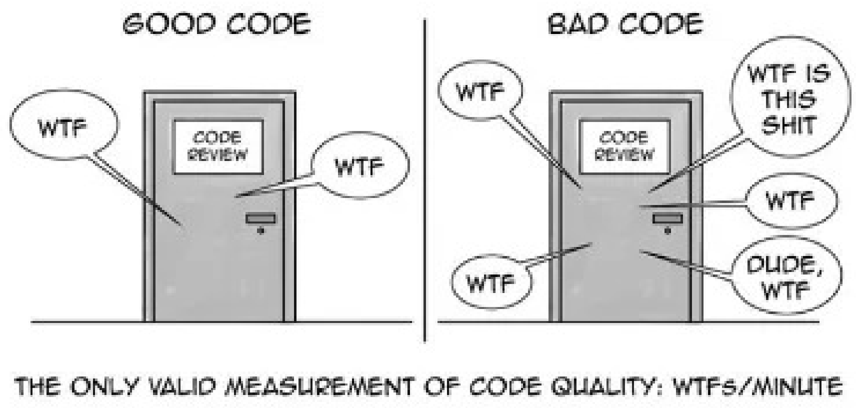
The SOLID Principles

Jan Wedekind

Thursday, Feb 22nd 2024

Motivation



Find guiding design principles to maintain software quality over time.

Software Rot

Symptoms of rotting software design:[a]

- **Rigidity**: software difficult (a lot of work) to change
- **Fragility**: changes easily break the software
- **Immobility**: it is easier to rewrite than reuse parts
- **Viscosity**: design preserving methods are harder to employ than hacks

[a] Robert C. Martin: Design Principles and Design Patterns

Aims

In contrast we want to achieve the following:[a]

- Keep software application **flexible**
- Keep software application **robust**
- Keep software application **reusable**
- Keep software application **developable**

[a] Robert C. Martin: Design Principles and Design Patterns

SOLID Authors

Robert C. Martin

- Author of Clean Code, Functional Design, and more books
- Author of Design Principles and Design Patterns paper based on his experience and on work by Bertrand Meyer, Barbara Liskov, and Erich Gamma et al.
- `http://cleancoder.com/`

Michael Feathers

- Author of Working Effectively With Legacy Code
- Summarized Robert C. Martin's paper using the SOLID acronym
- `https://www.r7krecon.com/`

The SOLID Principles

1. **S**ingle responsibility
2. **O**pen-closed
3. **L**iskov substitution
4. **I**nterface segregation
5. **D**ependency inversion

Single Responsibility - Before

```
def adults_to_html(people):
    result = "<ul>\n"
    for person in people:
        if person.age >= 18:
            result += "  <li>" + person.name + "</li>\n"
    result += "</ul>"
    return result
# ...
page = adults_to_html(people)
```

Single Responsibility - After

```
def select_adults(people):
    return [person for person in people if person.age >= 18]

def people_to_html(people):
    result = "<ul>\n"
    for person in people:
        result += "  <li>" + person.name + "</li>\n"
    result += "</ul>"
    return result
# ...
page = people_to_html(select_adults(people))
```

Open-Closed - Before

```python
def total_area(shapes):
  result = 0
  for shape in shapes:
    match type(shape):
      case Rectangle:
        result += shape.width * shape.height
      case Sphere:
        result += math.pi * shape.radius ** 2
      case _:
        raise f"Unsupported shape {shape}"
  return result
```

Open-Closed - After

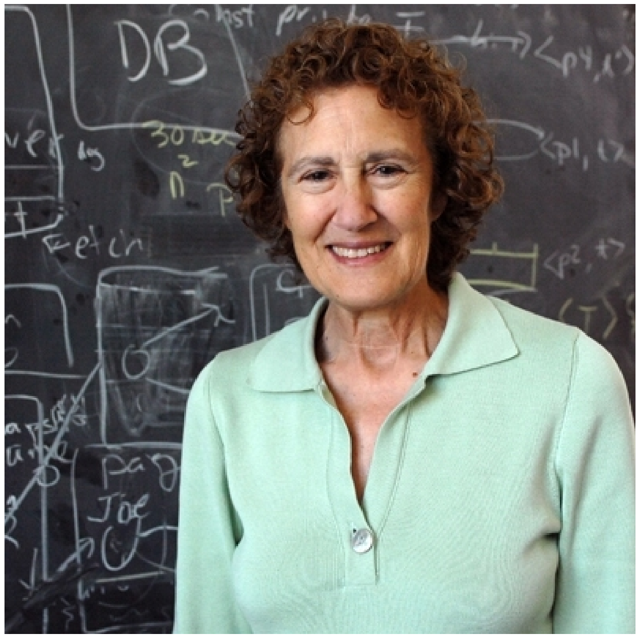
```python
class Rectangle:
  def area(self):
    return self.width * self.height

class Circle:
  def area(self):
    return math.pi * self.radius ** 2

def total_area(shapes):
  result = 0
  for shape in shapes:
    result += shape.area()
  return result
```

