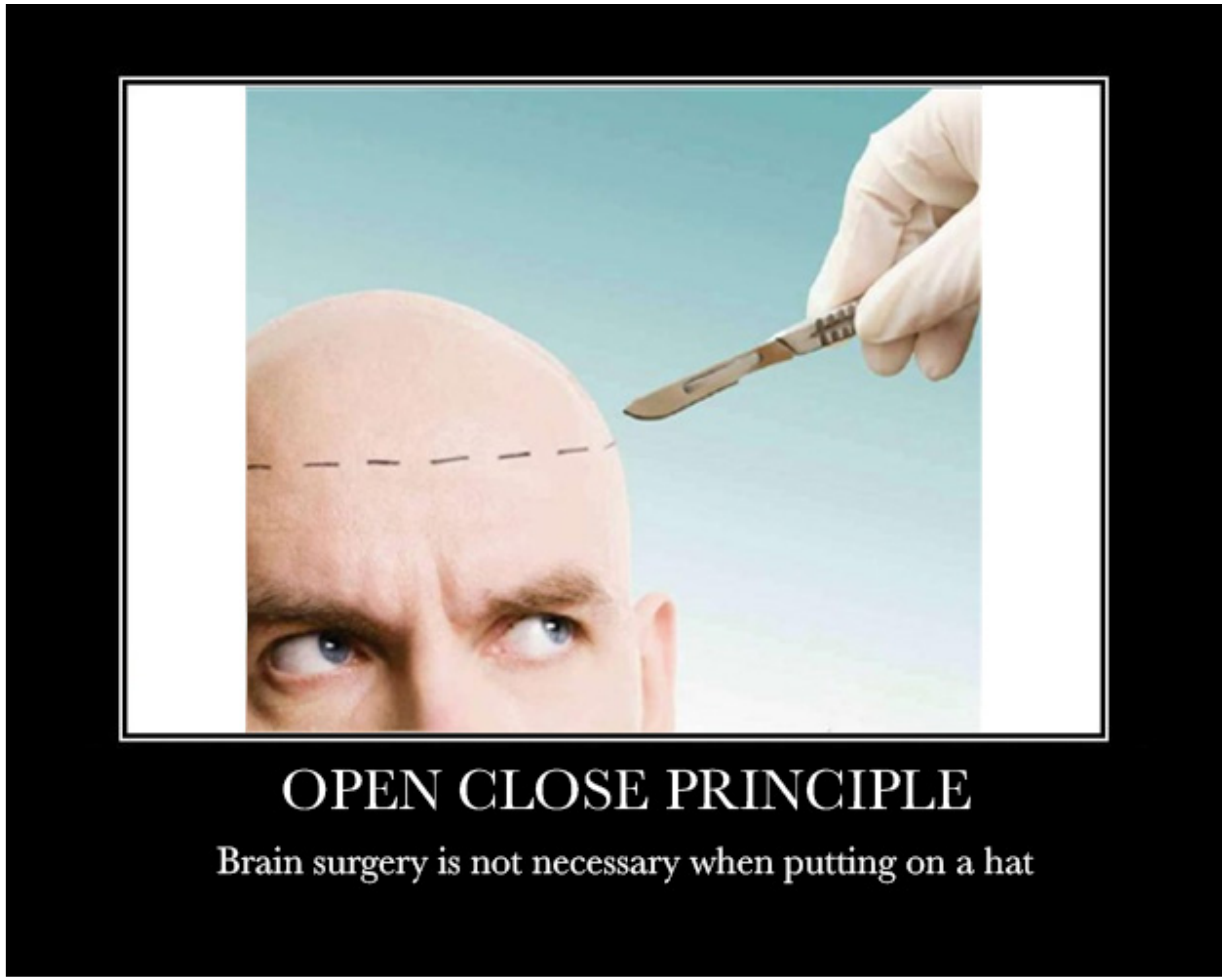
SOLID

- Open for extension
- Closed for modification



Open closed
SOLID

- Open for extension
- Closed for modification
- New feature new code



Liskov substitution

SOLID

- Derived class should be substitutable with their base class

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Liskov substitution

SOLID

- Derived class should be substitutable with their base class
- Inheritance
  - Is a kind of…

