Clean Architectures in Python

A tale of durability, utility, and beauty

LEONARDO GIORDANI

SOFTWARE DEVELOPER AND BLOGGER

WWW.THEDIGITALCATONLINE.COM

@TW_LGIORDANI - @THEDIGICAT

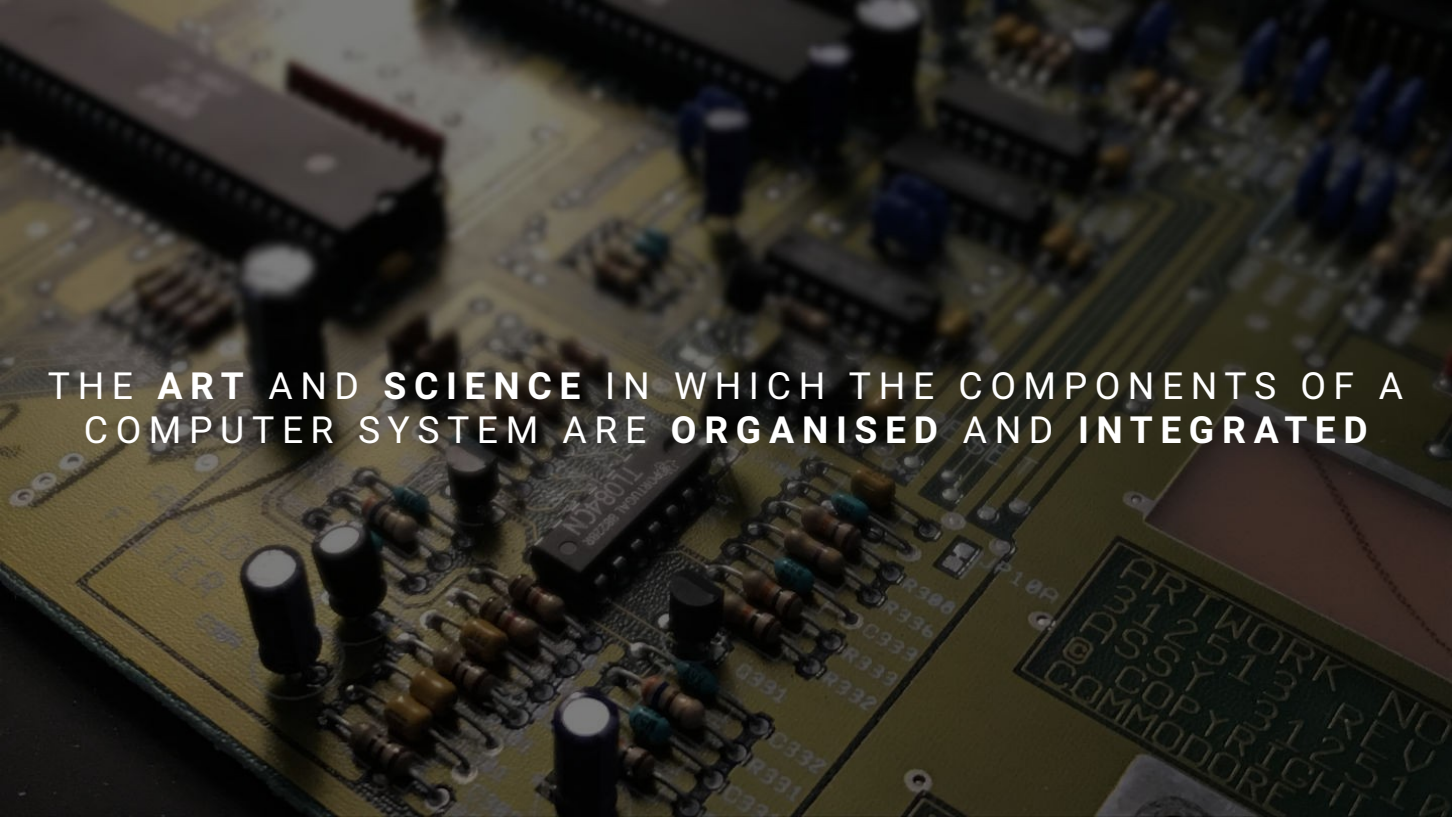

WHAT IS THE DEFINITION
OF **ARCHITECTURE**?

FIRMITAS, UTILITAS, VENUSTAS

Vitruvius, *De architectura*

DURABILITY, UTILITY, BEAUTY

Vitruvius, *De architectura*


THE **ART** AND **SCIENCE** IN WHICH THE COMPONENTS OF A COMPUTER SYSTEM ARE **ORGANISED** AND **INTEGRATED**

DO WE **NEED** ARCHITECTURE?

