ТИНЬКОФФ

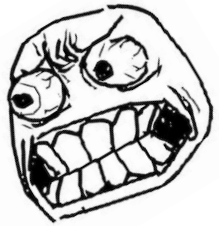
Как выжать максимум из Code Review

Бусидо программиста

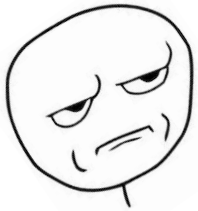
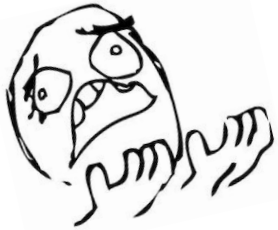
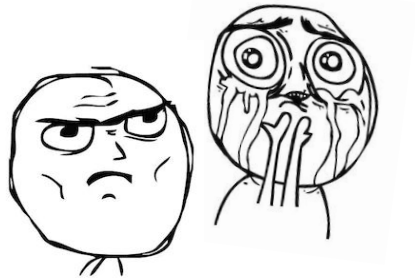
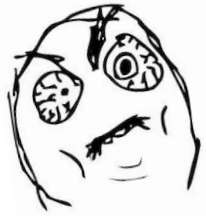
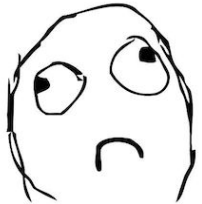
Дмитрий Бурмистров

07.11.2023



Code Review

Agenda

- пример становления процесса Code Review в команде
- мой взгляд на цели Code Review и его место в общей картине
- три столпа эффективности Code Review — взаимопонимание, знания, консолидация
- мой подход к разработке

Merged **Allow set token with a function and fix a error** #8

Changes from all commits · File filter · Conversations

13 src/orm/axios.js

28 src/support/interfaces.js

```
@@ -130,24 +130,18 @@ export const AxiosRequestConfig = {
```

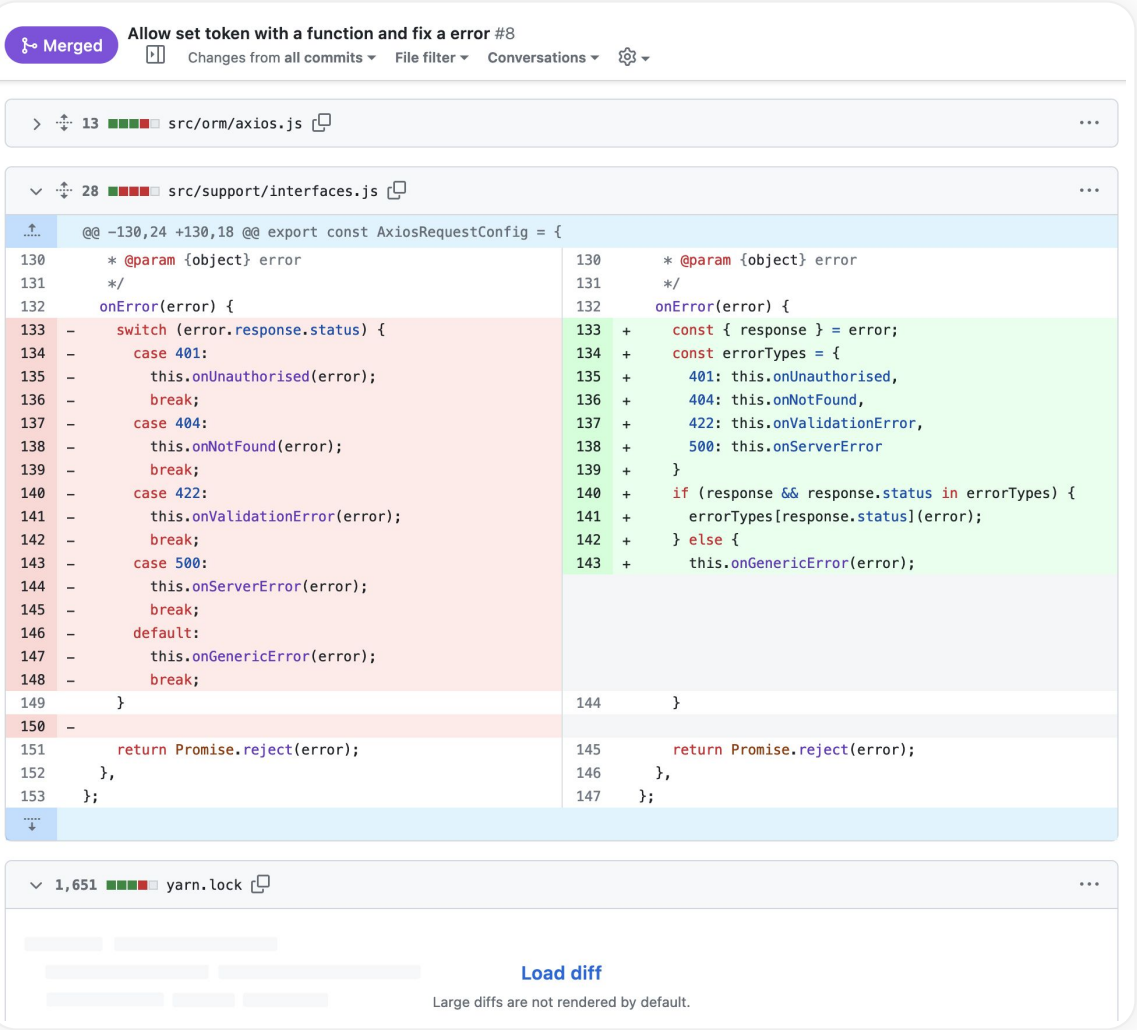
```
130     * @param {object} error
131     */
132    onError(error) {
133 -    switch (error.response.status) {
134 -      case 401:
135 -        this.onUnauthorised(error);
136 -        break;
137 -      case 404:
138 -        this.onNotFound(error);
139 -        break;
140 -      case 422:
141 -        this.onValidationError(error);
142 -        break;
143 -      case 500:
144 -        this.onServerError(error);
145 -        break;
146 -      default:
147 -        this.onGenericError(error);
148 -        break;
149      }
150 -
151      return Promise.reject(error);
152    },
153  };
```

```
130     * @param {object} error
131     */
132    onError(error) {
133 +    const { response } = error;
134 +    const errorTypes = {
135 +      401: this.onUnauthorised,
136 +      404: this.onNotFound,
137 +      422: this.onValidationError,
138 +      500: this.onServerError
139 +    }
140 +    if (response && response.status in errorTypes) {
141 +      errorTypes[response.status](error);
142 +    } else {
143 +      this.onGenericError(error);
144      }
145      return Promise.reject(error);
146    },
147  };
```