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Liskov substitution

SOLID

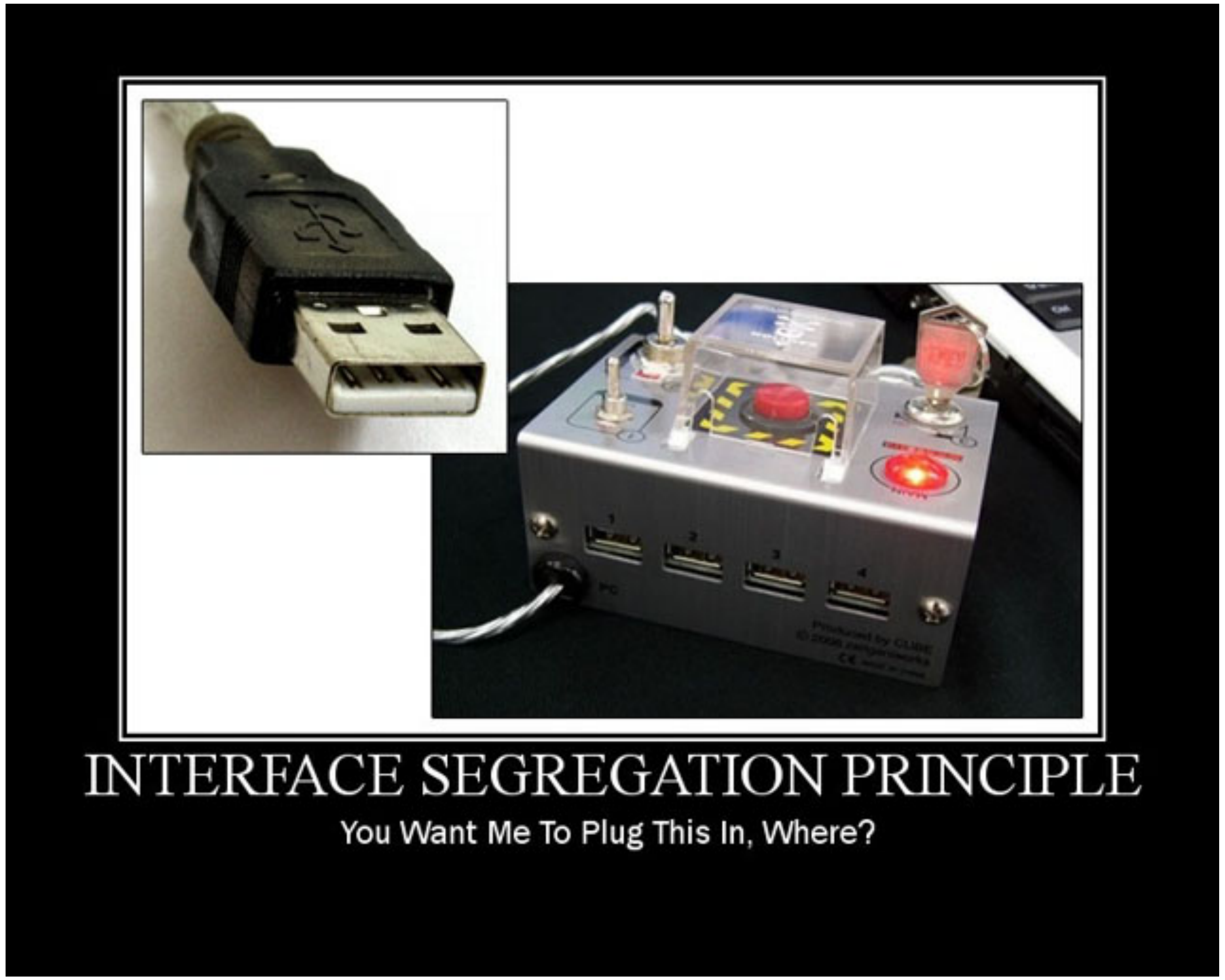
- Derived class should be substitutable with their base class
- Inheritance
  - Is a kind of…
- Composition
  - Has a…

