



DGTL



Внедряя SOLID

Денис Аникин, <https://xfenix.ru>

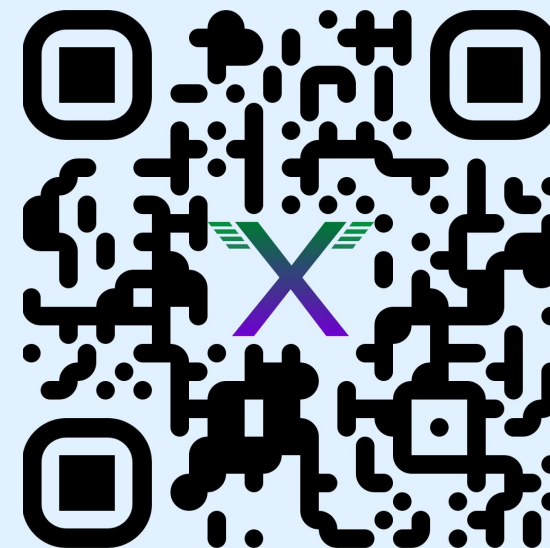
Кто я?



Что я такое?

- Я техлид/комьюнити лид
- fullstack, python, typescript, devops, микросервисы, kubernetes
- Выступаю на конференциях
- Отвечаю за внутреннее сообщество питонистов

<https://xfenix.ru>



Мотивация доклада



- С SOLID проще поддерживать код
- С SOLID проще читать код
- С SOLID проще тестировать код
- Это все сложно доказать и обосновать (в книге написано «делайте так», есть и аргументы, но всё, в основном, на словах)
- Наше ремесло полагается на опыт больше, чем на теорию

Поговорим о пользе SOLID

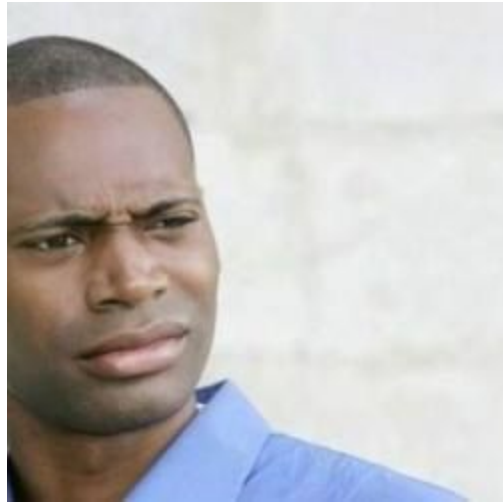


- Поговорим о мнениях про SOLID
- Пройдем по каждой букве
- Разберемся в деталях её внедрения и заблуждениях вокруг
- Я постараюсь показать пользу каждой буквы

Мнения о SOLID

Индустрия говорит







DGTL

«Я его чувствую»

Почти любой
разработчик о SOLID





DGTL

«Все его понимают»

Другой разработчик





DGTL

«Каждый знает SOLID»

Ещё один разработчик





DGTL

«ЭТО ОЧЕВИДНО»

И ещё один разработчик





DGTL

«Никому не надо объяснять»

Почти любой
разработчик о SOLID





DGTL

«SOLID применяю каждый день! Сейчас вспомню эээ Single что-то там и эээ Лисков что-то...»

Лучшая цитата про SOLID, шлягер 2010 — 2024





«**Всё давно сказано**»

и такое тоже можно услышать



«SOLID вообще
не нужен, а дядя Боб —
инфобизнесмен»

и такое тоже можно услышать



«SOLID должен знать каждый, а иначе он не программист»

программист с главным, правильным и единственным мнением
важный техлид (вчера им стал)
инфлюенсеры, программисты с «вертикальным» мнением



excuse me зачем нужен этот доклад???