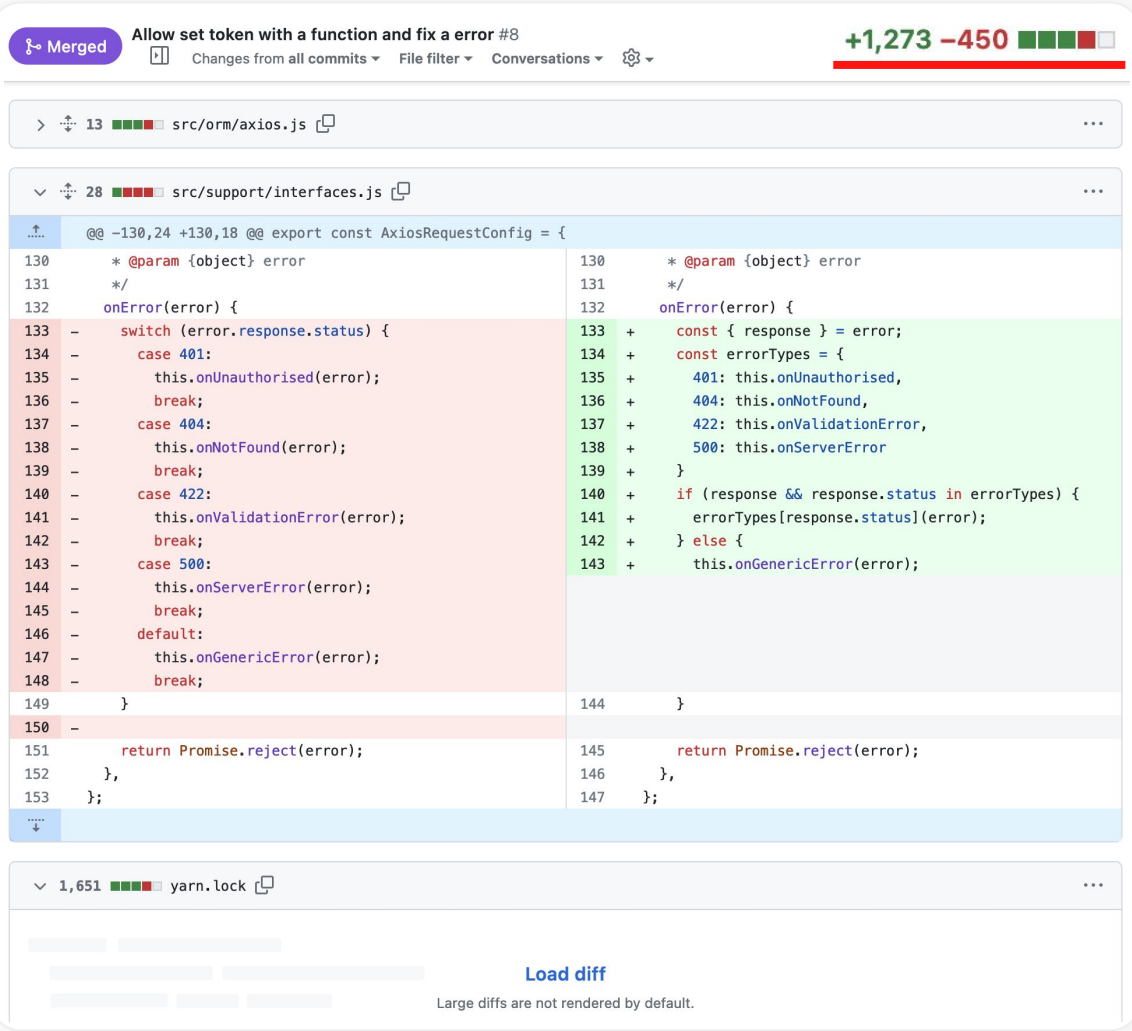
1,651 yarn.lock

Load diff

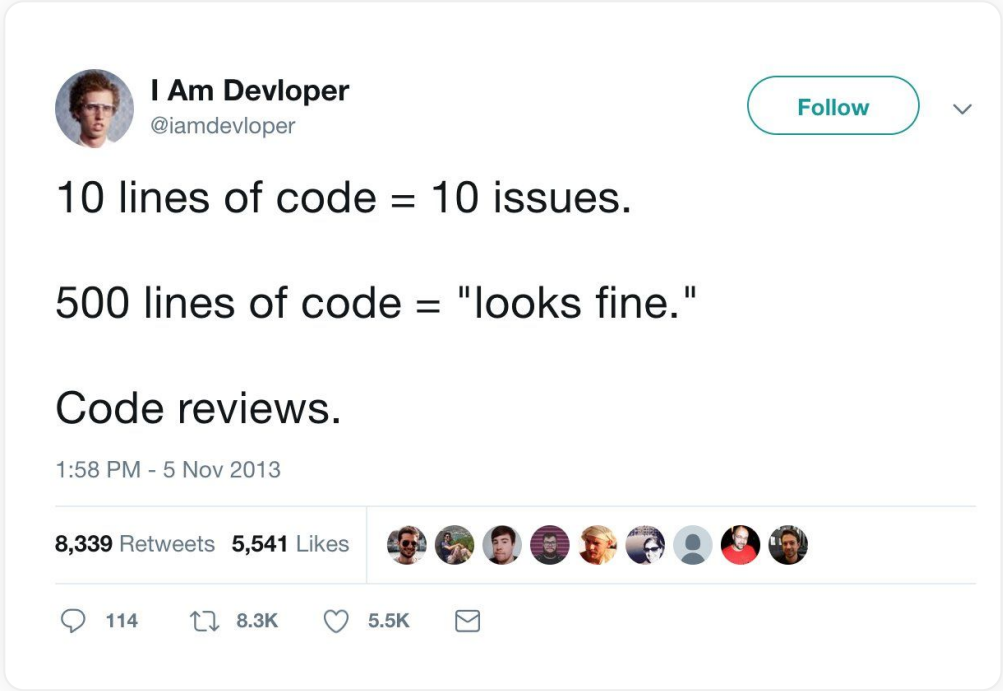
Large diffs are not rendered by default.

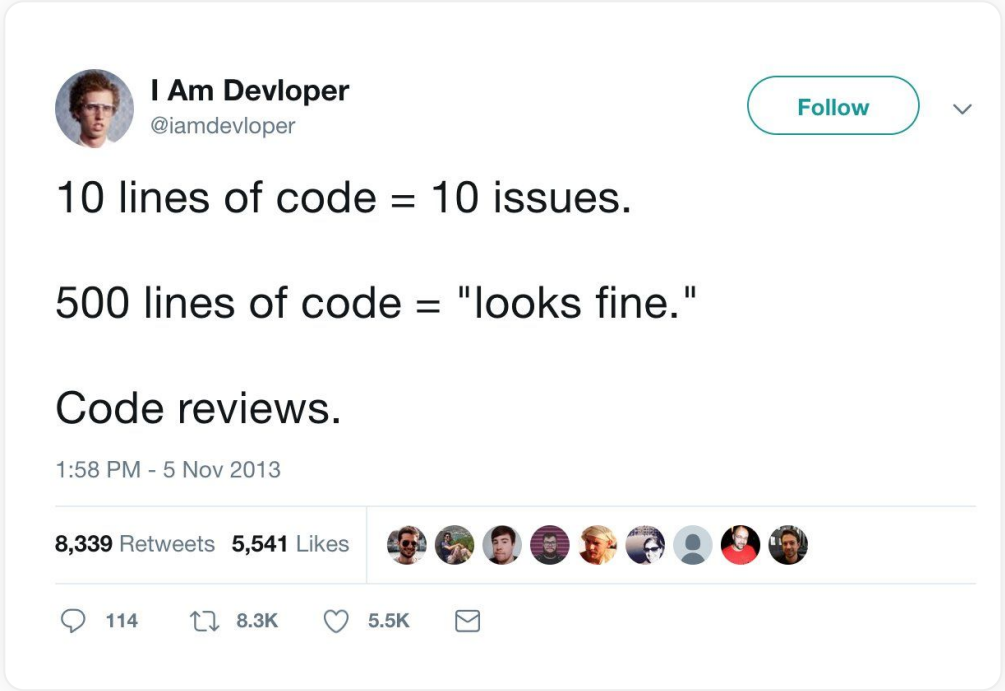


- обнаружение багов
- улучшение качества кода
- поиск наиболее эффективных решений
- поддержание согласованности внутри проекта
- формирование чувства взаимной ответственности

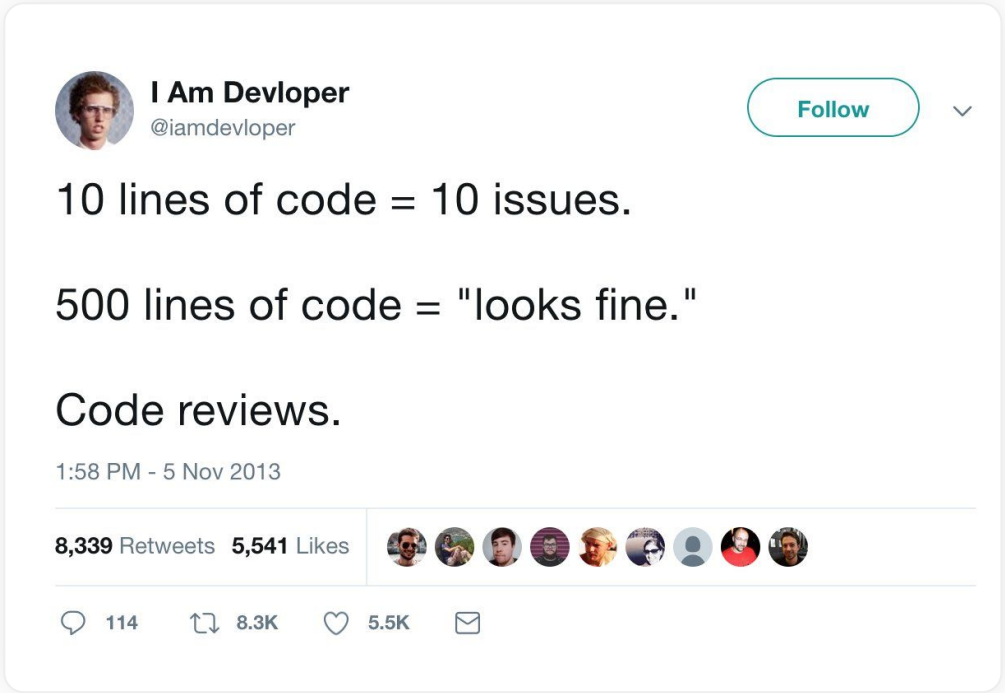


- обнаружение багов
- улучшение качества кода
- поиск наиболее эффективных решений
- поддержание согласованности внутри проекта
- формирование чувства взаимной ответственности


I Am Devloper
@iamdevloper
Follow
10 lines of code = 10 issues.
500 lines of code = "looks fine."
Code reviews.
1:58 PM - 5 Nov 2013
8,339 Retweets 5,541 Likes
114
8.3K
5.5K



- “чужой код”
- “требование плюсиков”
- длинно и долго
- сложно и утомляет
- конфликты убеждений и холивары



- “чужой код”
- “требование плюсиков”
- длинно и долго
- сложно и утомляет
- конфликты убеждений и холивары

Итог: избегание

“Что такое хорошо?”