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Interface segregation

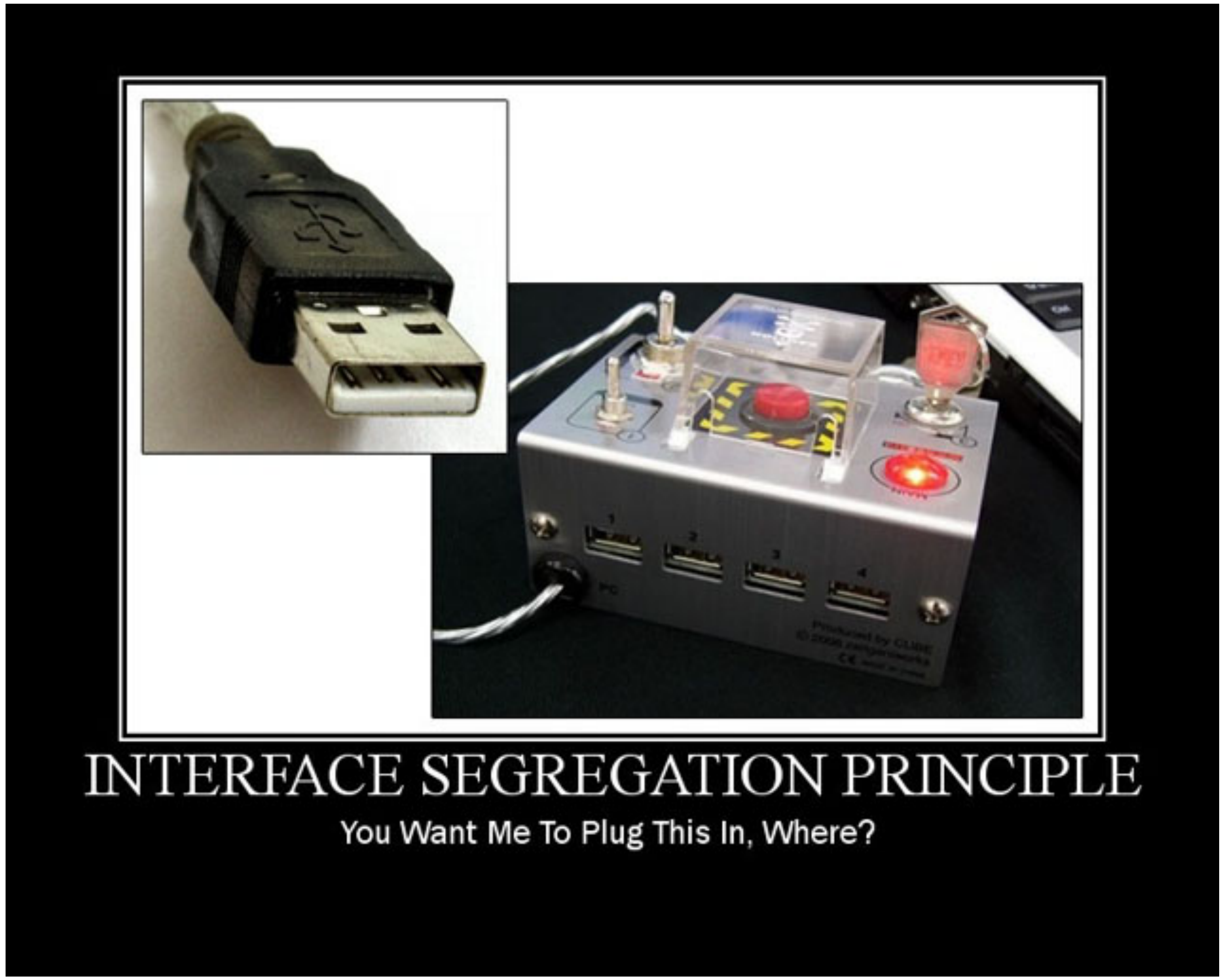
SOLID

- Clients should not be forced to depend on methods that they do not use.