Liskov-Substitution - Before

```python
class Rectangle:
  def __init__(self, width, height):
    self.width = width
    self.height = height
  def set_width(self, width):
    self.width = width
  def set_height(self, height):
    self.height = height

class Square(Rectangle):
  def __init__(self, side):
    super().__init__(side, side)
  def set_width(self, width):
    super().set_width(width)
    super().set_height(width)
  def set_height(self, height):
    self.set_width(height)
```

Liskov-Substitution - After

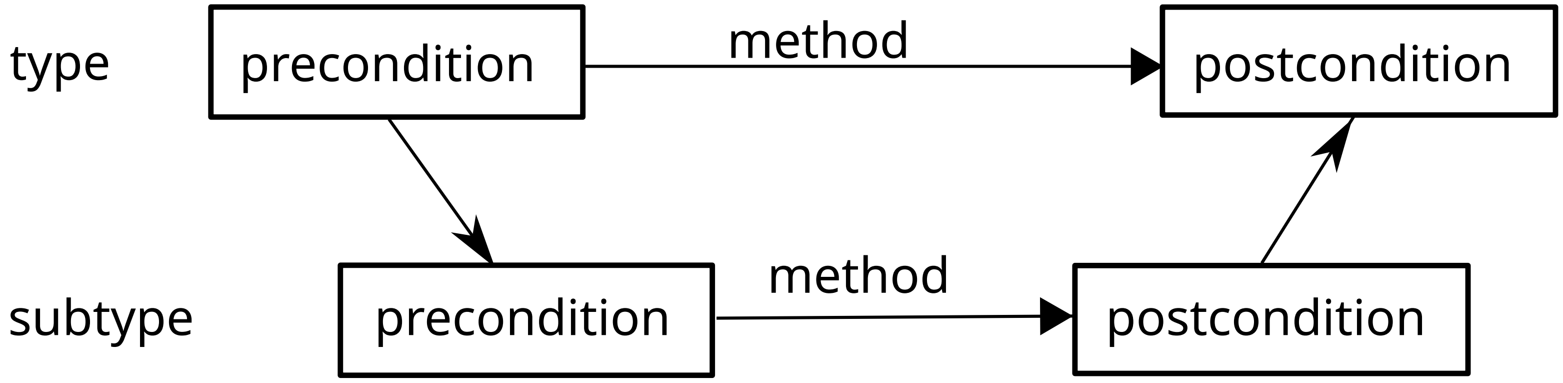
```
class Shape:
  pass

class Rectangle(Shape):
  def __init__(self, width, height):
    self.width = width
    self.height = height
  def set_width(self, width):
    self.width = width
  def set_height(self, height):
    self.height = height

class Square(Shape):
  def __init__(self, side):
    self.side = side
  def set_side(self, side):
    self.side = side
```



Barbara Liskov

Liskov-Substitution - Contracts

"The Liskov Substitution Principle states, among other constraints, that a subtype is not substitutable for its super type if it strengthens its operations' preconditions, or weakens its operations' postconditions"[a]



[a] Baniassad: Making the Liskov Substitution Principle Happy and Sad

Interface Segregation - Before

```
class AccountHolder:
  def __init__(self, name, age, balance):
    self.name = name
    self.age = age
    self.balance = balance
  def is_adult(self):
    return self.adult >= 18
  def deposit(self, amount):
    self.balance += amount
  def withdraw(self, amount):
    self.balance -= amount
```

Interface Segregation - After

```
class Person:
  def __init__(self, name, age):
    self.name, self.age = name, age
  def is_adult(self):
    return self.adult >= 18

class Account:
  def __init__(self, balance):
    self.balance = balance
  def deposit(self, amount):
    self.balance += amount
  def withdraw(self, amount):
    self.balance -= amount

class AccountHolder(Person):
  def __init__(self, name, age, account):
    super().__init__(name, age)
    self.account = account
```