Регламент

Ответственность автора

Ответственность ревьюера

Договорённости

“Что такое хорошо?”

Регламент

Ответственность автора

Ответственность ревьюера

- что такое готовый к релизу “продукт/фича/изменение/код”
- что улучшает, а что ухудшает код и продукт

“Что такое хорошо?”

Регламент

Ответственность автора

Ответственность ревьюера

- наблюдаемость изменений
- (ревью бывает не только для улучшения, но и для ознакомления)
- 1-ая обратная связь за 0-20 мин
- качественное ревью за ~7-14 минут

“Что такое хорошо?”

Регламент

Ответственность автора

Ответственность ревьюера

- самопроверка
- тестирование
- оформление по принятым правилам
- описание объясняет что и зачем сделано
- читабельный код

“Что такое хорошо?”

Регламент

Ответственность автора

Ответственность ревьюера

– верифицировать “хорошо”

***At Google, we optimize for the speed at which a team of developers can produce a product together**, as opposed to optimizing for the speed at which an individual developer can write code. The speed of individual development is important, it's just not as important as the velocity of the entire team.*

https://google.github.io/eng-practices/review/

Micro-Patch

- минимизация нагрузки на ревьюера
- упрощение чтения и восприятия
- контроль качества решения
- контроль соответствия

Micro-Patch

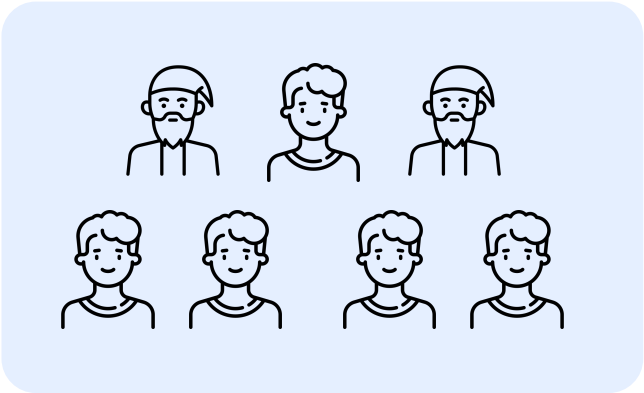
содержит 1 логическое изменение

ограничен по размеру

- soft limit: `50`
- hard limit: `150`
- молько удаление: `350`

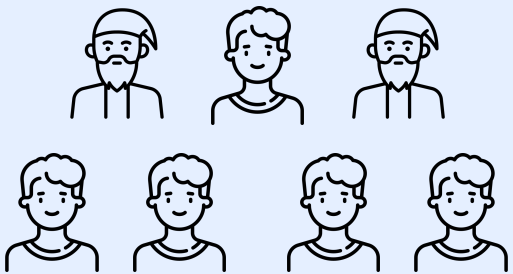
(только код, без учёта тестов)

- **минимизация нагрузки на ревьюера**
- **упрощение чтения и восприятия**
- **контроль качества решения**
- **контроль соответствия**



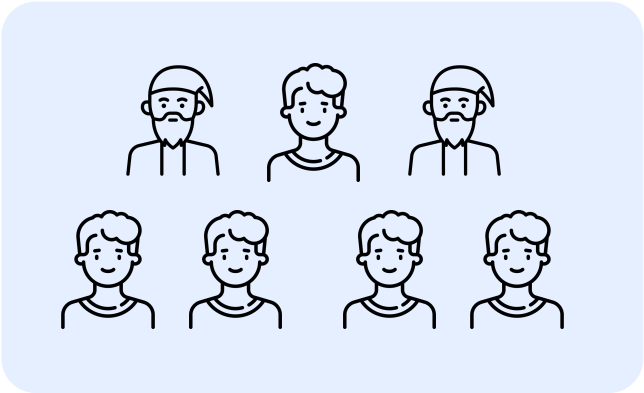
Reviewers (2+ approves)

- Начинающий: обучение/вовлечение
- Эксперт: компетентность
- Сторонний взгляд



Команда

- до 7 человек
- 1 ключевая зона ответственности
- разработчик состоит ровно в 1 команде



Расширение экспертизы

- ротация ревьюеров
- приветствие добровольцев

- Все договорённости должны быть зафиксированы!
- Должна быть договорённость о процессе изменения договорённостей!
- Проблемы “до ревью” не должны решаться “на ревью”!

5 developers enjoy code review

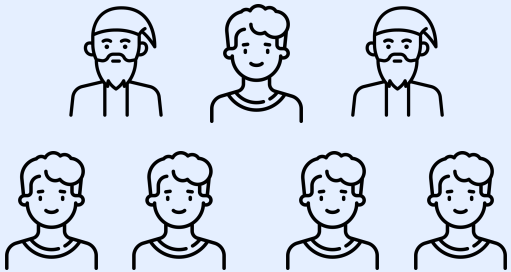

4 of 5 developers enjoy
code review

Оси ревью

- продуктовая / сторонний взгляд
- архитектурная / организация домена
- техническая / поиск ошибок и улучшений
- культурная / контроль оформления кода

Многогранность оценки

- у каждого разработчика свои взгляды
- у каждого разработчика свои приоритеты
- невозможно учесть всё
- невозможно угодить всем



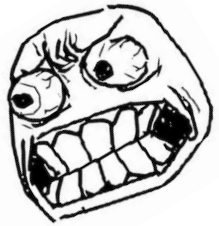
Собрать минимальное Code Review несложно

- работа требует:
  - ответственности
  - соглашений и регламента
- заимствуйте практики лидеров индустрии
- большие патчи никто не смотрит

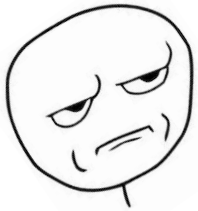
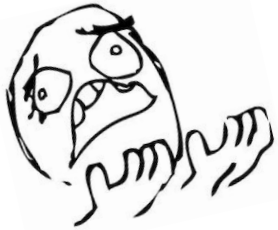
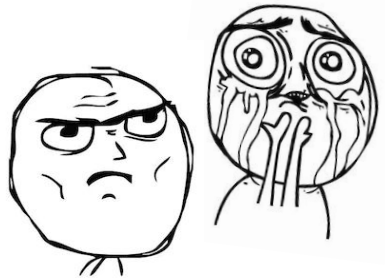
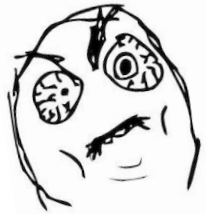
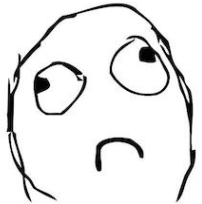
Code Review — столкновение разных мировоззрений и взглядов