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Interface segregation
SOLID

- Clients should not be forced to depend on methods that they do not use.
- Avoid generic interface

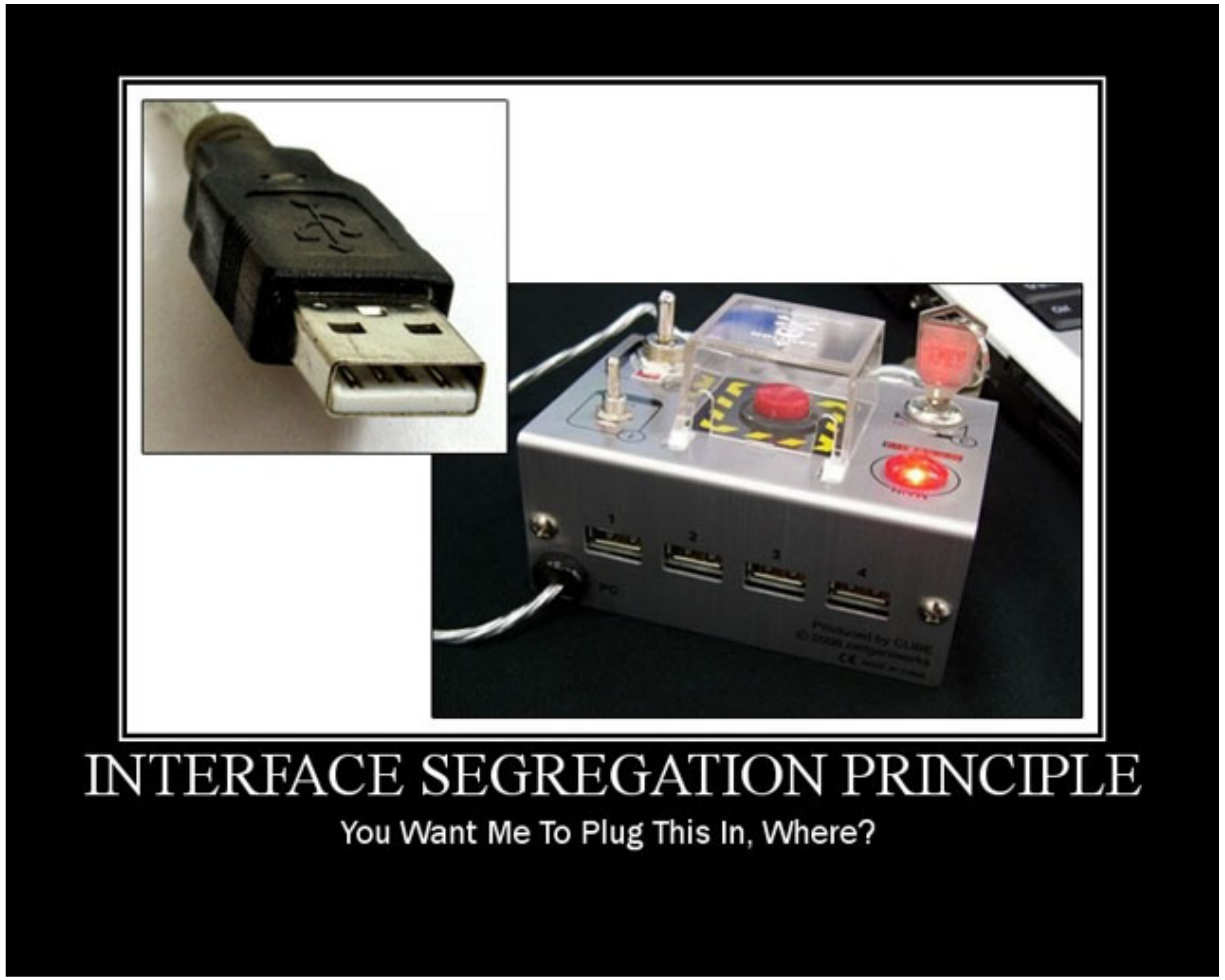

INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Interface segregation
SOLID

- Clients should not be forced to depend on methods that they do not use.