Breaking news



Это всё неправда

и я постараюсь обосновать почему



Чувствую SOLID от природы



Разработчик

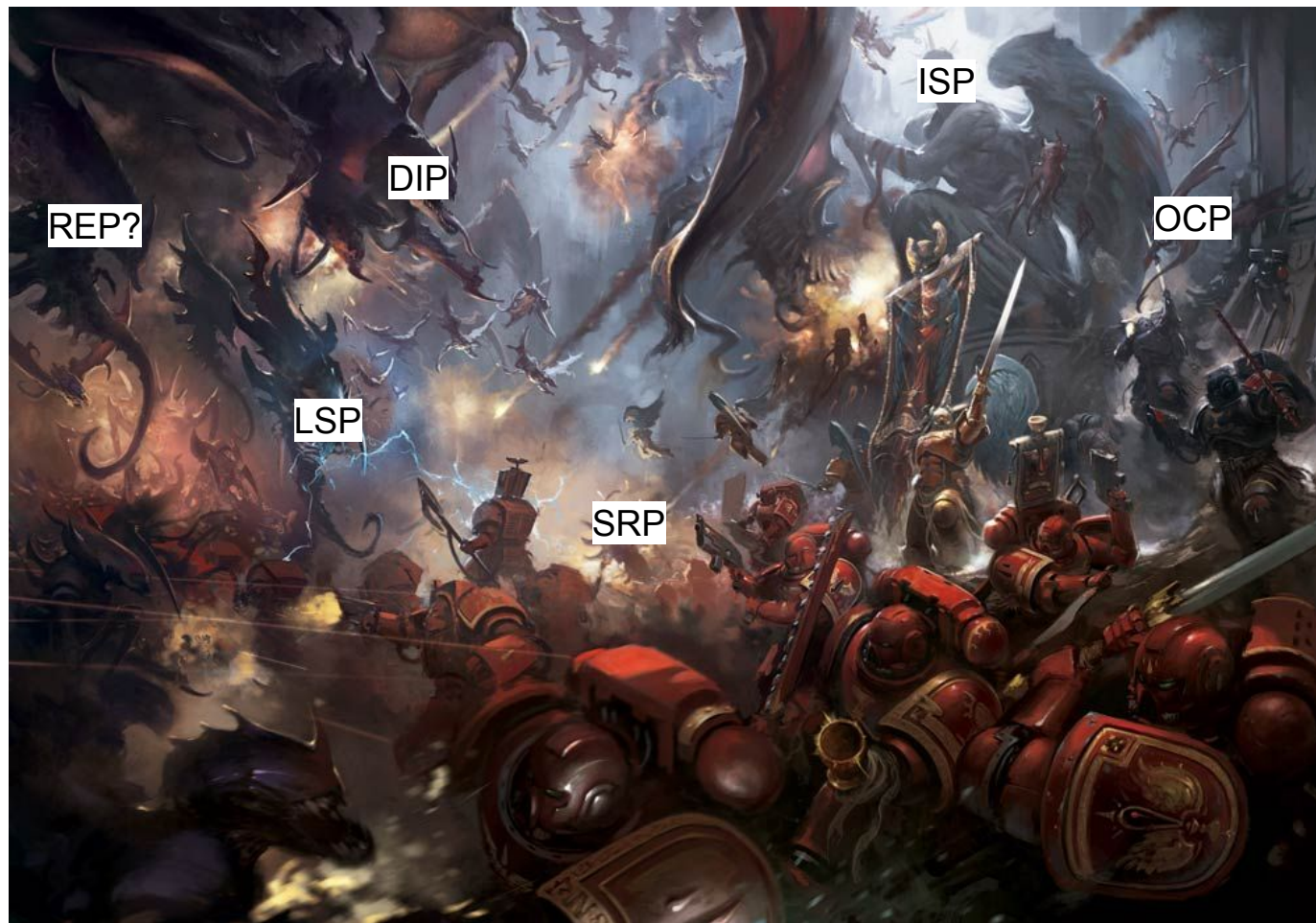


Почему с SOLID хаос



- Принципы SOLID — сложные!
- Трактовка SOLID — не проста, а местами противоречива
- Всё давно сказано настолько, что SOLID подавляющее большинство не может объяснить ни на собеседованиях, ни в реальной жизни
- От рождения невозможно получить «инстинкт» и «чувство» SOLID
- Приведу простой пример: даже «базовые» термины, такие как интерфейс или архитектура чаще всего имеют разные трактовки у разных людей (например, я могу выделить (как минимум) 3 вида архитектуры)
- В докладе мы обратимся к оригинальным публикациям и будет понятнее