Сложность — выстроить эффективное взаимодействие

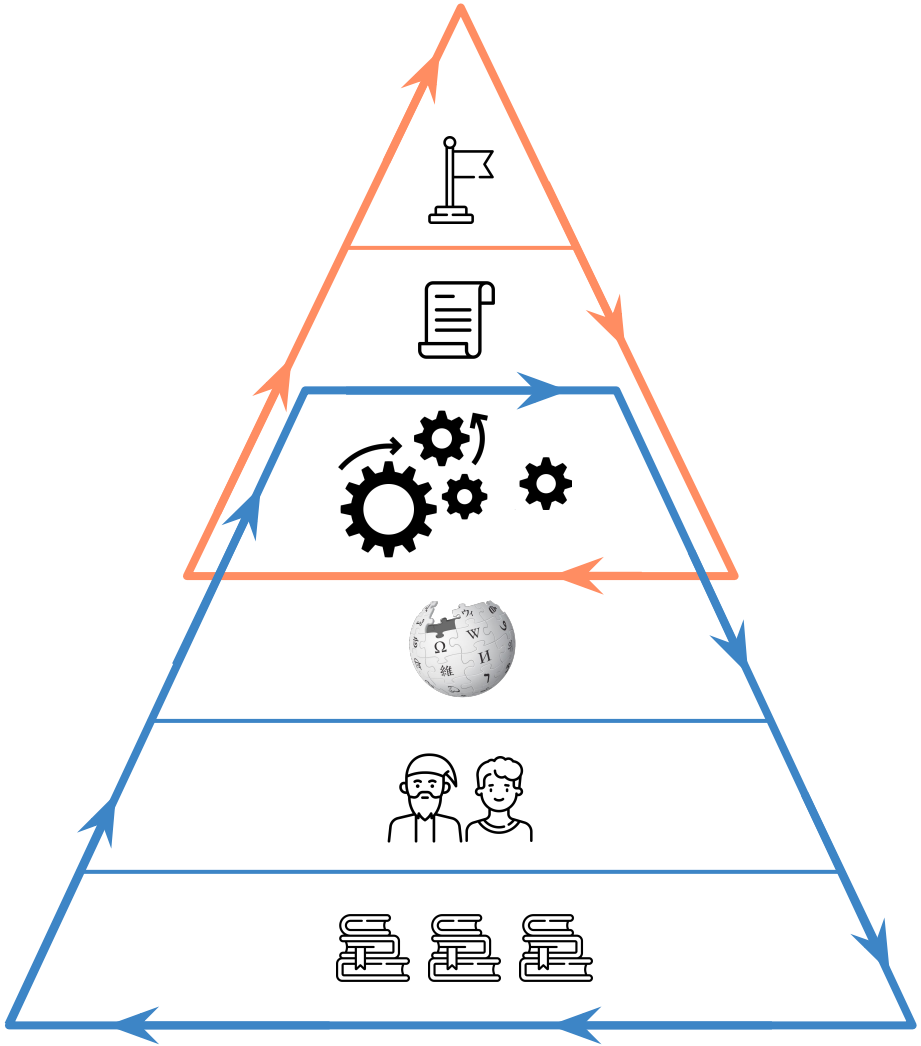


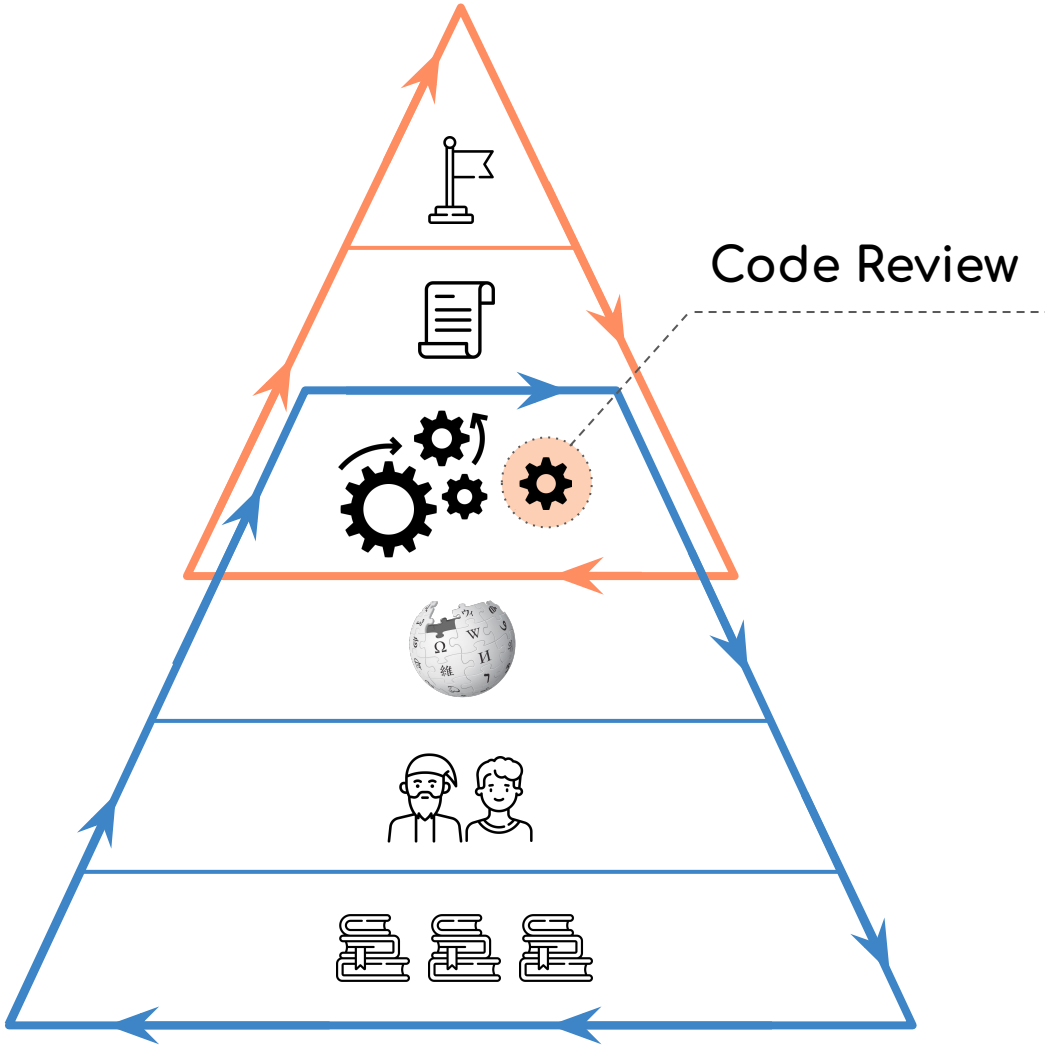
Code Review

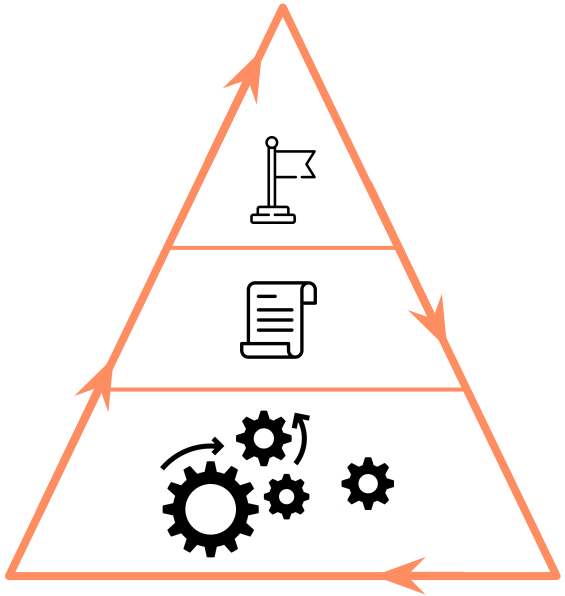
“Процесс командного производства надежного поддерживаемого кода”



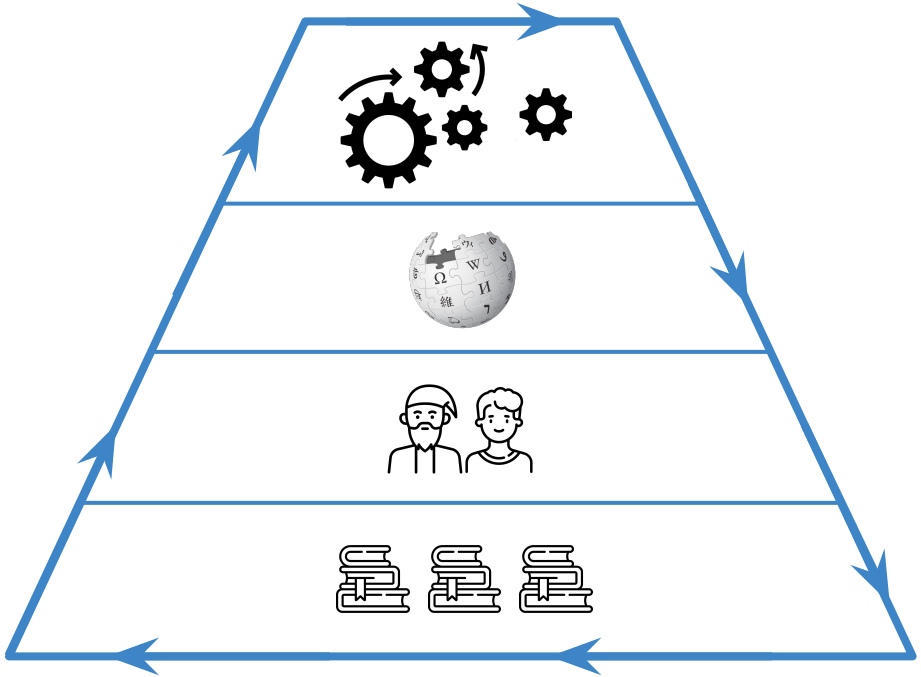
- Цель
- Результат
- Процессы
- Соглашения и документы
- Команда
- Знания и понимание