Dependency Inversion - Before

```python
def get_names(connection):
    cursor = connection.cursor()
    cursor.execute('SELECT name FROM member_table')
    rows = cursor.fetchall()
    names = [row[0] for row in rows]
    return names

connection = sqlite3.connect('example.db')
names_list = get_names(connection)
connection.close()
print(names_list)
```

Dependency Inversion - After

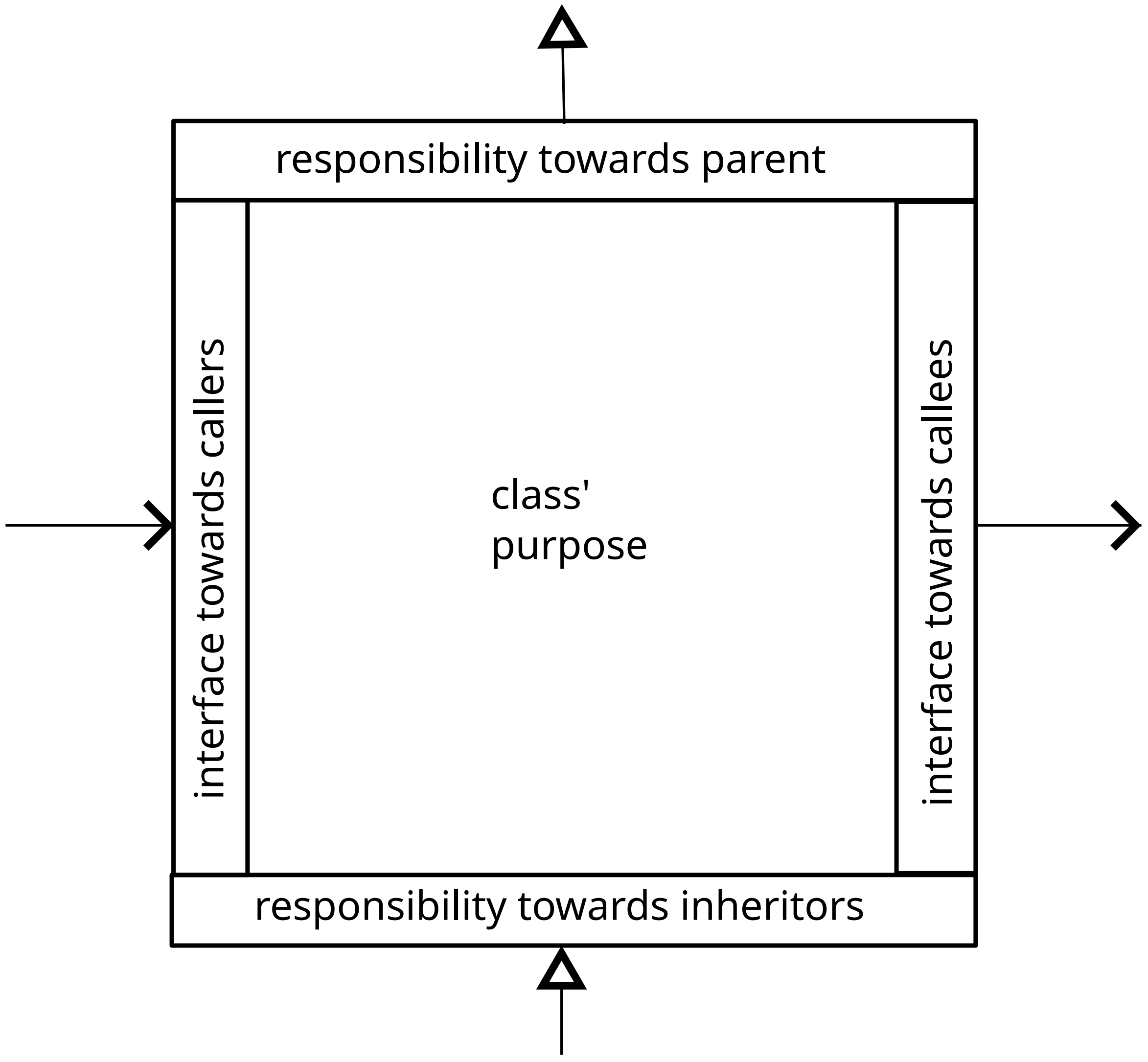
```
class Database(abc.ABC):
  @abc.abstractmethod
  def sql(self, query):
    pass

class SQLiteDatabase(Database):
  def __init__(self, db_file_name):
    self.connection = sqlite3.connect(db_file_name)
  def __del__(self):
    self.connection.close()
  def sql(self, query):
    cursor = self.connection.cursor()
    cursor.execute(query)
    return cursor.fetchall()

def get_names(database):
  rows = database.sql('SELECT name FROM member_table')
  return [row[0] for row in rows]
```