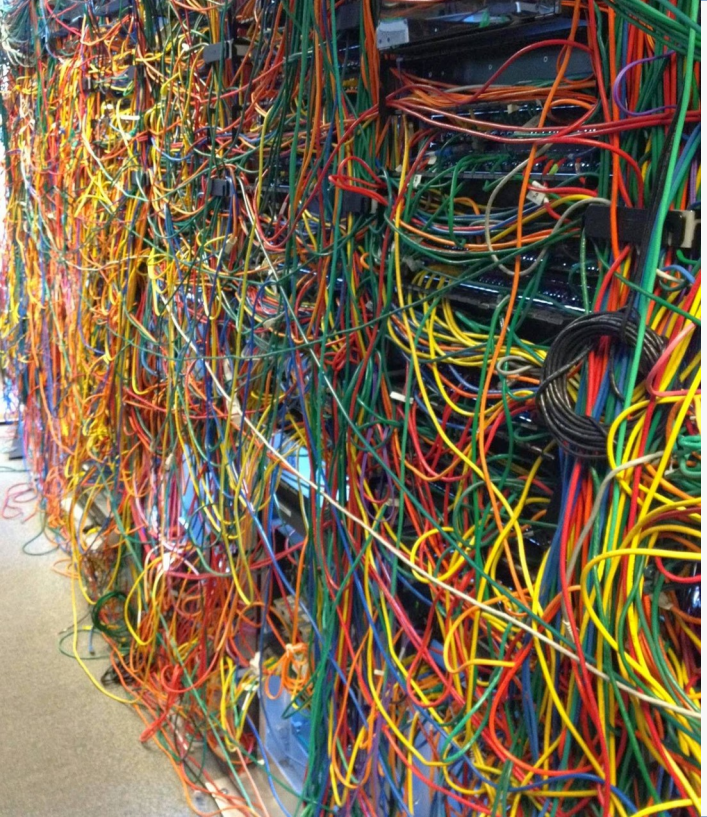
Ivar Jacobson (1992)
Object Oriented Software Engineering: A Use-Case Driven Approach

E. Gamma, R. Helm, R. Johnson, J. Vlissides (1994)
Design Patterns

Robert Martin (2000)
Design Principles and Design Patterns

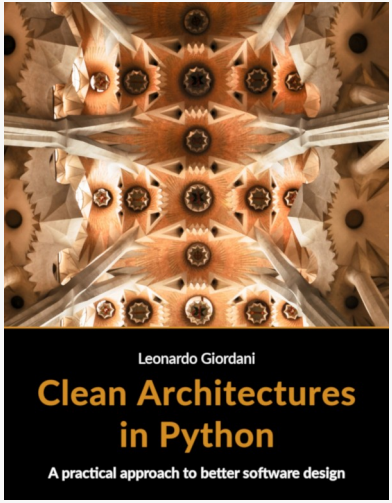
Eric Evans (2003)
Domain-Driven Design: Tackling Complexity in the Heart of Software

H. Hohpe, B. Woolf (2003)
Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions



What is the meaning of **clean**?

You know
where things are,
why components are there,
what something is.



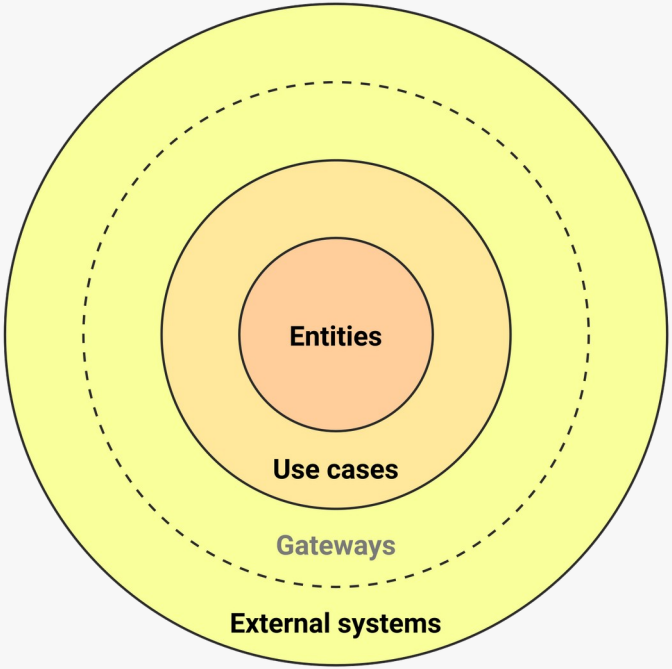
Leonardo Giordani

Clean Architectures in Python

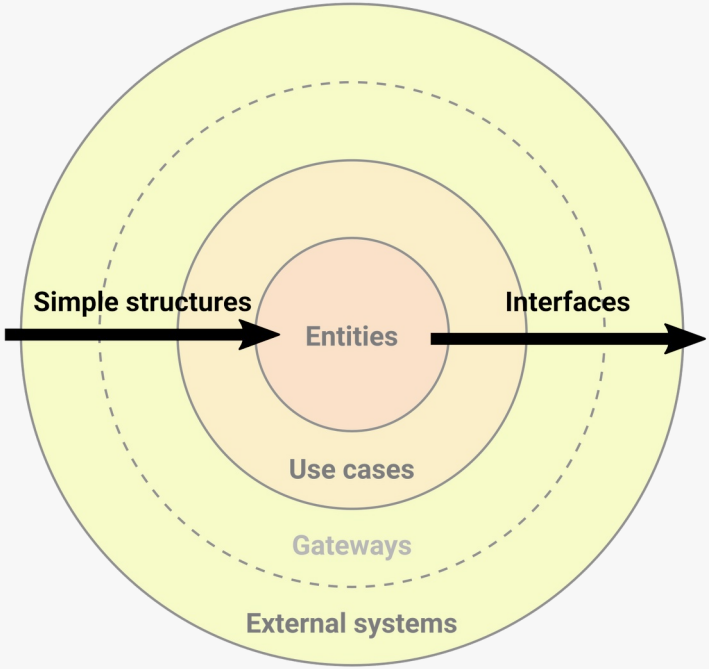
A practical approach to better software design