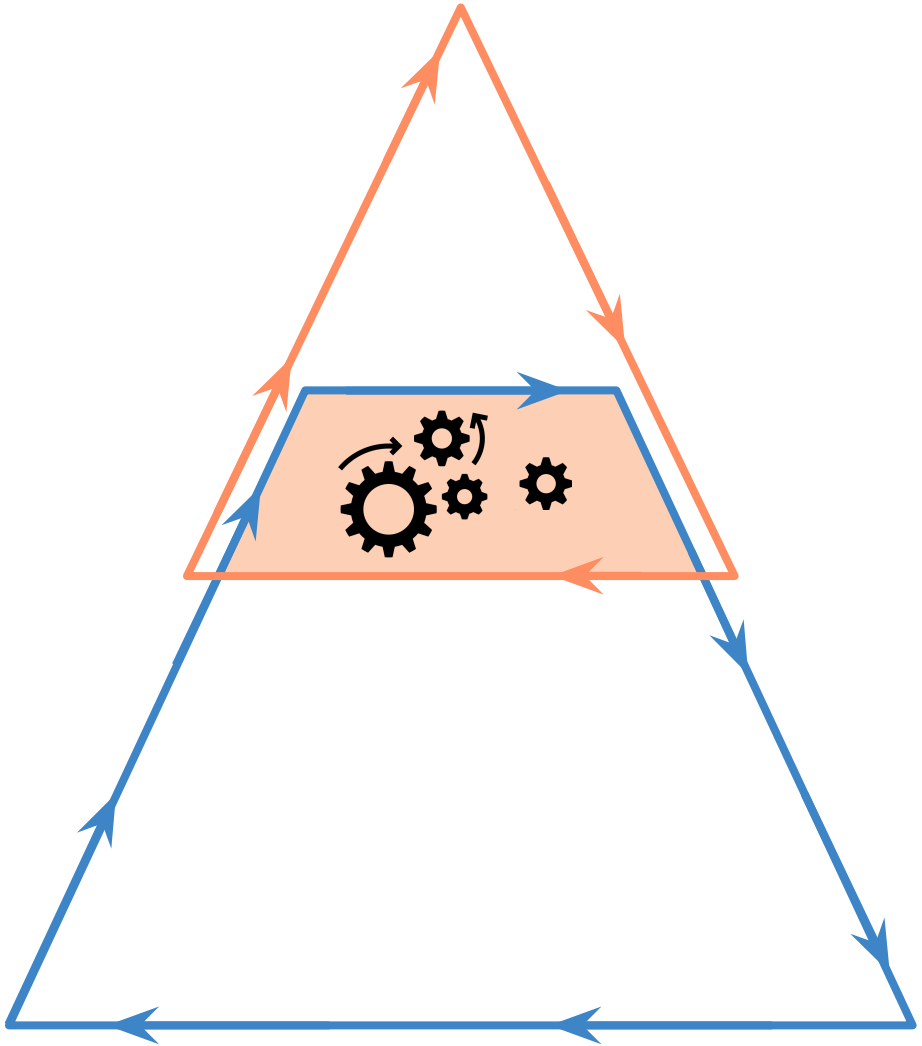
- Цель
- Результат
- Процессы
- Соглашения и документы
- Команда
- Знания и понимание



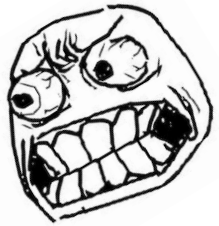
- Цель
- Результат
- Процессы



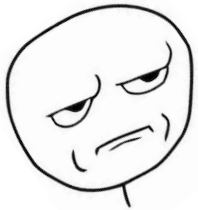
- Соглашения и документы
- Команда
- Знания и понимание

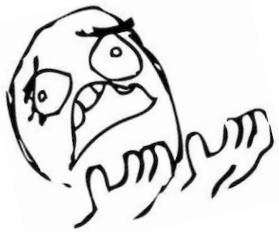


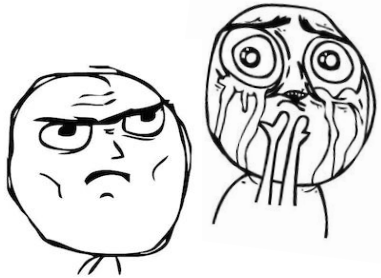
Мой подход к разработке

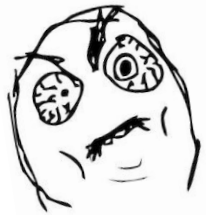


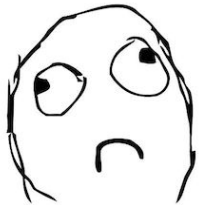
Команда: взаимопонимание

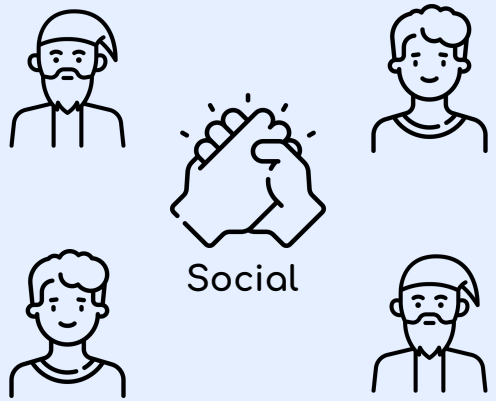












- совместная работа и общий результат
- открытость и принятие друг друга
- уважение и конструктивность
- общий язык

Проблемы начинающих

- слабые решения и частые ошибки

  критика, регулярное “прилетание по шапке”

- нехватка знаний и опыта

  непонятно, как начать делать правильно

- большее время разработки

  чувства одиночества и неуверенности, страх спрашивать

Проблемы экспертов

- достаточная экспертиза

  консерватизм и проблема слышать

- сильные решения “на автомате“

  проблема объяснить, как воспроизвести сильное решение

- большой объём работы и ответственности

  усталость, директивность, скатывание в “лучше сам“

Открытость взглядов:

- желание делиться знаниями и прислушиваться
- поиск других взглядов и взаимное обогащение
- готовность узнать новое о вроде бы понятных вещах

Социальная открытость:

- предлагать и предоставлять помощь
- уметь просить и принимать помощь

“Почему у нас столько асинхронного кода?”

Конструктивный диалог —
это обмен аргументами,
обоснованиями и контекстом.

```
% grep -roh async . | wc -w
  5829

% grep -roh await . | wc -w
  7350
```

“Почему у нас столько асинхронного кода?”

Конструктивный диалог —
это обмен аргументами,
обоснованиями и контекстом.

“Почему у нас столько асинхронного кода?”

Аргументы

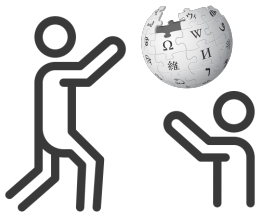
асинхронный код

- медленнее синхронного
- сложнее качественно написать
- запутанней работа с состоянием
- дороже поддерживать и развивать

Обоснования и контекст

наши задачи

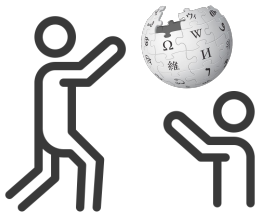
- преимущественно диспетчеризация I/O
- объём CPU-bound задач минимален
- существенно экономит затраты компании на инфраструктуру



Создавайте свой **собственный язык** для общения и взаимопонимания

- термины
- определения
- соглашения
- и всё, что поможет одинаково трактовать используемые слова и выражения

Model — это ...



Создавайте свой **собственный язык** для общения и взаимопонимания

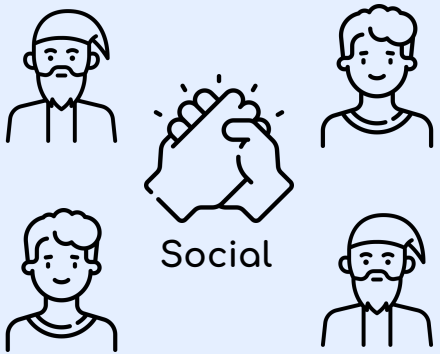
- термины
- определения
- соглашения
- и всё, что поможет одинаково трактовать используемые слова и выражения

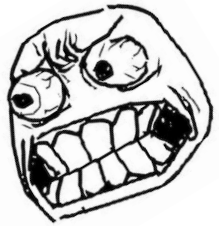
Model — это ...

- ORM (SQL Alchemy)
- Pydantic
- DTO
- Domain Model
- ...