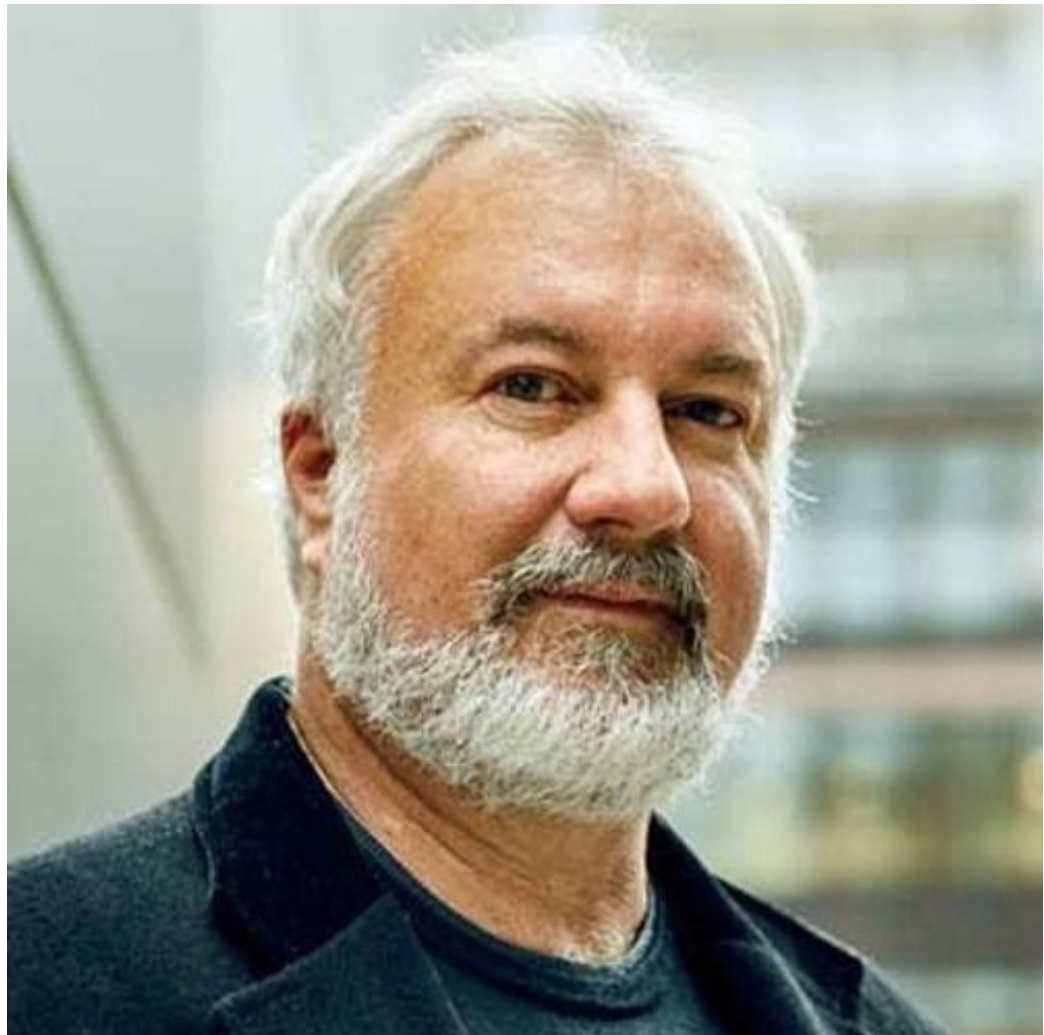
Разработчики внедряют SOLID+



Возникновение SOLID



Придумал название Michael Feathers



Резюме



Michael Feathers
Director at R7K Research & Conveyance



Experience



Director

R7K Research & Conveyance
Dec 2013 - Present · 10 yrs 10 mos
Miami/Fort Lauderdale Area



Advisory Board

CodeScene
Sep 2019 - Present · 5 yrs 1 mo
Malmö, Sweden · Remote



Chief Architect

Globant
Oct 2020 - Aug 2023 · 2 yrs 11 mos



Member of Technical Staff

Groupon
Aug 2011 - Sep 2012 · 1 yr 2 mos
Miami, FL



Chief Scientist

Obtiva
Feb 2011 - Aug 2011 · 7 mos
Miami, FL



Senior Consultant

Object Mentor
Feb 1999 - Feb 2011 · 12 yrs 1 mo



Beckman Coulter

8 yrs 1 mo



Sr. Software Engineer II

1995 - 1999 · 4 yrs

ids=R7K+Research+%26+Conveyance II, Sr. Software Engineer

Распиарил Robert C. Martin



Резюме



Experience



Owner

Uncle Bob Consulting LLC.

Sep 2010 - Present · 14 yrs 1 mo

Software training and consulting.



President

Object Mentor Inc.

1991 - 2010 · 19 yrs

Founded and built the business. Trainer, consultant, speaker.



Contractor

Rational

1990 - 1991 · 1 yr

Worked with Grady Booch on the first release of Rose.



Development Manager

Clear Communications, Inc.

1986 - 1990 · 4 yrs

Managed development of T1 Network Quality Testing System.



Development Manager

Teradyne Inc. - Telecommunications Division

1976 - 1988 · 12 yrs

A LOT of software.

К чему я это



- Что Майкл, что Роберт код писали ну очень давно
- Большую часть жизни, причём с 80х, они консультируют как писать большие кодовые базы (миллионы строк)
- Мы сейчас пишем куда меньшие кодовые базы (микросервисы)
- Результат их работы для нас не очевиден
- Книги содержат неточные правила, а абстрактные идеи, подкрепленные личными мнениями
- Поэтому SOLID так непросто внедрять в наши проекты
- Но это не значит, что он не полезен

Таймлайн SOLID (кусоч)



2000

Публикация о
«гниении» софта

Design Principles
and Design Patterns

2003

Принципы
изложены вместе

Agile Software
Development,
Principles, Patterns,
and Practices

2004

Придумано
название **SOLID**

2014

Уточнение SRP

А это к чему?



- К мифу «всё давно придумано» и «давно все всё знают»
- Не так давно
- Питон в энтерпрайз шагнул не так давно
- Принципы — почти «свежак» (*примечание: OCP 1988, LSP 1987*)



Как мы видим —
вокруг SOLID
бушует океан

Что такое SOLID

Поговорим о буквах





Изучаем Design Principles and Design Patterns

Single Responsibility

DGTL



Принцип единой ответственности
SRP



SRP: The Single-Responsibility Principle



None but Buddha himself must take the responsibility of giving out occult secrets...