bit.ly/getpycabook

The Clean Architecture
A **layered** approach for a more civilized age

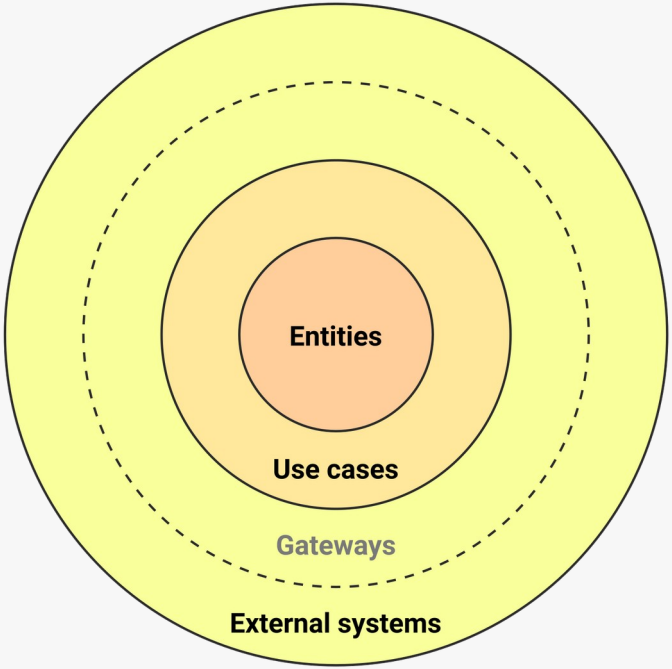


The golden rule
Talk inward with **simple structures**,
talk outwards through **interfaces**.



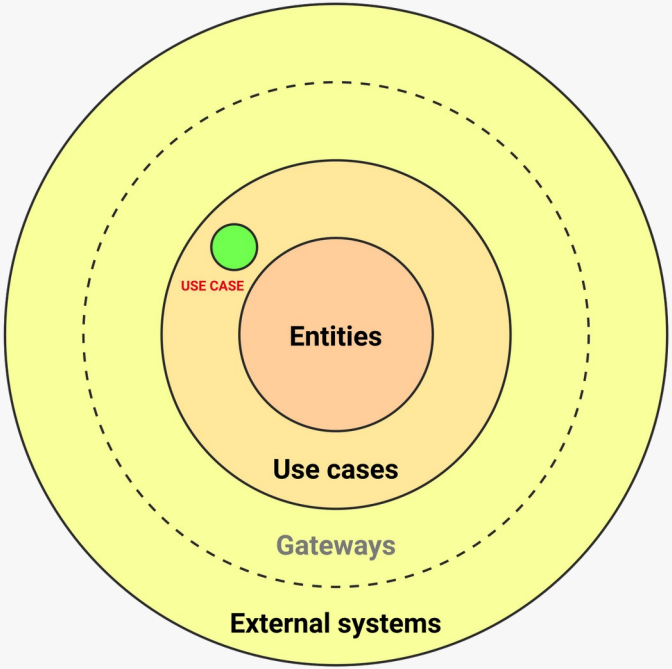
Entities: simple models

```
class Item:
    def __init__(self, code, price):
        self.code = code
        self.price = price
```



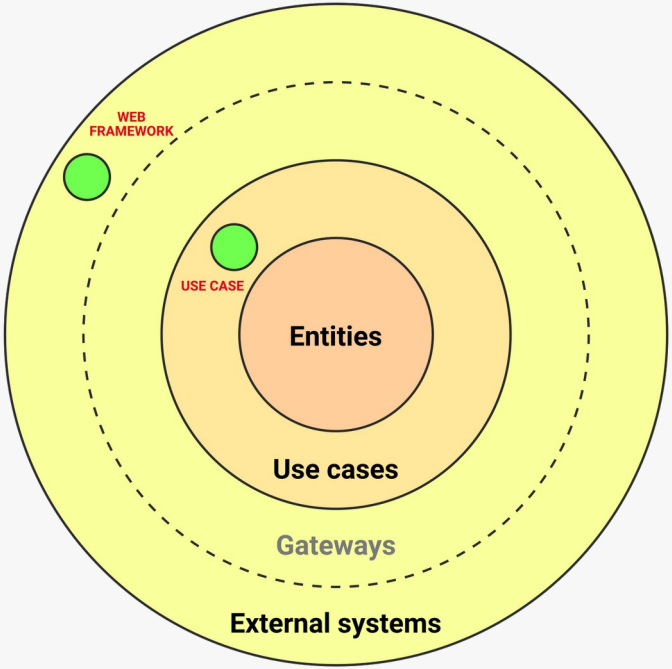
Use case: retrieve a list of items

```
use_case = uc.ItemsListUseCase()
use_case.execute()
```