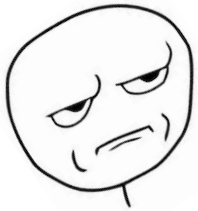
- работайте совместно
- помните, что результат общий, а не личный
- принимайте друг друга

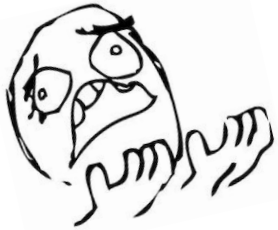
- старайтесь быть открытыми
- уважайте коллег и конструктивность общения
- формулируйте и говорите на языке команды

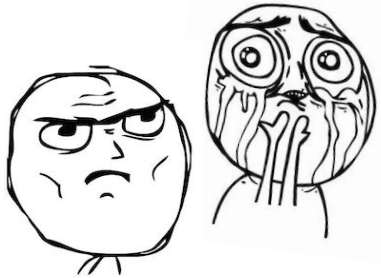


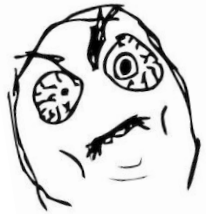


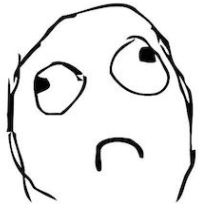
Знания и понимание

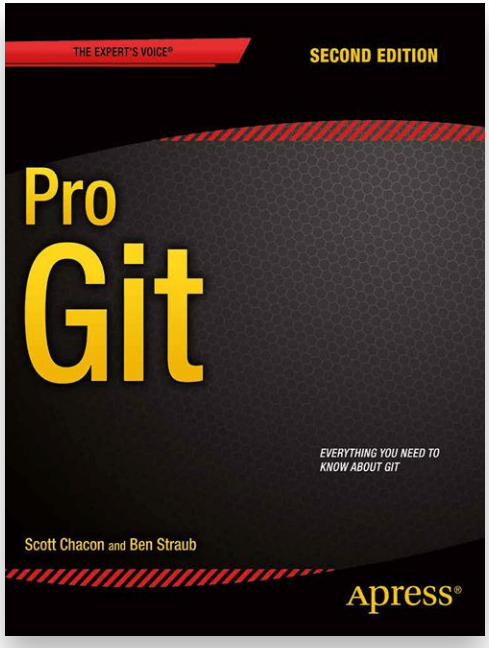


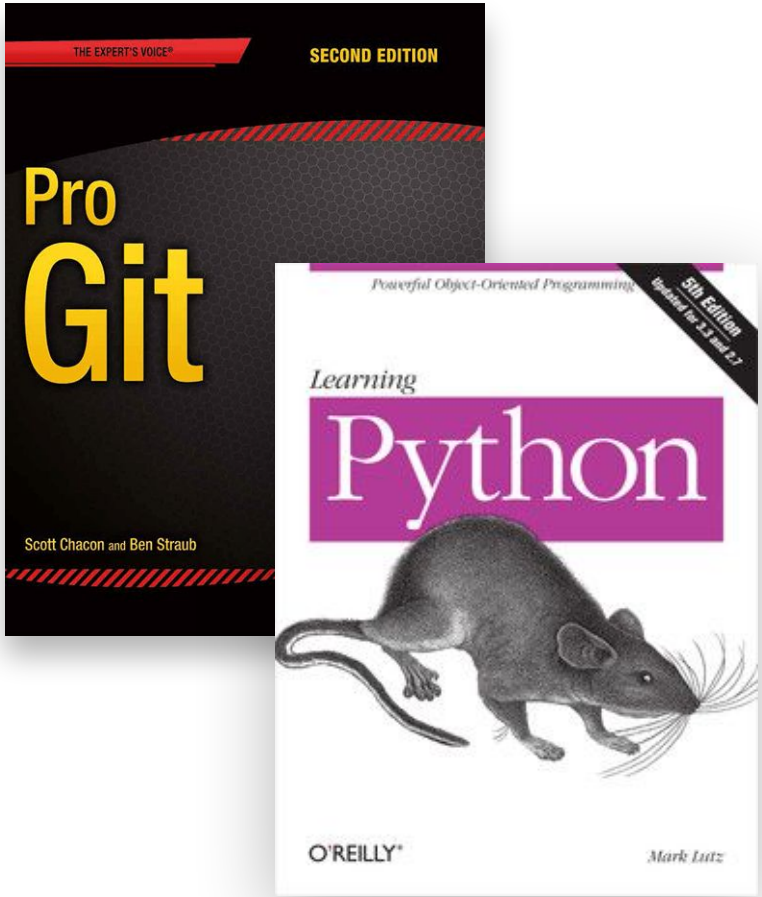


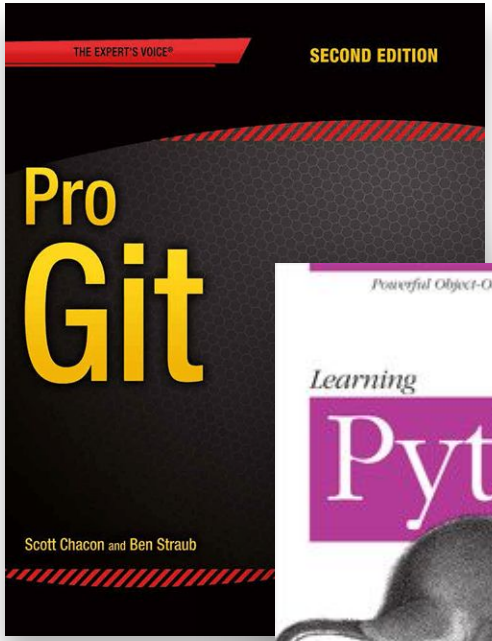


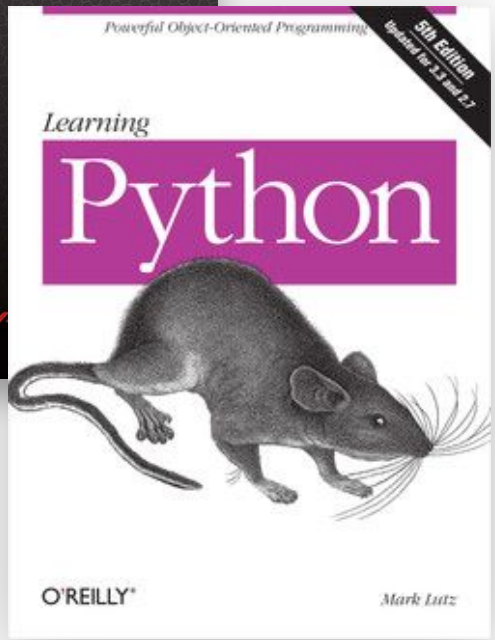


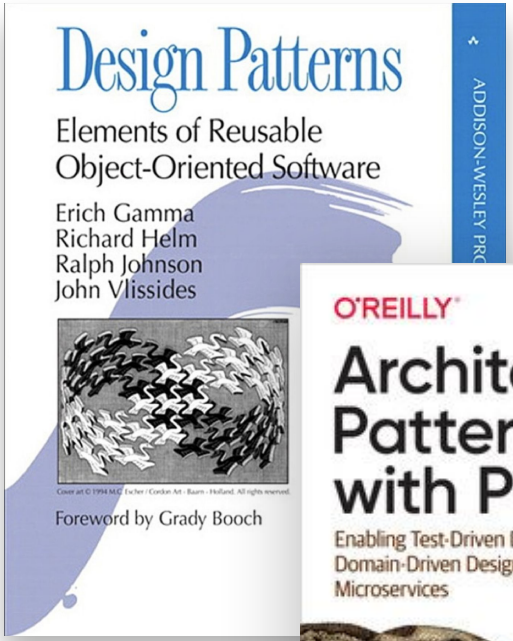


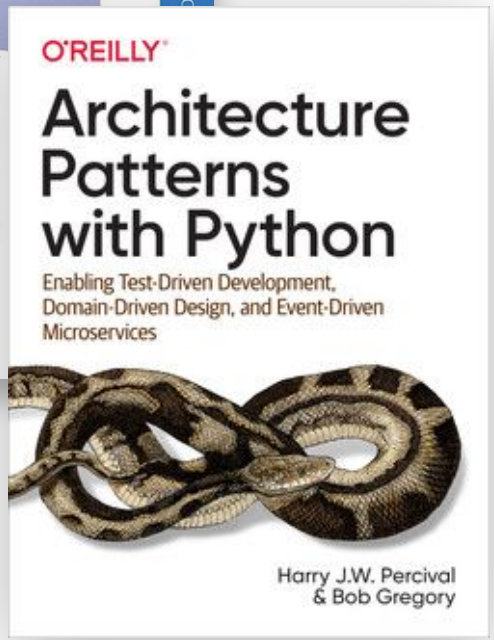

THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®


THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
Scott Chacon and Ben Straub
Powerful Object-Oriented Programming
5th Edition
Updated for 3.3 and 2.7
Learning
Python
O'REILLY®
Mark Lutz


THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
Scott Chacon and Ben Straub


Powerful Object-Oriented Programming
5th Edition
Learning
Python
O'REILLY
Mark Lutz


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY

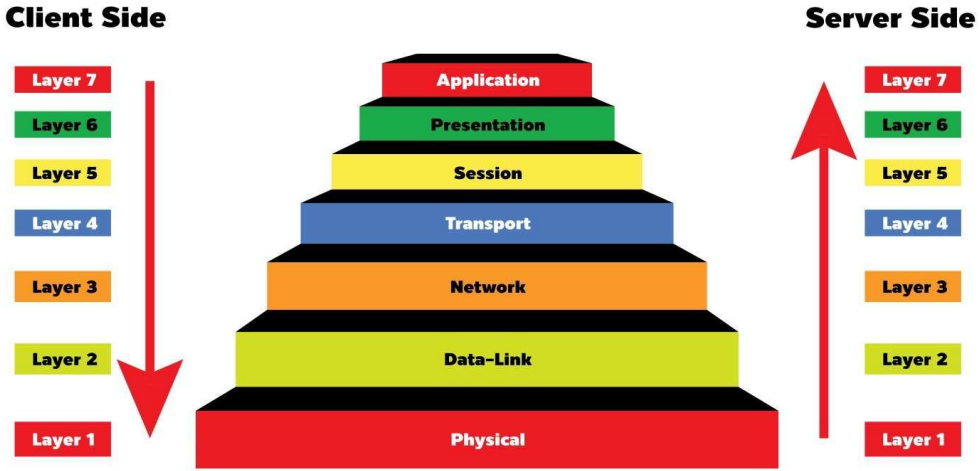

O'REILLY
Architecture
Patterns
with Python
Enabling Test-Driven Development,
Domain-Driven Design, and Event-Driven
Microservices
Harry J.W. Percival
& Bob Gregory

- PEP 20 – The Zen of Python
- KISS: Keep it simple, stupid
- Философия Unix
- DRY: Don’t repeat yourself
- YAGNI: You aren't gonna need it
- Worse is better
- BDUF: Big Design Up Front
- S.O.L.I.D

“Код не должен удивлять
и иметь побочных эффектов”

“Преждевременная оптимизация —
корень всех зол”

OSI MODEL



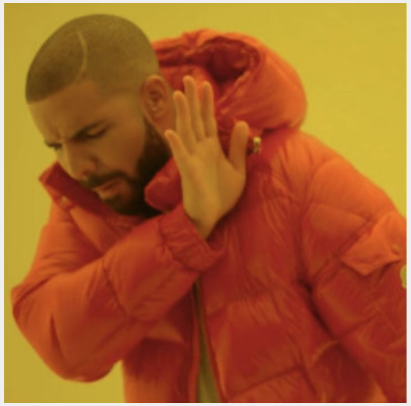
SSL TLS

gRPC

protobuf
Protocol Buffers

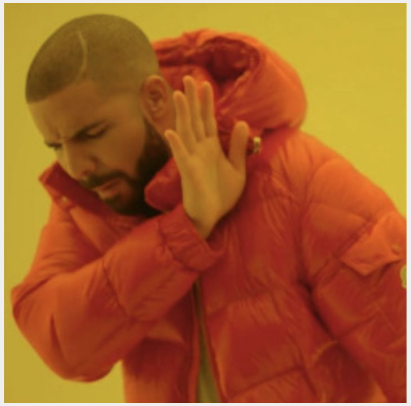
http://


RabbitMQ
envoy
Pydantic
PostgreSQL
redis
django
LINUX
FastAPI
ORY
MySQL
kafka
docker
kubernetes
HAPROXY



```
found = False

for item in resources:
    if item.is_available():
        found = True
        break

if not found:
    raise NoItemAvailable()
```

```
for item in resources:
    if item.is_available():
        break
else:
    raise NoItemAvailable()
```



```python
def func(**kwargs):
    if "special_key" not in kwargs:
        kwargs["special_key"] = 42

    another_func(kwargs)
```

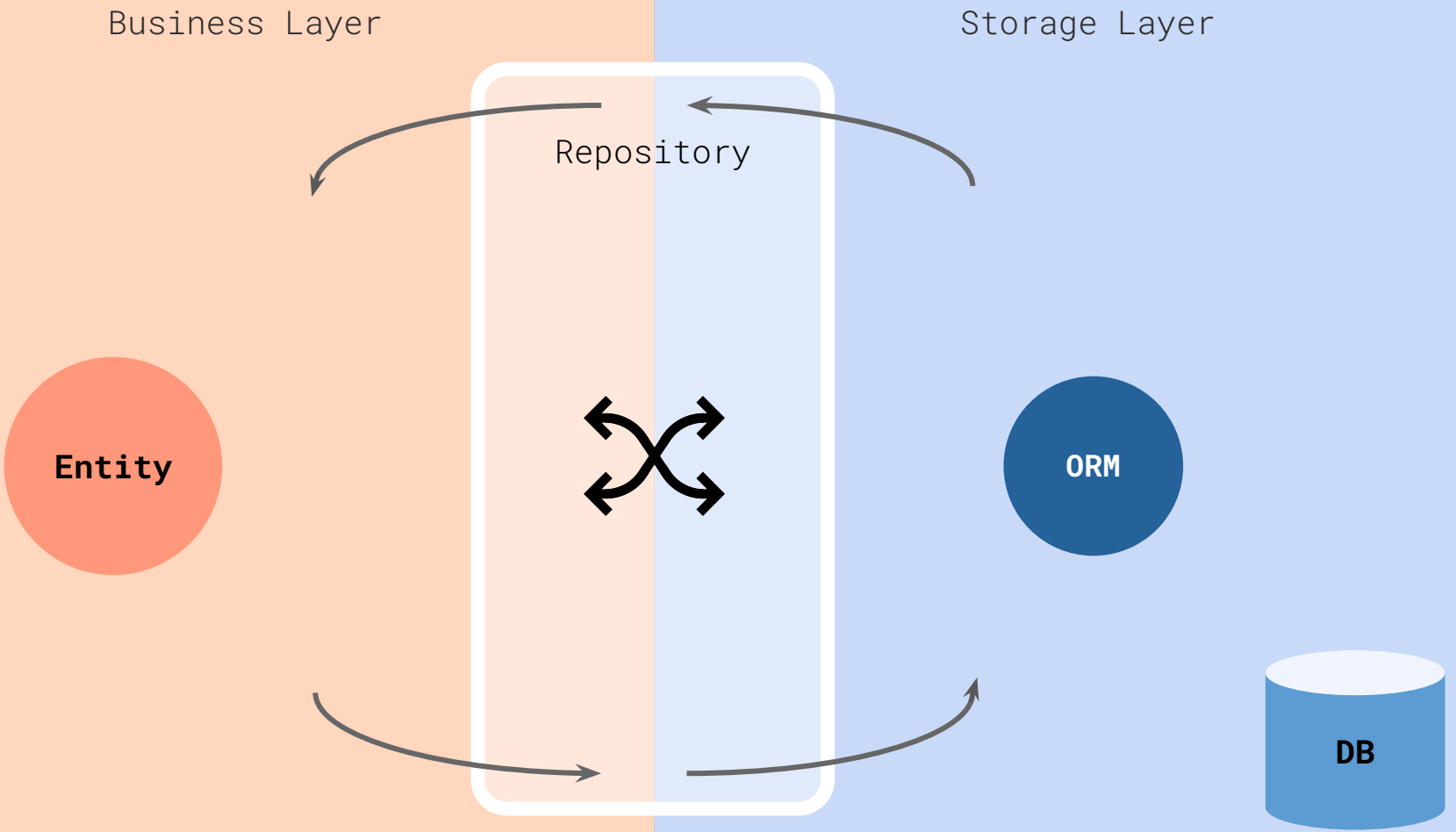
```python
def func(**kwargs):
    kwargs.setdefault("special_key", 42)
    another_func(kwargs)
```