—E. Cobham Brewer 1810–1897.
Dictionary of Phrase and Fable. 1898.

Что это



- У класса должна быть одна причина для изменений
- Его можно применять универсально не только к ООП, но и к функциям, и к модулям и к микросервисам (тут много мнений почему так)



Принцип не вошел в изначальную публикацию

но его можно найти в The Principles of OOD и
Agile Software Development, Principles, Patterns, and Practices



Слово «причина» вызвала непонимание длиной в 10+ лет

в 2014 Роберту пришлось писать статью с объяснением принципа

И вот тут важно проговорить



- SOLID всего 10 лет назад получил «дообъяснение»
- Люди, говорящие, «всё давно понятно», «всё всем известно» в этот момент должны глубоко задуматься и перестать кидаться этими фразами
- Слово «причина» — понятное (казалось), а на деле, как видим, в контексте принципа даже такое слово вызывает вопросы
- Статья 2014 года всё так же абстрактна
- В оригинальной книге 2003 года автор приводит интересный кейс с модемом...
- В книге можно встретить интересные трактовки вроде «не имеет смысла применять к коду, который не меняется» (смысл условно передан)

Практика SRP



Что думает автор

- Принцип прост и понятен
- Позволяет избегать хрупкости софта
- Он вводит много понятий, таких как «ось изменений»
- Нарушение приводит к сбоям

Как в жизни?

- Принцип очень. сложно. понять
- Смысла ломать копья в поиске правильной трактовки нет.
Договоритесь внутри как вы читаете слово «причина»
- Соблюдение, кажется, делает код чище и проще в тестировании

Как внедрять?



- Имеет смысл применять для классов, а так же функций, модулей и даже микросервисов
- Проверять его можно только одним способом: задавать себе вопрос «соблюдаем ли мы тут SRP?»
- Главный помощник — code review с внимательными ревьюерами
- Внутри вы должны договориться о том, что такое «причина» изменений, а так же иметь людей с, которые научились видеть нарушение SRP

SRP пример из книги



```
interface Modem
{
    public void dial(String pno); // так себе имя аргумента
    public void hangup();
    public void send(char c); // так себе имя аргумента
    public char recv();
}
```


SRP пример из книги



- Две ответственности («причины») для изменений с точки зрения автора
- 1 — передача данных
- 2 — управление подключением
- Разделять имеет смысл, если часто будет меняться передача данных
- С этим я не очень согласен

SRP — мой пример (так плохо)



```
class UserRepository:
    def __init__(self):
        self.database_connection = init_database_conn()

    def fetch_users(...):
        ...

    def update_one_user(...):
        ...
```

SRP — мой пример (так получше)



```
@dataclass
class UserRepository:
    database_connection: DBConn

    def fetch_users(...):
        ...

    def update_one_user(...):
        ...
```

Open-Closed

DGTL



Принцип открытости-закрытости
ОСР

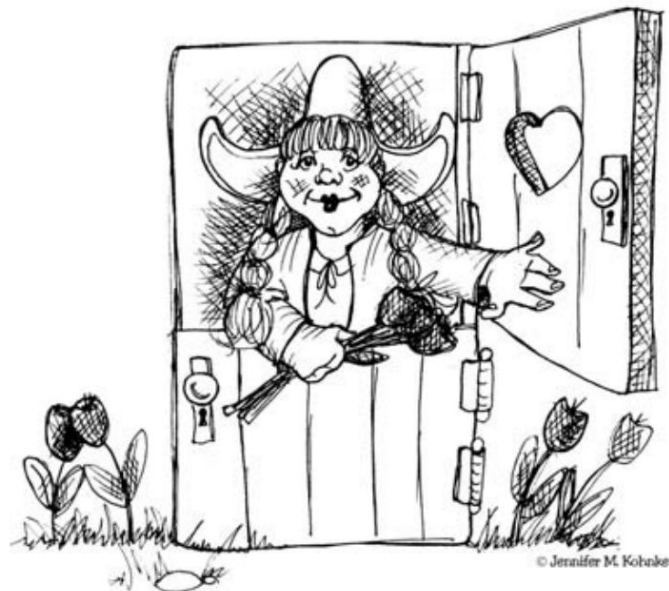
Что это



- Программные сущности стоит развивать путем расширения, а не модификации исходного кода
- ИЛИ программа развивается путем добавления кода, а не изменения существующего
- Принцип так же универсально применим и к модулям, функциям



OCP: The Open–Closed Principle



***Dutch Door**—(Noun) A door divided in two horizontally so that either part can be left open or closed.*

—The American Heritage[®] Dictionary of the English Language: Fourth Edition. 2000.