Dependency Inversion - After

```python
database = SQLiteDatabase('example.db')
names_list = get_names(database)
print(names_list)
```

Aspects of a Class

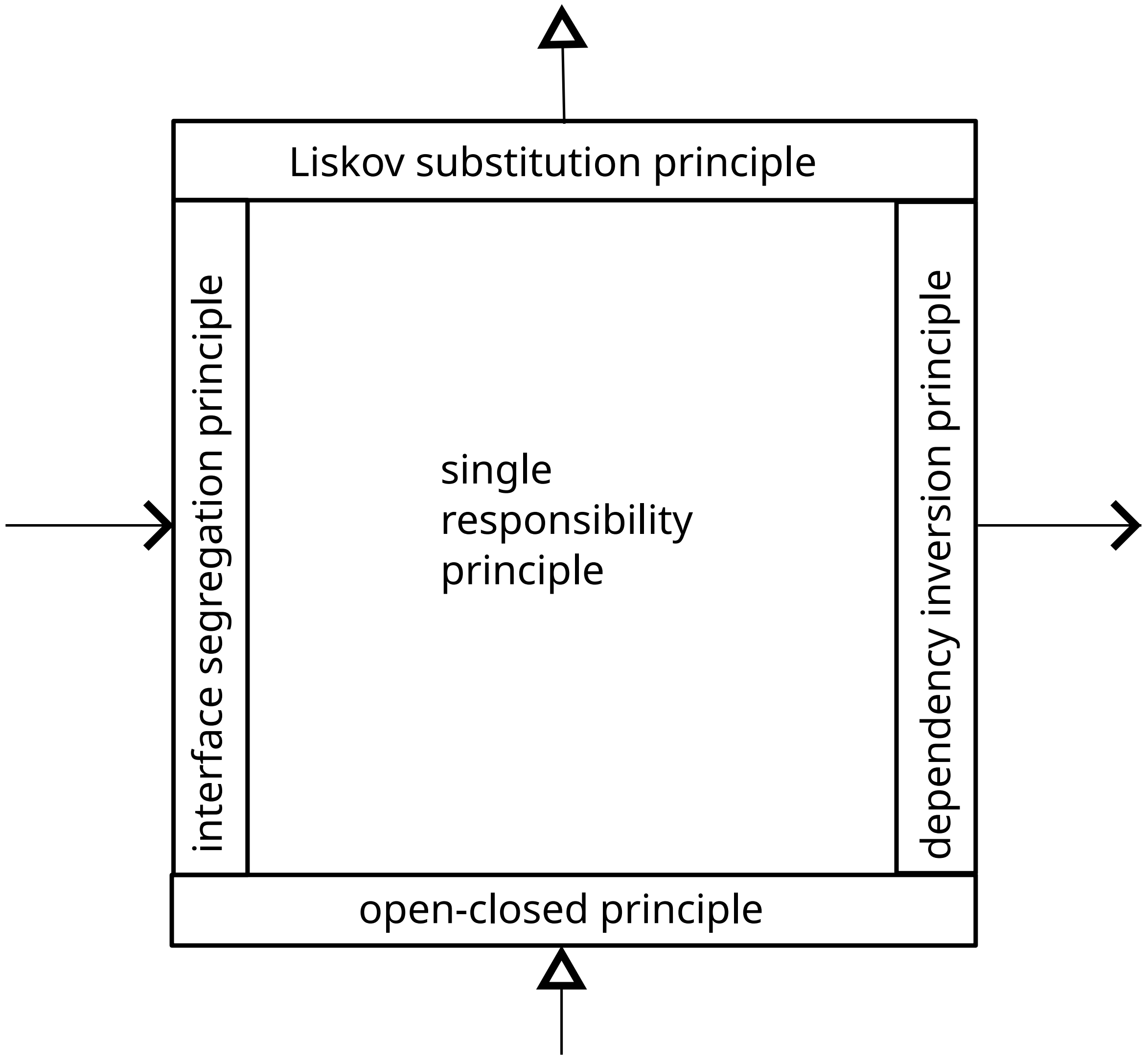
The 5 aspects of the class are:[a]


responsibility towards parent
interface towards callers
class' purpose
interface towards callees
responsibility towards inheritors