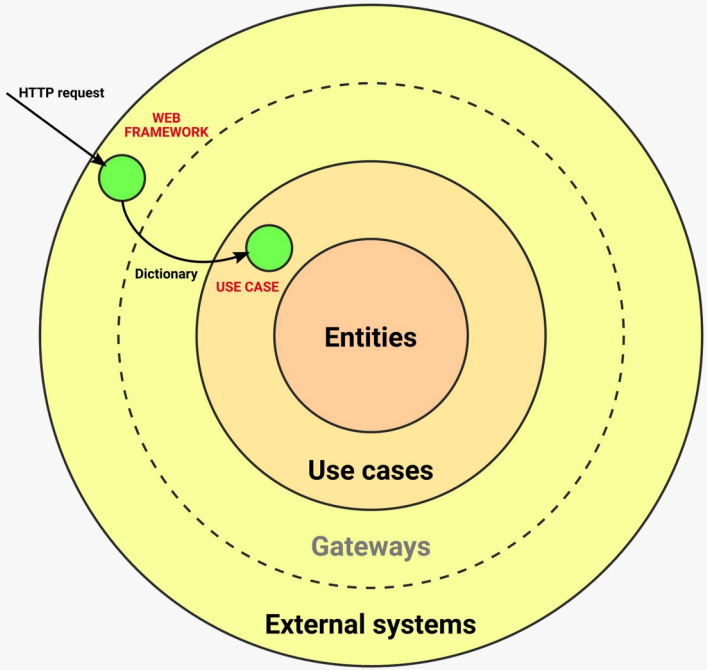
We want to build a **web application**

```
@blueprint.route('/items', methods=['GET'])
def items():
    pass
```



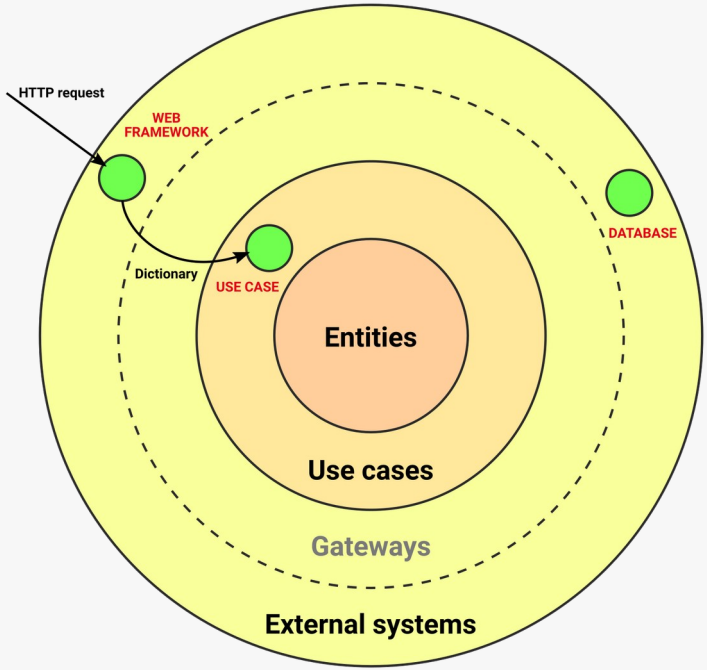
Incoming HTTP requests become a **call** and **simple structures**

```
@blueprint.route('/items', methods=['GET'])
def items():
    use_case = uc.ItemsListUseCase()
    use_case.execute(request.args)
```



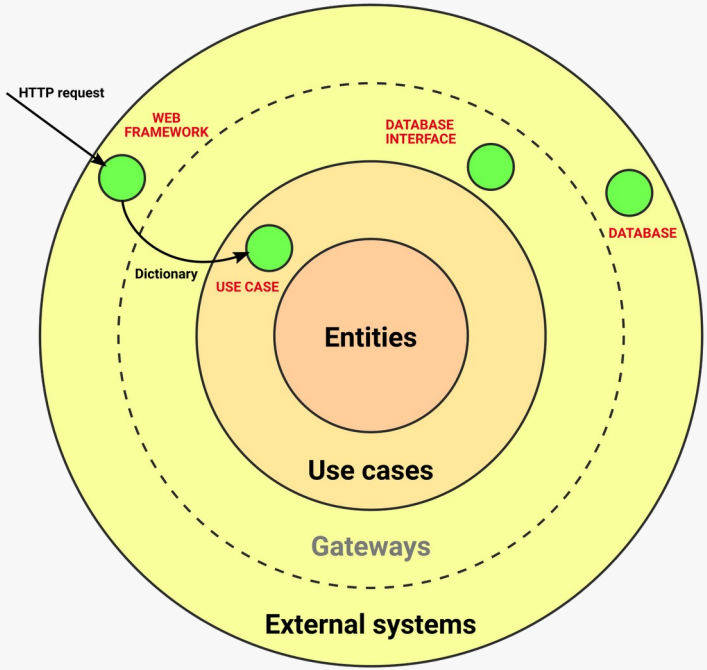
The use case extracts data from a **repository**, which can be any source of data

```
@blueprint.route('/items', methods=['GET'])
def items():
    use_case = uc.ItemsListUseCase()
    use_case.execute(request.args)
```