Практика ОСР



Что думает автор

- Принцип позволяет создавать расширяемый код без изменений текущего
- Идеал трудно достичь
- Даже частичное соблюдение помогает улучшить структуру
- Позволяет меньше ломать текущий код

Как в жизни?

- Принцип плохо трактуют
- Многие чаще рефакторят существующие классы и прочие объекты, чем дополняют их
- Если тестировать не юнитами, а «интеграцией», то код ломается не часто
- Если завязываться на контракты, есть шанс соблюсти

ОСР — плохо



```
class Shape:
    def calc_area(self):
        ...

class Rectangle(Shape):
    def calc_area(self):
        ...

class Circle(Shape):
    def calc_area(self):
        ...

def draw_shape_ocp_bad(self, shapes):
    size = 0
    for shape in shapes:
        if isinstance(shape, Rectangle):
            size += shape.width * shape.height
        elif isinstance(shape, Circle):
            size += 3.14 * shape.radius ** 2
    return size
```


ОСР — хорошо



```
class Shape:
    def calc_area(self):
        ...

class Rectangle(Shape):
    def calc_area(self):
        ...

class Circle(Shape):
    def calc_area(self):
        ...

def draw_shape_ocp_good(shapes):
    size = 0
    for shape in shapes:
        size += shape.calc_area()
    return size
```

Как внедрять

- Только вручную
- Только code review



Важные выводы



- Изменять текущие классы не обязательно запрещено, а скорее стоит минимизировать это
- Имеет смысл завязываться на контракты и интерфейсы при развитии софта
- Композиция > наследование
- + в копилку DI

Liskov substitution

DGTL



Принцип подстановки Барбары Лисков
LSP



LSP: The Liskov Substitution Principle



Что это



- В вашей программе можно заменить типы на подтипы и она не сломается
- Можно читать «типы» как «классы»
- Подклассы не должны накладывать больше ограничений, чем есть у их предков и давать меньше, чем их предки



The pronunciation of **Barbara Liskov's** name, with the emphasis on the first syllable (**LIS-kov**), follows common English pronunciation patterns for names of Slavic origin when adapted to English. While in many Slavic languages, the stress might fall on the second syllable, names often shift in pronunciation when they enter a different linguistic context, such as English.

In English, stress often tends to be placed on the first syllable of names, which explains why Liskov is pronounced **LIS-kov**. Additionally, since Barbara Liskov is American, the English pronunciation reflects her cultural and linguistic context rather than the original Slavic pronunciation.

< 2/2 > 🔊 📄 👍 🗨️ 🔄

Can i pronounce her surname in slavic culture on russian language with stress on second syllable

Yes, in the context of Slavic languages, especially Russian, it would be natural to pronounce Liskov with the stress on the second syllable, as **Lis-KOV**. Russian names and surnames often have the stress on the last syllable, so this pronunciation would align with how a native Russian speaker might pronounce it.

If you're speaking in a Russian or Slavic cultural context, it's perfectly acceptable to use this pronunciation. However, in English contexts, especially when referring to Barbara Liskov herself, the stress on the first syllable (**LIS-kov**) is more common.



Вам конец, если вы
его не соблюдаете

Один разработчик



Это мега-очевидно

Другой разработчик

LSP — соблюдение



```
class Dep1:
```

```
...
```

```
class Dep2(Dep1):
```

```
...
```

```
class A:
```

```
    def get_some(self, argument: Dep2):
```

```
    ...
```

```
class B(A):
```

```
    def get_some(self, argument: Dep1):
```

```
    ...
```



Вот это и было то самое
«мега-очевидно»

Это уже я говорю

Чуть детальнее



- Если вы не поняли, это норм
- Я бы тоже не понял сразу
- Если рядом сидит человек, который говорит, что это очевидно, осторожно, не привлекая внимания, пересядьте :)

LSP нарушение



```
class Dep1:
```

```
...
```

```
class Dep2(Dep1):
```

```
...
```

```
class A:
```

```
    def get_some(self, argument: Dep1):
```

```
    ...
```

```
class B(A):
```

```
    def get_some(self, argument: Dep2): # вот тут тип «сузили»
```

```
        # если мы везде заменим A на B
```

```
        # то получится функция get_some получает более специфичную реализацию
```

```
        # аргумента и это может привести к сбоям
```



Нарушение LSP является скрытым нарушением OSR

Мысль из оригинальной книги

Практика LSP



Что думает автор

- Очень много всего (глава одна из самых больших про солид)
- LSP — главный «энеблер» OCR
- LSP уменьшает хрупкость вашего кода

Как в жизни?

- LSP не очень многие понимают
- Муру помогает
- Полностью его соблюдения я не видел, но это не значит, что не стоит пытаться
- Несоблюдение принципа тянет за собой остальной SOLID

Как внедрять?