- Avoid generic interface

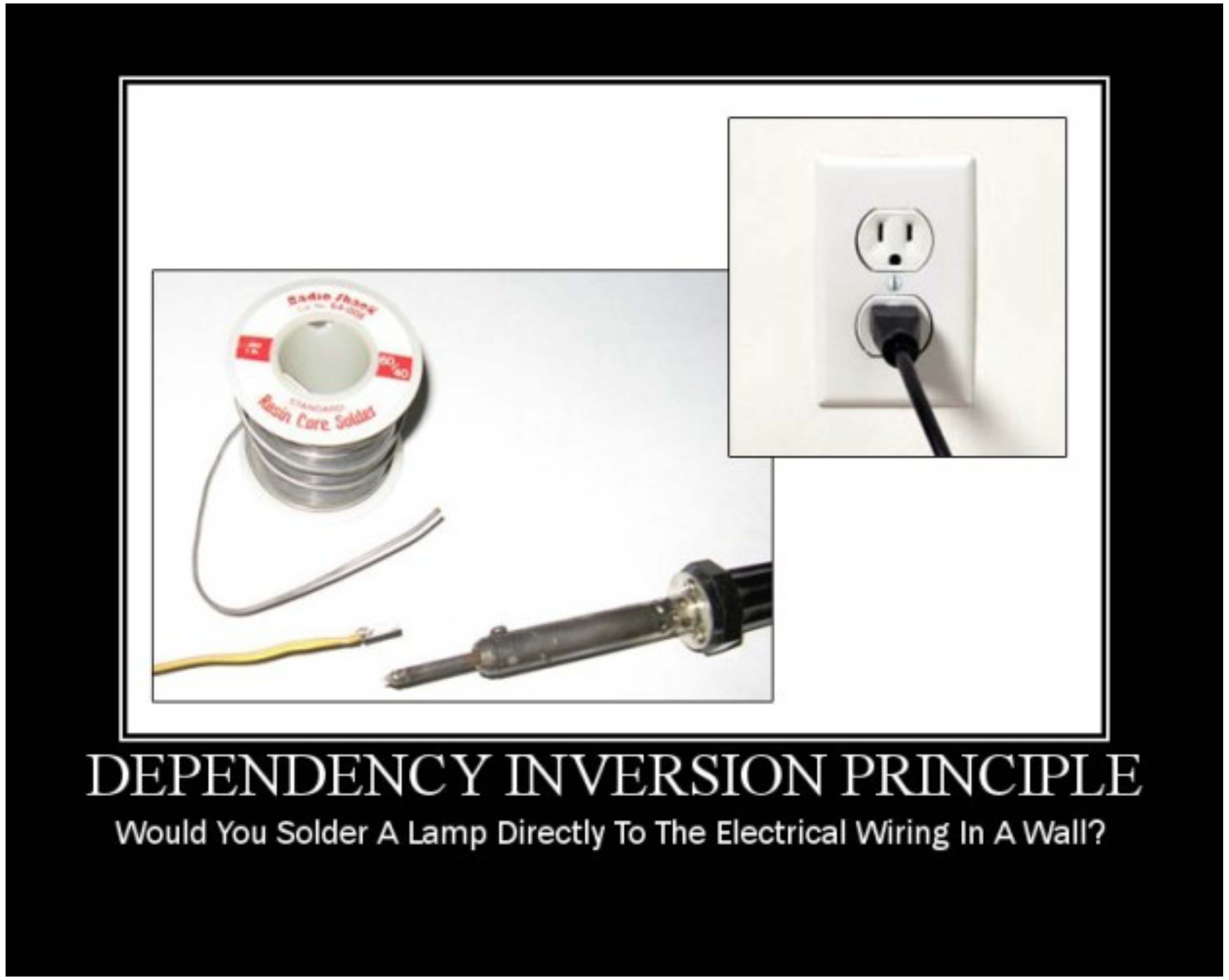
- Promote specific interface


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Dependency Inversion