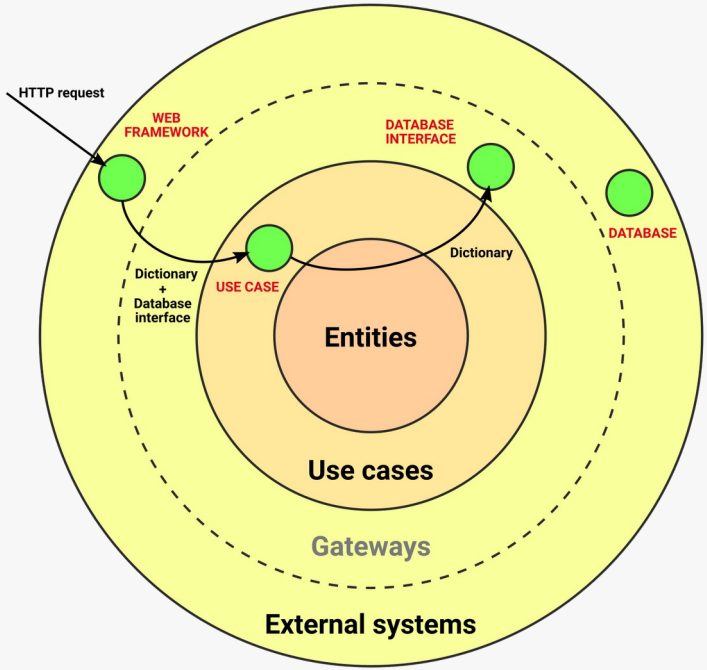
And the repository can be accessed through an **interface**

```
@blueprint.route('/items', methods=['GET'])
def items():
    use_case = uc.ItemsListUseCase()
    use_case.execute(request.args)
```



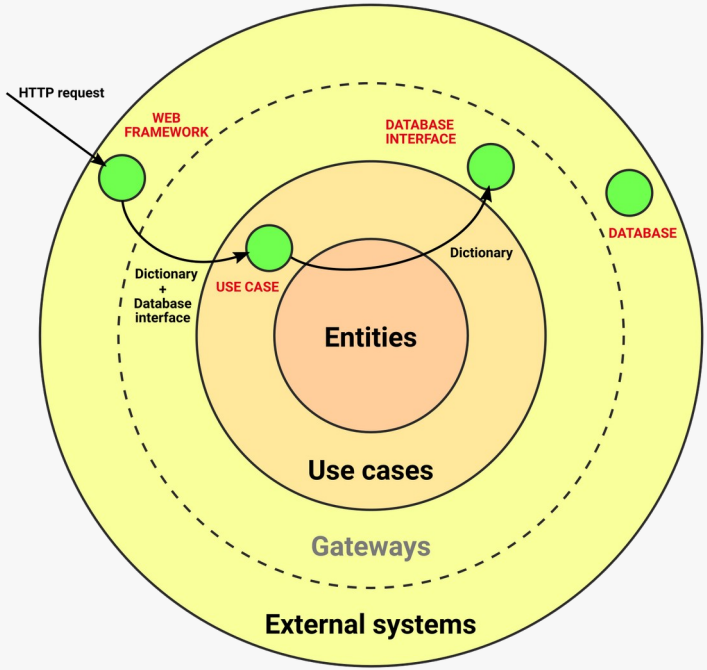
The use case receives the **repository interface** as an argument of the call

```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    use_case = uc.ItemsListUseCase(repo)
    use_case.execute(request.args)
```



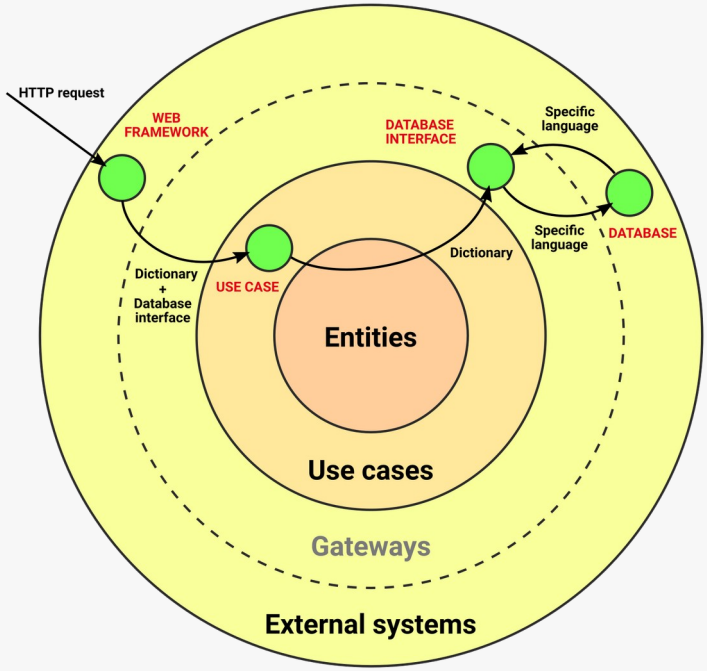
The use case queries the repository interface with **simple structures**

```
class ItemsListUseCase:

    def __init__(self, repo):
        self.repo = repo

    def execute(self, params):

        # BUSINESS LOGIC HERE

        result = self.repo.list(params)

        # BUSINESS LOGIC HERE

        return result
```