- В первую очередь — писать аннотации
- Во вторую — подключать туру
- В третью — проводить ревью



mypy Playground

Run

Gist

mypy latest (1.11.2) ▾

Python 3.12 ▾

Options

```
1 class Dep1:
2     ...
3
4
5 class Dep2(Dep1):
6     ...
7
8
9 class A:
10     def get_some(self, argument: Dep1):
11         ...
12
13 class B(A):
14     def get_some(self, argument: Dep2): # вот тут тип «сузили»
```

```
1 [X] Argument 1 of "get_some" is incompatible with supertype "A"; supertype defines the argument type
1 as "Dep1" [override]
1 | This violates the Liskov substitution principle
1 | See https://mypy.readthedocs.io/en/stable/common_issues.html#incompatible-overrides
```



mypy 1.11.2 documentation

🔍 Search

FIRST STEPS

[Getting started](#)

[Type hints cheat sheet](#)

[Using mypy with an existing codebase](#)

TYPE SYSTEM REFERENCE

[Built-in types](#)

[Type inference and type annotations](#)

[Kinds of types](#)

[Class basics](#)

[Annotation issues at runtime](#)

[Protocols and structural subtyping](#)

[Dynamically typed code](#)

Incompatible overrides

It's unsafe to override a method with a more specific argument type, as it violates the [Liskov substitution principle](#). For return types, it's unsafe to override a method with a more general return type.

Other incompatible signature changes in method overrides, such as adding an extra required parameter, or removing an optional parameter, will also generate errors. The signature of a method in a subclass should accept all valid calls to the base class method. Mypy treats a subclass as a subtype of the base class. An instance of a subclass is valid everywhere where an instance of the base class is valid.

This example demonstrates both safe and unsafe overrides:

```
from typing import Sequence, List, Iterable

class A:
    def test(self, t: Sequence[int]) -> Sequence[str]:
        ...

class GeneralizedArgument(A):
    # A more general argument type is okay
    def test(self, t: Iterable[int]) -> Sequence[str]: # OK
        ...

class NarrowerArgument(A):
    # A more specific argument type isn't accepted
    def test(self, t: List[int]) -> Sequence[str]: # Error
        ...
```

LSP пример 2 (из книги)



```
class Shape:
    def draw(self):
        ...

@dataclass
class Circle(Shape):
    center: Point
    radius: float

    def draw(self):
        ...

@dataclass
class Square(Shape):
    top_left: Point
    side: float

    def draw(self):
        ...

def draw_shape(shape):
    if shape.shape_type == ShapeType.SQUARE:
        shape.draw()
    elif shape.shape_type == ShapeType.CIRCLE:
        shape.draw()
```



Муру это не ОТЛОВИТ

Увы

Interface segregation

DGTL



Принцип разделения интерфейсов
ISP



Тут нет картинки

Её правда нет в оригинальной книге :)

Что это



- Вместо убер-интерфейса делайте много маленьких, специфичных
- Делайте интерфейсы под потребителей

Неоднозначности в python



- В python нет интерфейсов
- Есть `typing.Protocol`
- Есть `abc`
- Можно делать «приватные» и «протектед» методы
- `raise NotImplementedError`

Практика ISP



Что думает автор

- «Жирные» интерфейсы плохо
- Потребители должны знать об абстракциях/интерфейсах/контрактах
- Наличие лишних методов для потребителей приводит к нарушению и LSP принципа
- Делегирование и множественное наследование помогает

Как в жизни?

- Принцип многие понимают
- На деле все довольно далеко от книги
- На практике все сводится к идее — делайте «интерфейсы» покороче, для одной цели, разделяйте по назначению

ISP — плохо



```
class Worker:
    def work(self):
        pass

    def eat(self):
        pass

class HumanWorker(Worker):
    def work(self):
        ...

    def eat(self):
        ...

class RobotWorker(Worker):
    def work(self):
        ...

    # Нарушает ISP: RobotWorker не нужен `eat`
    def eat(self):
        raise NotImplementedError("Robots don't eat!")
```

ISP — хорошо



```
class Workable:  
    def work(self):  
        pass
```

```
class Eatable:  
    def eat(self):  
        pass
```

```
class HumanWorker(Workable, Eatable):  
    def work(self):  
        ...  
  
    def eat(self):  
        ...
```

```
class RobotWorker(Workable):  
    def work(self):  
        ...
```

```
# eat не нужен, ISP соблюли
```

Как внедрять?

— Опять — руками

— Code review



Dependency inversion

DGTL



Принцип инверсии зависимостей
DIP



DIP: The Dependency-Inversion Principle



© Jennifer M. Kohnke

*Nevermore
Let the great interests of the State depend
Upon the thousand chances that may sway
A piece of human frailty*

—Sir Thomas Noon Talfourd (1795–1854)



@funkindy 2 года назад (изменено)

Depends это элегантный способ внедрения зависимости, а не инверсия зависимостей. Вспомни SOLID.