SOLID

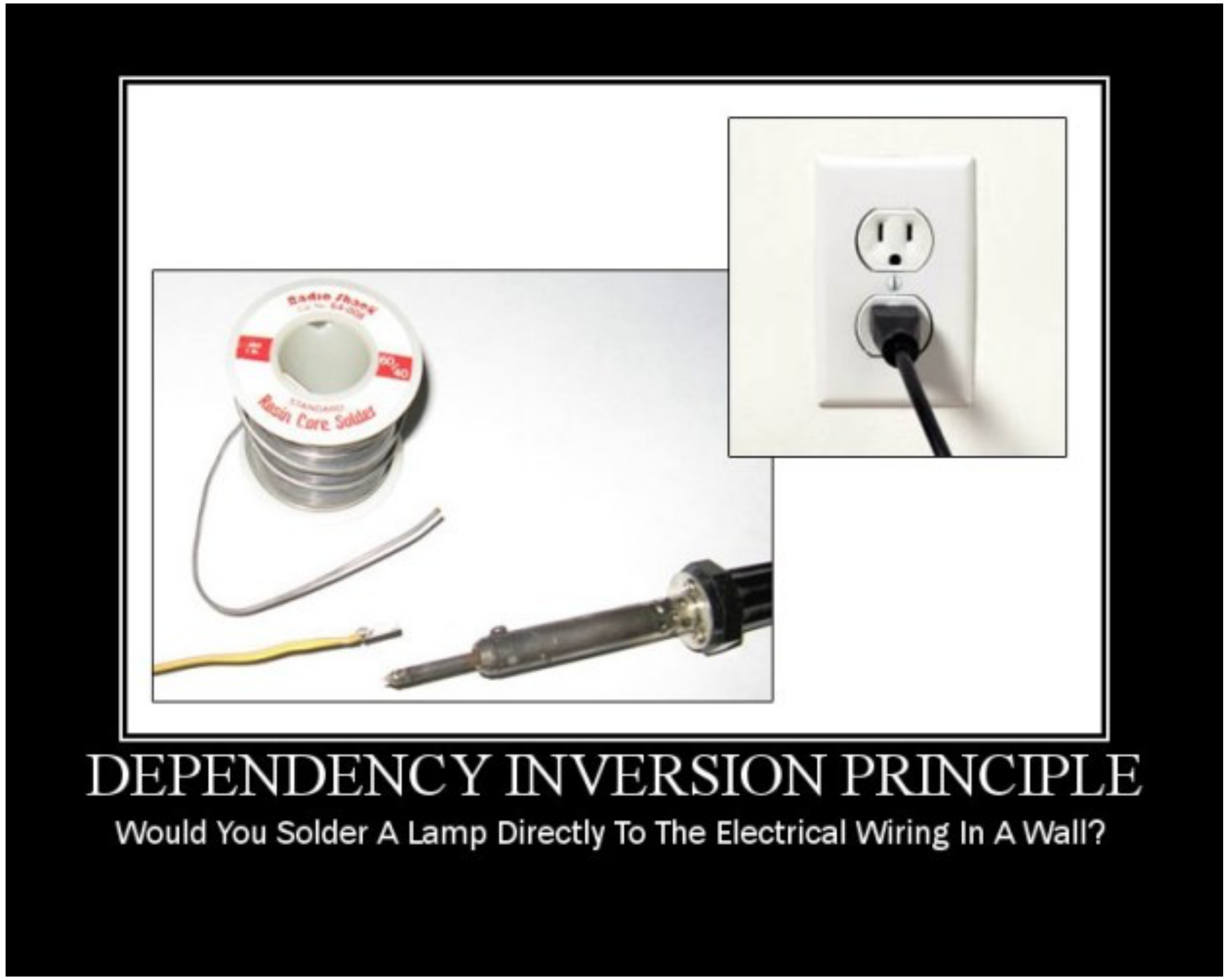
- High-level modules should not depend on low-level modules. Both should depend on the abstraction.


DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Dependency Inversion

SOLID

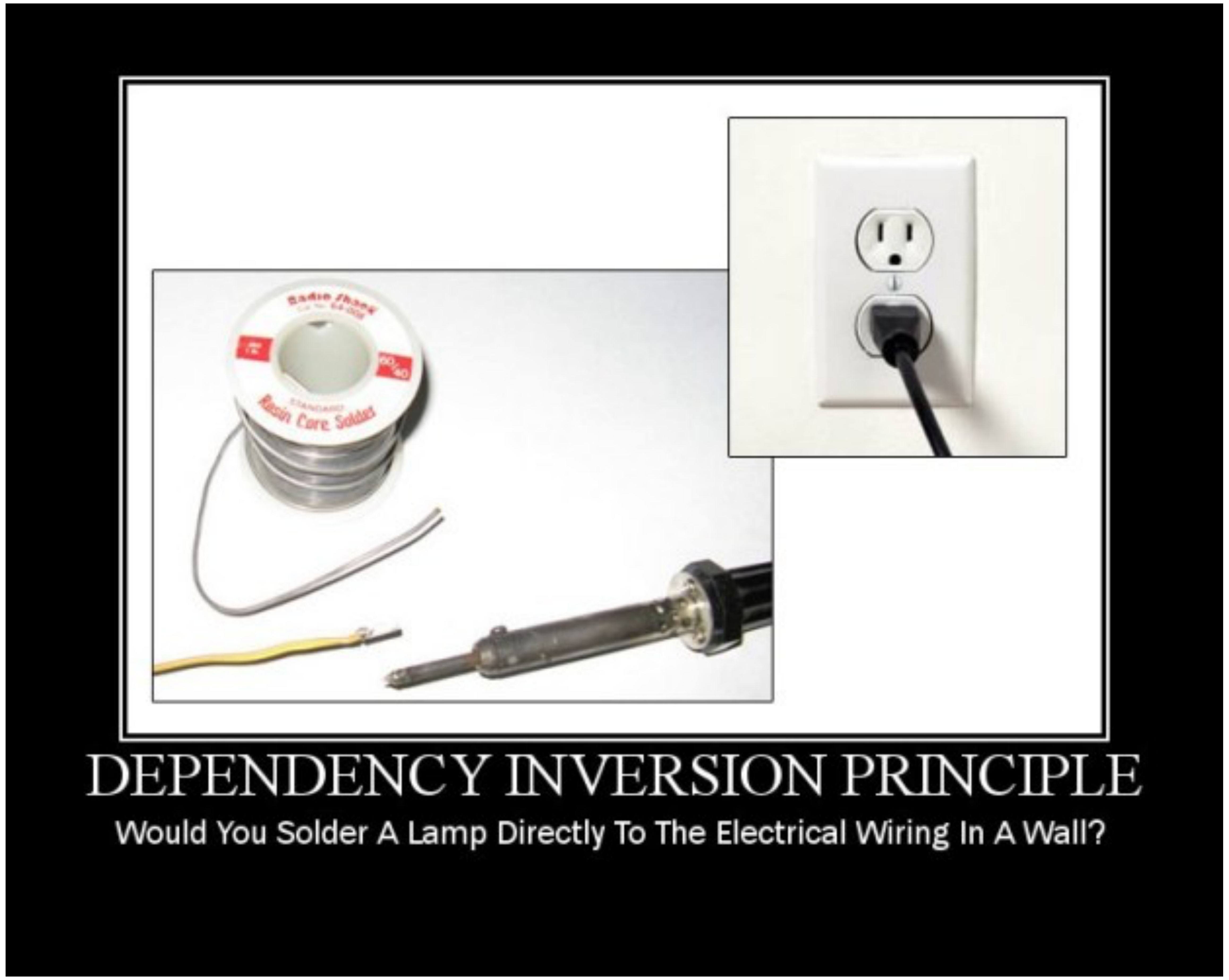
- High-level modules should not depend on low-level modules. Both should depend on the abstraction.
- Abstractions should not depend on details. Details should depend on abstractions.


DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Dependency Inversion

SOLID

- High-level modules should not depend on low-level modules. Both should depend on the abstraction.
- Abstractions should not depend on details. Details should depend on abstractions.
- Contract between elements


DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Frontend code

Old approach

- app
- common

```
app
    App.tsx
    routes.ts
    store.ts
common
    components
        Routes.tsx
        Form.tsx
        Table.tsx
features
    …
```

Frontend code

Old approach

- app
- common
- features

```
app
common
features
  users
    config
      routes.ts
      columns.ts
      form.ts
    NewUser.tsx
    ViewUser.tsx
    EditUser.tsx
    ViewUsers.tsx
  products
  orders
```

Frontend code

New possible approach

- app
- common
- features

```
app
common
features
  users
    new
      index.ts
      form.ts
      actions.ts
      NewUser.tsx
    edit
      index.ts
      form.ts
      actions.ts
      EditUser.tsx
```

Frontend code

New possible approach

- app
- common
- features

```
app
common
features
  users
    view
      index.ts
      form.ts
      actions.ts
      ViewUser.tsx
    table
      index.ts
      columns.ts
      TableUsers.tsx
```

References

- https://www.educative.io/blog/solid-principles-oop-c-sharp
- https://thedavidmasters.com/2018/10/27/solid-design-principles/
- https://imgflip.com/memegenerator

Question?

Thank you for your attention

- pietro@balestra.dev
- github.com/p1e7r0