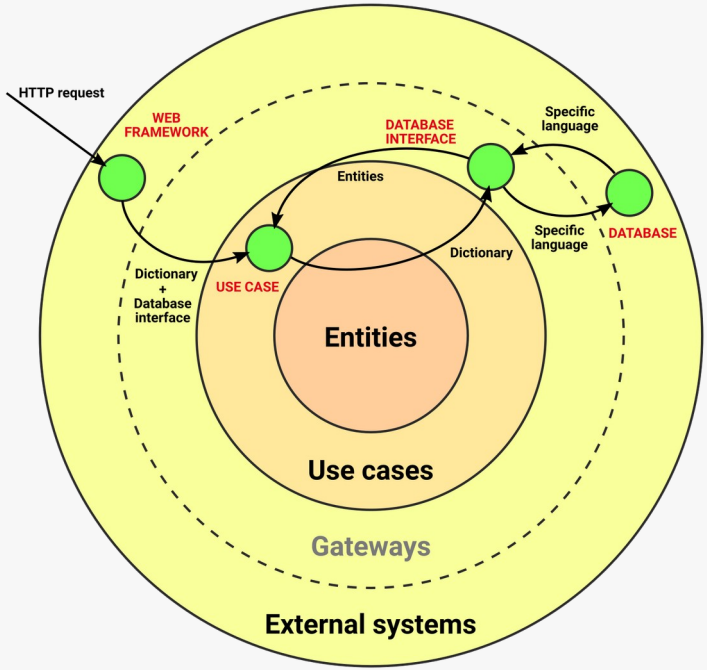
The database interface and the database exchange data in a **specific language**

```
class PostgresRepo:
    def __init__(self, CONNECTION_STRING):
        self.ng = create_engine(
                        CONNECTION_STRING)
        Base.metadata.bind = self.ng

    def list(self, filters):
        DBSession = sessionmaker(bind=self.ng)
        session = DBSession()

        query = ...
```



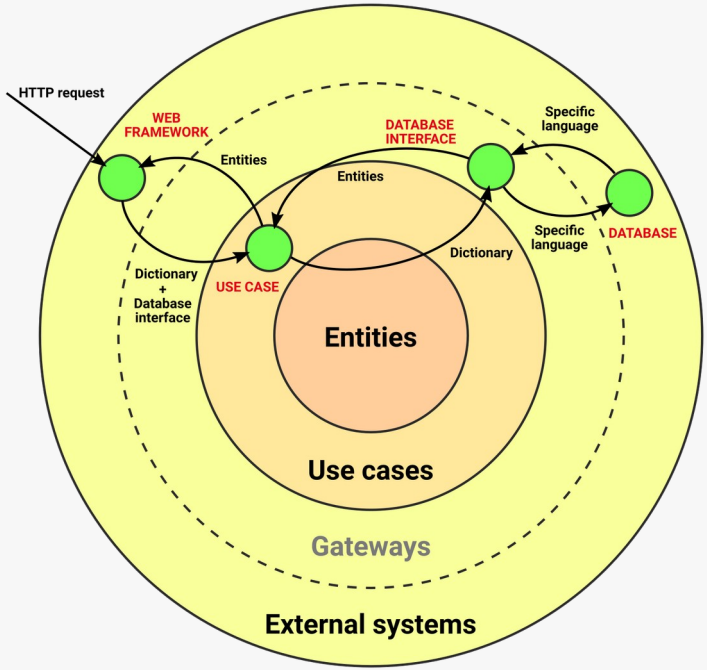
The database interface translates the specific language into **simple structures** and **entities**

```
class PostgresRepo:
    def __init__(self, CONNECTION_STRING):
        self.ng = create_engine(
                        CONNECTION_STRING)
        Base.metadata.bind = self.ng

    def _create_items(self, results):
        return [Item(code=q.code, price=q.price)
            for q in results]

    def list(self, filters):
        DBSession = sessionmaker(bind=self.ng)
        session = DBSession()

        query = ...

        return self._create_items(query.all())
```



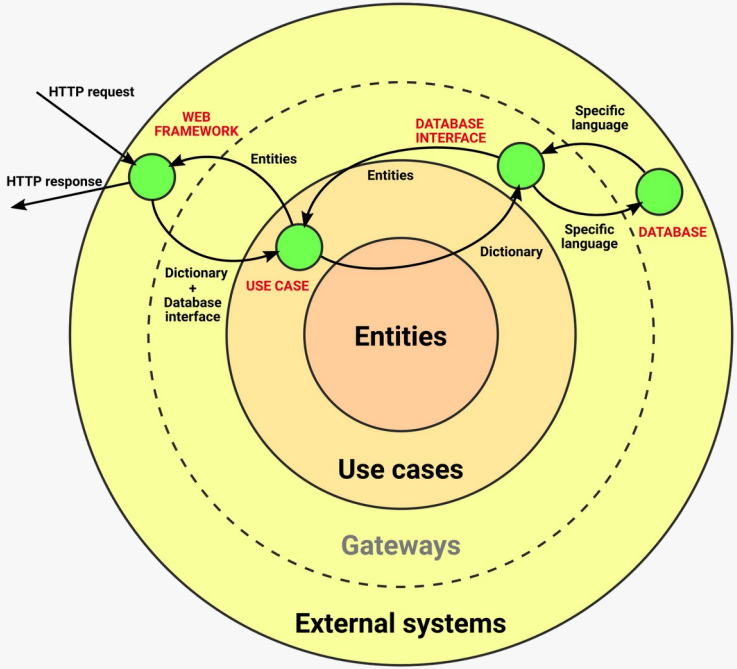
The use case returns the result of the business logic: **entities** and **simple structures**

```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    use_case = uc.ItemsListUseCase(repo)
    result = use_case.execute(request.args)
```