@АртемИванов-й4р 2 года назад

Чувак, инверсия зависимостей и внедрение зависимостей (Depends()) это разные вещи, посмотри



5



Ответить

@kandreyk9159 2 года назад

внедрение зависимостей это частный случай инверсии управления



1

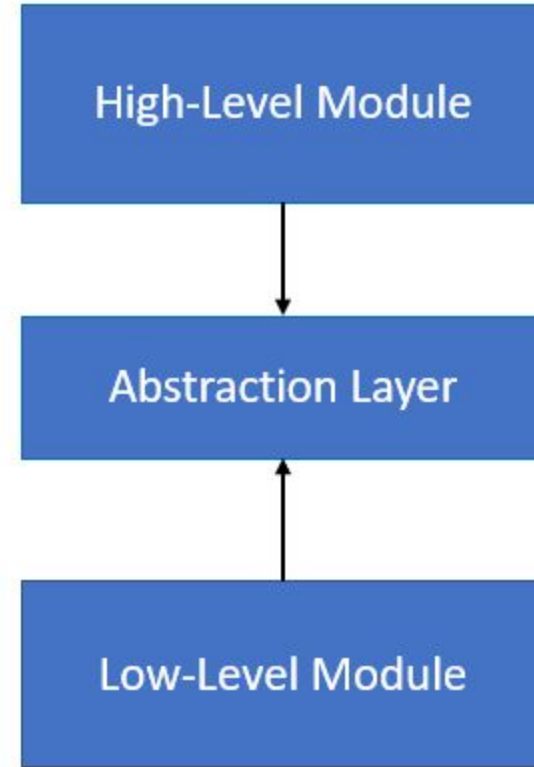
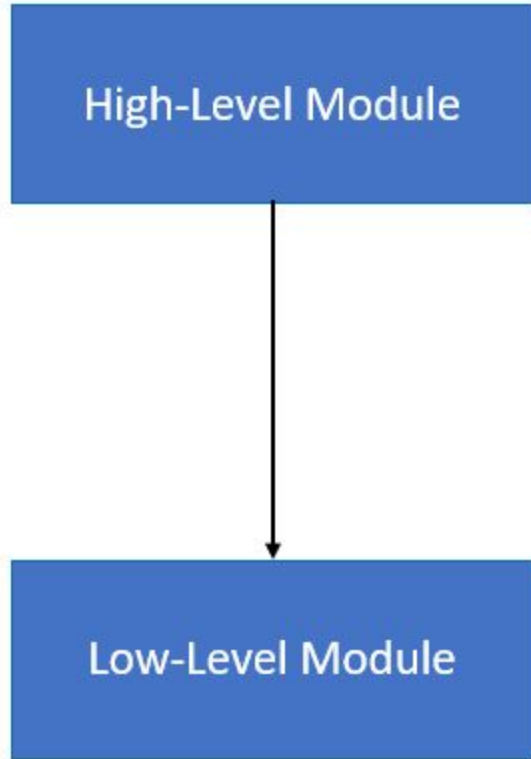


Ответить

Минутка буквоедства!



- **D**ependency **I**nversion **P**rinciple и **D**ependency **I**njection — разные вещи!
- DIP — принцип
- DI — паттерн





DI

Помогает достигать DIP (в моей интерпретации)

DI: без него



```
class ImportantApp:  
    def do_staff(self):  
        ...  
        self.other_app = OtherApp()  
        ...
```

DI: первые шаги



```
class ImportantApp:  
    def do_staff(self, other_app: OtherApp):  
        ...
```

DI: типовой класс



```
@dataclass.dataclass(frozen=True, slots=True, kw_only=True)
class ImportantApp:
    other_app: OtherApp

    def do_staff(self):
        ...
```

DI: финальный шаг — фреймворки



- That depends
- Dishka
- Встроенные в fastapi, littestar, faststream
- Дополнительно: punq, blacksheep + rodi (проплачено Колей Хитровым)



Но не только DI

есть кое-что ещё



SVCS

A Flexible Service Locator for Python.

Пример использования



Plain

```
import svcs

async def example_view(rqst):
    db, api, cache = \
        await svcs.starlette.aget(
            rqst, Database, WebAPIClient, Cache
        )
```

FastAPI

```
import svcs

@app.get("/")
async def example_view(
    services: svcs.fastapi.DepContainer
):
    db, api, cache = \
        await services.aget(
            Database, WebAPIClient, Cache
        )
```

Как регистрировать



```
>>> import svcs
>>> import uuid

>>> registry = svcs.Registry()

>>> registry.register_factory(uuid.UUID, uuid.uuid4)
>>> registry.register_value(str, "Hello World")
>>> uuid.UUID in registry
True
>>> str in registry
True
>>> int in registry
False
```

Практика DIP



Что думает автор

- Зависеть от деталей плохая идея
- Делайте зависимости на абстракциях

Как в жизни?

- Не все умеют «сырой» DI
- Мало кто пользуется DI фреймворками
- Увидев код с DI, вы (скорее всего) другой писать не захотите
- Мысли автора здесь наиболее понятны
- Польза от этого принципа наиболее наглядна
- DI помогает соблюдать OCP

Минутка буквоедства!