[a] Mike Lindner: The Five Principles For SOLID Software Design

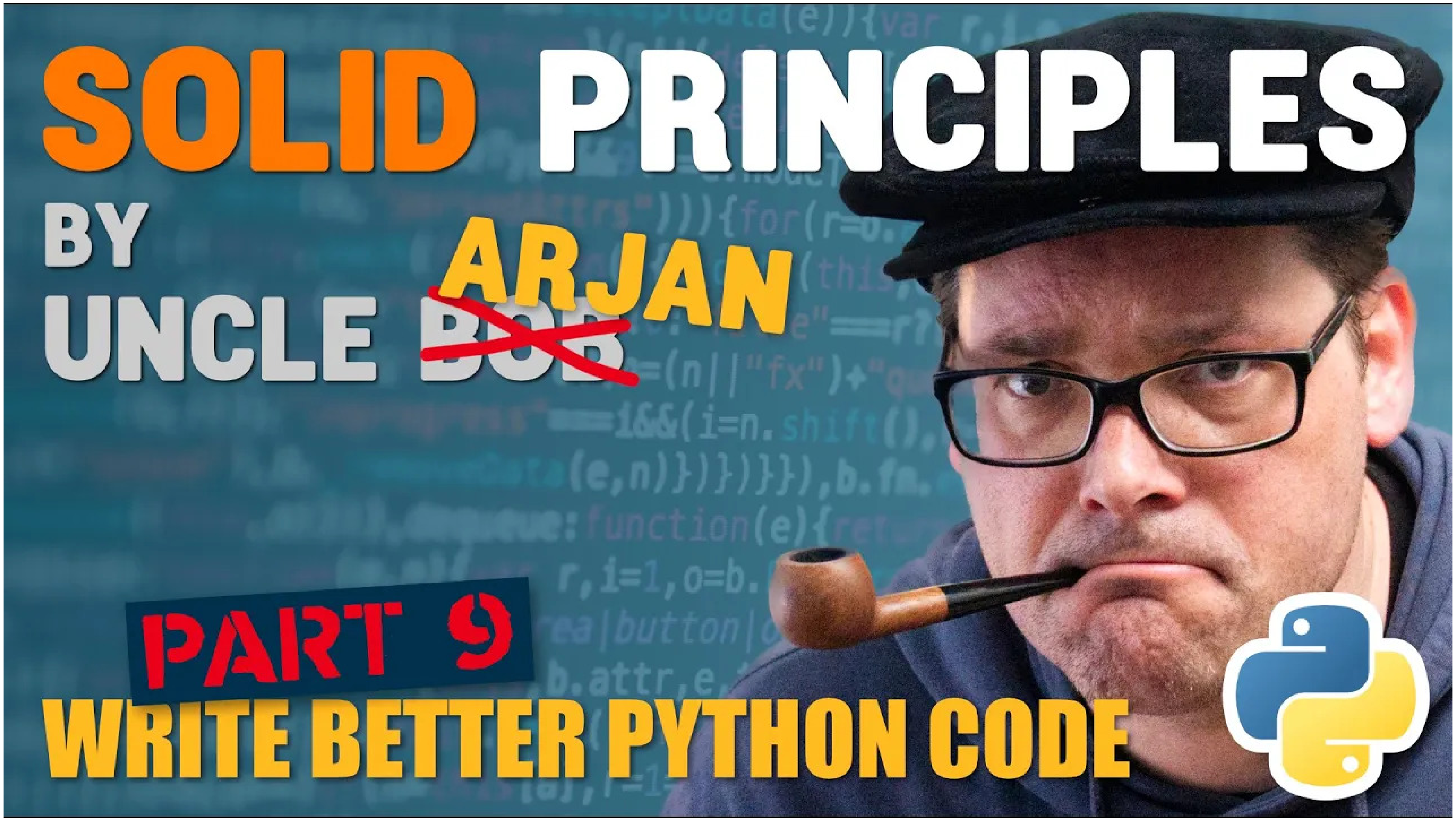
The 5 Principles

The 5 corresponding principles are:[a]


Liskov substitution principle
interface segregation principle
single responsibility principle
dependency inversion principle
open-closed principle

[a] Mike Lindner: The Five Principles For SOLID Software Design

Arjan Egges: Uncle Bob's SOLID Principles Made Easy



19 minutes video

Jim Weirich: The Building Blocks of Modularity

33 minutes video