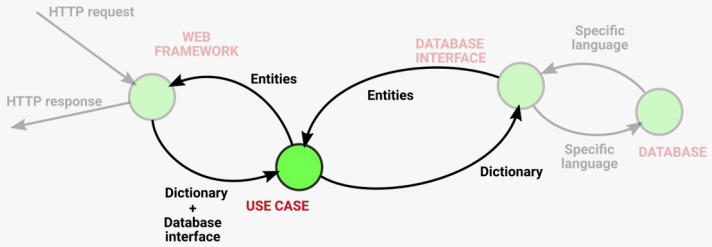
The web framework converts entities and simple structures into **HTTP responses**

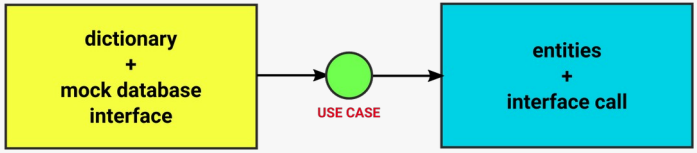
```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    use_case = uc.ItemsListUseCase(repo)
    result = use_case.execute(request.args)

    return Response(
        json.dumps(result),
        mimetype='application/json',
        status=200)
```



Testing the **use case**

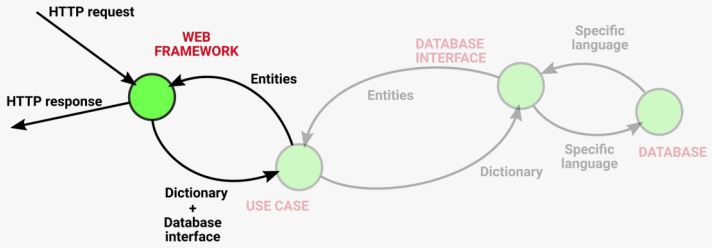
```
class ItemsListUseCase:

    def __init__(self, repo):
        self.repo = repo

    def execute(self, params):

        # BUSINESS LOGIC HERE

        result = self.repo.list(params)

        # BUSINESS LOGIC HERE

        return result
```

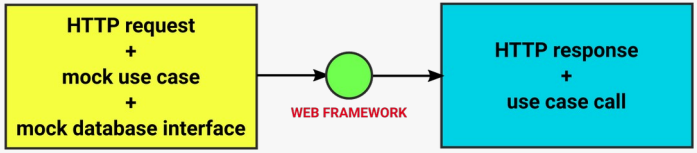




Testing the **HTTP endpoint**

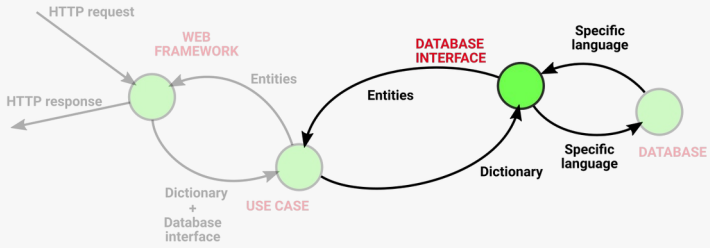
```
@blueprint.route('/items', methods=['GET'])
def items():
    repo = PostgresRepo(CONNECTION_STRING)
    use_case = uc.ItemsListUseCase(repo)
    result = use_case.execute(request.args)

    return Response(
        json.dumps(result),
        mimetype='application/json',
        status=200)
```