- Если в зависимости DI передавать не абстракцию, а конкретную реализацию — нарушает ли это принцип DIP?
- На эту тему разворачивается холивар
- SOLID довольно размыт

Экстра принципы

DGTL



В оригинальной публикации и статье автор ссылается еще на 6 принципов, которые могут быть полезны!

3 принципа группировки/пакетов



- REP (The Release Reuse Equivalency Principle) — объединяйте вместе то, что будет переиспользовано и добавляйте версионирование (в общем, «делайте пакеты»)
- CCP (The Common Closure Principle) — изменяемые вместе классы надо хранить вместе (пакет, файл, папка)
- CRP (The Common Reuse Principle) — используемые вместе классы надо хранить вместе (пакет, файл, папка)

3 принципа взаимодействия



- ADP (The Acyclic Dependencies Principle) — зависимости не должны иметь циклов
- SDP (The Stable Dependencies Principle) — объекты с большим количеством зависимостей должны зависеть от объектов с меньшим количеством зависимостей
- SAP (The Stable Abstractions Principle) — в начале вашей «пирамиды зависимостей» должны лежать наиболее абстрактные объекты/классы

А нужен ли SOLID?

Это не очевидно



Это главный вопрос

и я не буду здесь вас путать



Наш опыт — ПОЗИТИВЕН

у нас SOLID делает код лучше

Может быть, SOLID нужен и вам

Внедряя SOLID

Итоги



Проблемы внедрения



- На словах все хорошо
- На деле принципы имеют нарушение
- Даже SRP принцип время от времени нарушается
- Нарушителями становятся далеко не только новички, но вполне себе и опытные люди



Что же делать?

Главный вопрос

Ответ прост и сложен



- Ваша задача — иметь постоянный контроль
- Нельзя внедрять SOLID самотёком и ждать его автоматического применения
- Внедрение SOLID — постоянный бой с хаосом
- Внедрение SOLID — холивары (мой совет: не позволять их затягивать)

Вопросы на собеседованиях по SOLID



Вопросы



- Какой принцип самый главный
- Какой мы можем удалить?
- Что такое принципы SOLID?
- Расскажите как внедряли (после того как вы с жаром признаетесь, что SOLID — круто)

Выводы

SOLID



- Сложен
- Несёт пользу
- Не очевиден, не появляется автоматом, его нельзя «просто чувствовать»
- Необходимо последовательно и осмысленно внедрять
- Не страшно лишний раз повторять и углублять его понимание
- Не страшно его не понимать, не позволяйте себя за это осуждать и не осуждайте других, если они не до конца или вовсе не понимают
- Нет нужды говорить другим «я его внедряю» чисто из мысли, что «у всех же есть, все говорят что им пользуются», т.к. это ложный нарратив

SOLID — если вас ими «давят»



- Лучше всего у таких людей изучить их код (особенно на гитхабе)
- А на собеседованиях просите объяснить как они внедряют принципы SOLID

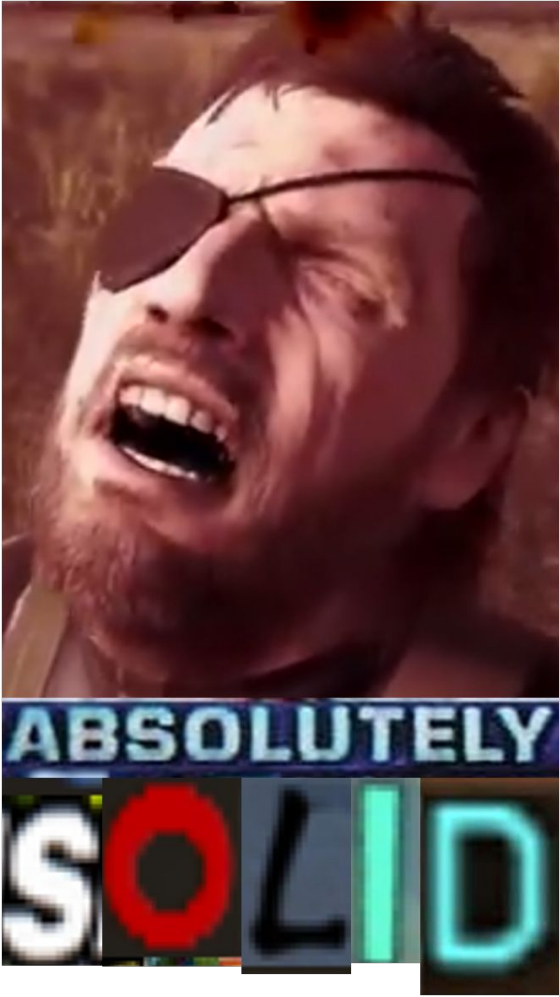
Ваша реакция на тему
«зачем обсуждать SOLID, всё уже давно понятно»





Давно всё **не**понятно

я поправил





DGTL

Спасибо, что
послушали