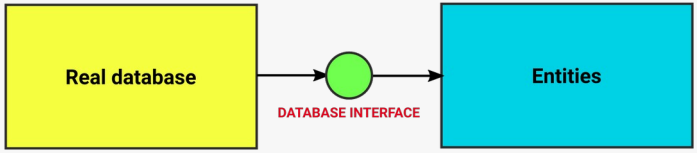


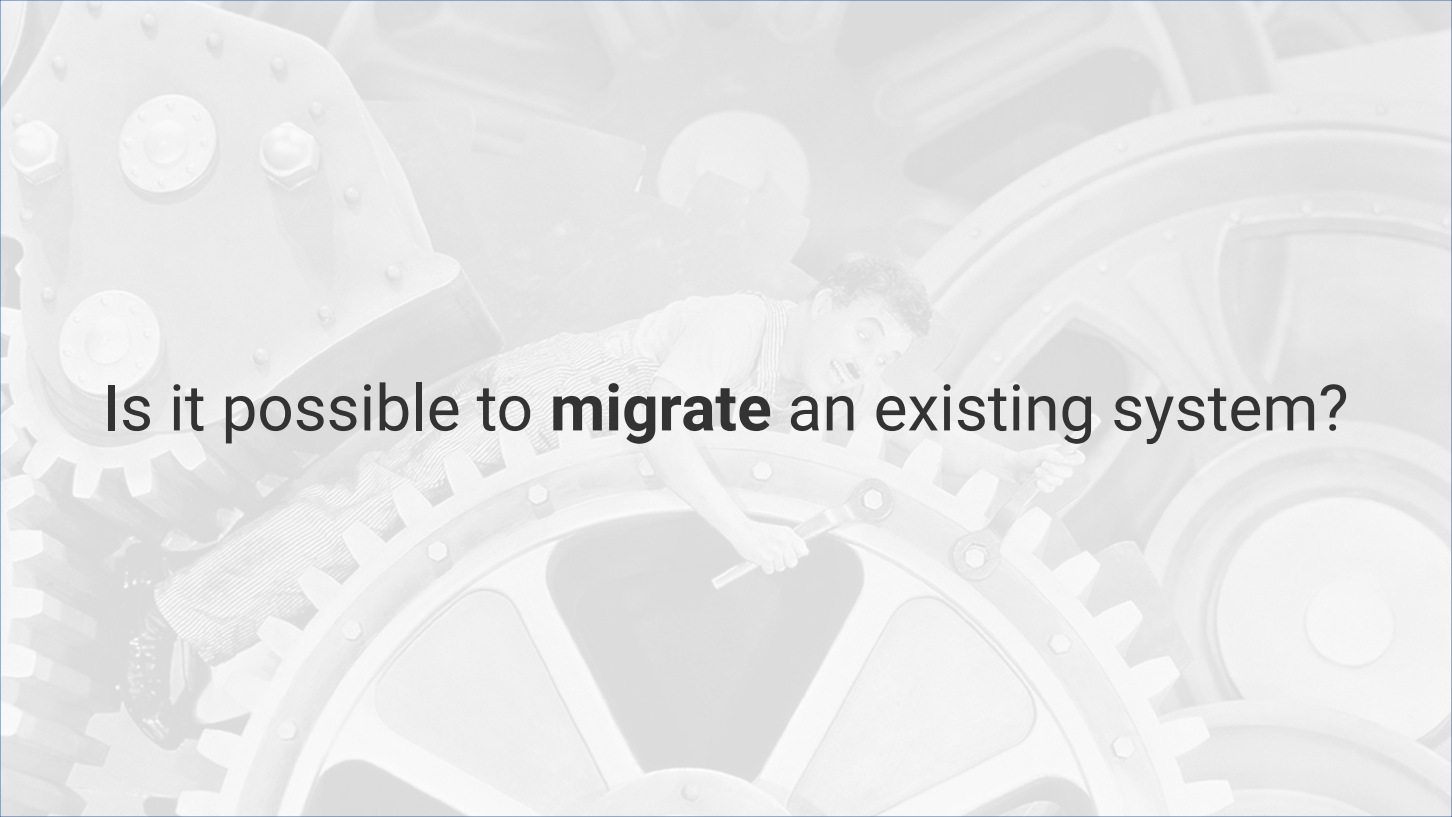


Testing the repository interface: **integration test**

```
class PostgresRepo:
    def __init__(self, CONNECTION_STRING):
        self.ng = create_engine(
                        CONNECTION_STRING)
        Base.metadata.bind = self.ng

    def _create_items(self, results):
        return [Item(code=q.code, price=q.price)
            for q in results]

    def list(self, filters):
        DBSession = sessionmaker(bind=self.ng)
        session = DBSession()

        query = ...

        return self._create_items(query.all())
```

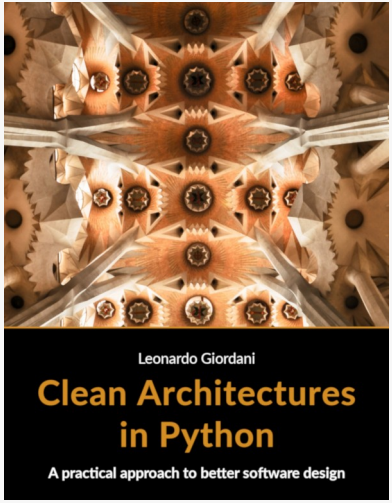






Is it possible to **migrate** an existing system?


Is this the **definitive** architecture?

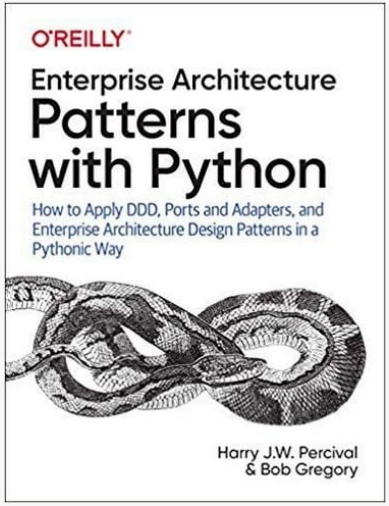


Leonardo Giordani

Clean Architectures in Python

A practical approach to better software design

bit.ly/getpycabook



Harry Percival, Bob Gregory

Enterprise Architecture Patterns with Python

`github.com/python-leap/book`

(Published by O'Reilly)

Thank you!

@tw_lgiordani - @thedigicat - bit.ly/getpycabook

https://speakerdeck.com/lgiordani