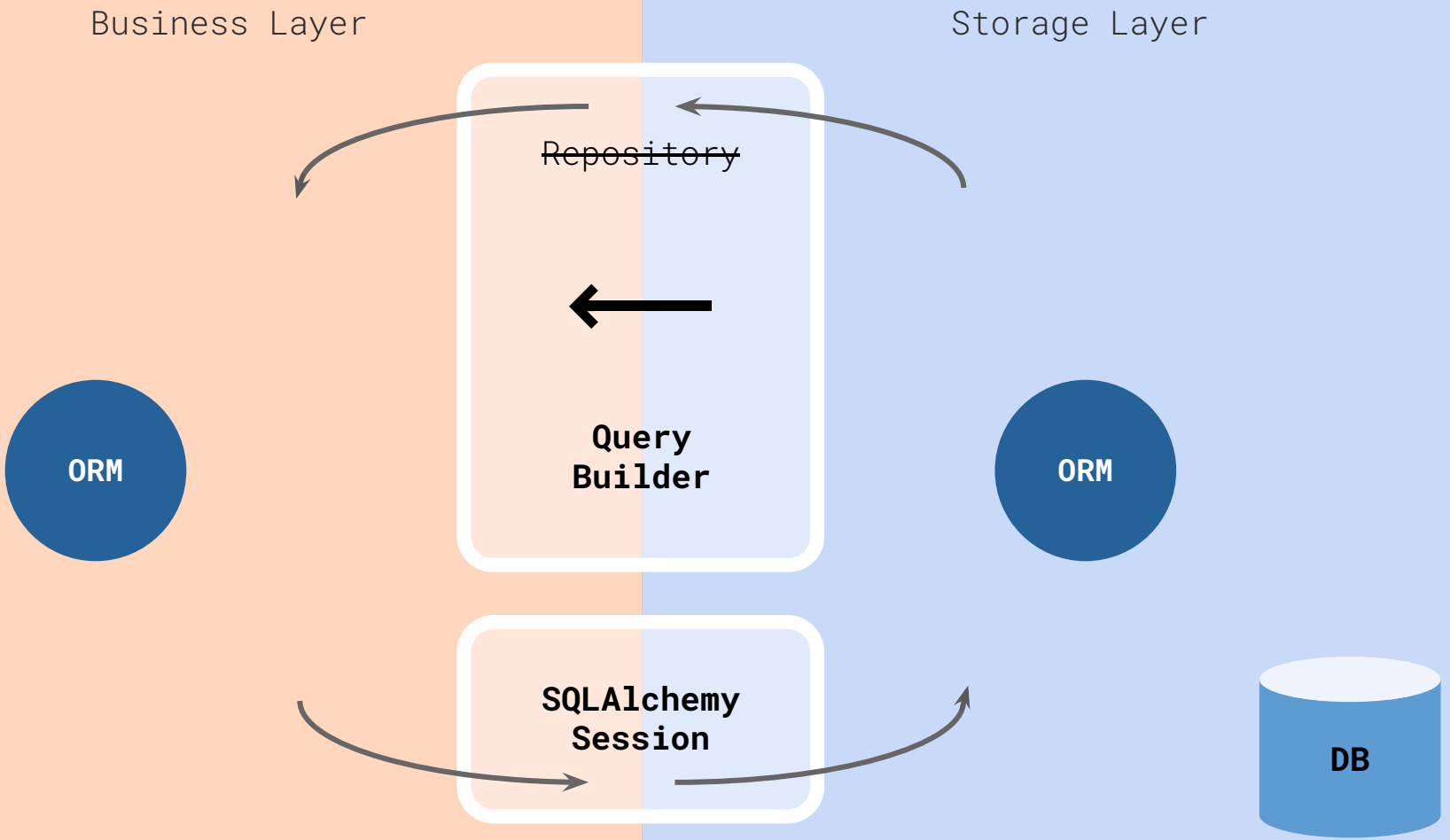
```
 def select_outdated(items: Iterable, max_delay:
float):
     for item in items:
         delay = time.now() - item.timestamp
         if delay > max_delay:
             yield item
```

```
class Entity:

    def _factory(self):
        return random.randint(0, 100)

    @functools.cached_property
    def field(self):
        return self._factory()

    # field = functools.cached_property(_factory)
```

- “__getattribute__” vs “__getattr__”
- property
- metaclasses
- pickle
- concurrency

https://github.com/satwikkansal/wtfpython


Business Layer
Storage Layer
Repository
Entity
ORM
DB


Business Layer
Storage Layer
~~Repository~~
Query Builder
ORM
ORM
SQLAlchemy Session
DB

Pydantic

is the most widely used

data ***validation***

library for Python

```python
class IntModel(pydantic.BaseModel):
    number: int



IntModel(number=42)
# IntModel(number=42)

IntModel(number=None)
# ValidationError: 1 validation error for IntModel
# number
#   none is not an allowed value
(type=type_error.none.not_allowed)
```

Pydantic

is the most widely used

data ***validation,***

conversion

library for Python

```python
class IntModel(pydantic.BaseModel):
    number: int



IntModel(number="42")
# IntModel(number=42)
```

Pydantic

is the most widely used

data ***validation,***

conversion,

casting

library for Python

```python
class IntModel(pydantic.BaseModel):
    number: int



IntModel(number=42.53)
# IntModel(number=42)

IntModel(number=False)
# IntModel(number=0)
```

Pydantic

is the most widely used

data ***validation,***

conversion,

casting

library for Python

```python
class StrictIntModel(pydantic.BaseModel):
    integer: pydantic.StrictInt



StrictIntModel(integer=42)
# StrictIntModel(integer=42)

StrictIntModel(integer=None)
# ValidationError: 1 validation error for StrictIntModel
# integer
#   none is not an allowed value
type=type_error.none.not_allowed)
```

Pydantic

is the most widely used

data ***validation,***

conversion,

casting

library for Python

```python
class StrictIntModel(pydantic.BaseModel):
    integer: pydantic.StrictInt



StrictIntModel(integer="42")
# ValidationError: 1 validation error for StrictIntModel
# integer
#   value is not a valid integer (type=type_error.integer)

StrictIntModel(integer=42.53)
# ValidationError: 1 validation error for StrictIntModel
# integer
#   value is not a valid integer (type=type_error.integer)

StrictIntModel(integer=False)
# ValidationError: 1 validation error for StrictIntModel
# integer
#   value is not a valid integer (type=type_error.integer)
```

Pydantic

is the most widely used

data ***validation,***

conversion,

casting

library for Python

```python
class DateModel(pydantic.BaseModel):
    timestamp: datetime.date



DateModel(timestamp=datetime.date.today())
# DateModel(timestamp=datetime.date(1970, 1, 1))

DateModel(timestamp="hello world!")
# ValidationError: 1 validation error for DateModel
# timestamp
#   invalid date format (type=value_error.date)
```

Pydantic

is the most widely used

data ***validation,***

conversion,

casting

library for Python

```python
class DateModel(pydantic.BaseModel):
    timestamp: datetime.date



DateModel(timestamp=str(datetime.date.today()))
# DateModel(timestamp=datetime.date(2023, 11, 4))

DateModel(timestamp=datetime.date.today().isoformat())
# DateModel(timestamp=datetime.date(2023, 11, 4))

DateModel(timestamp=datetime.datetime.now())
# DateModel(timestamp=datetime.date(2023, 11, 4))
```

Pydantic

is the most widely used

*data **validation,***

conversion,

casting

library for Python

```python
class DateModel(pydantic.BaseModel):
    timestamp: datetime.date



DateModel(timestamp=2000.05)
# DateModel(timestamp=datetime.date(1970, 1, 1))

DateModel(timestamp=False)
# DateModel(timestamp=datetime.date(1970, 1, 1))

DateModel(timestamp=object())
# ValidationError: 1 validation error for DateModel
# timestamp
#   invalid type; expected date, string, bytes, int or
float
 (type=type_error)
```

Pydantic

is the most widely used

data ***PARSING***

library for Python

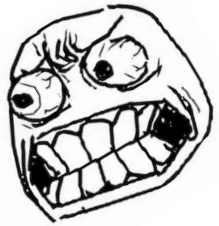
```python
class DictModel(pydantic.BaseModel):
    data: dict[int, str]


DictModel(data={1: "one", 2: "two"})
# DictModel(data={1: 'one', 2: 'two'})

DictModel(data=[[1, "one"], [2, "two"]])
# DictModel(data={1: 'one', 2: 'two'})
```

“Я знаю, что я ничего не знаю”
Сократ

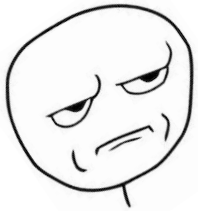
Развивайте:

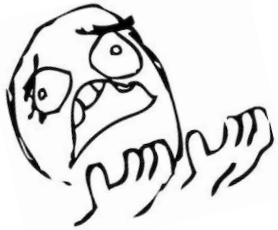
- реальное понимание
- эрудированность:
  - учите матчасть
  - изучайте корневые технологии
  - разбирайтесь в популярных технологиях
- эффективное применение технологий и инструментов

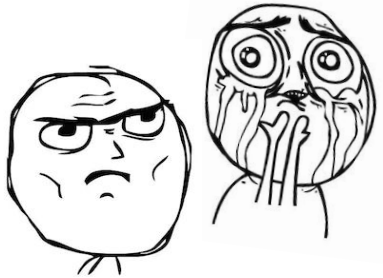
Профессионализм начинается с базовых навыков

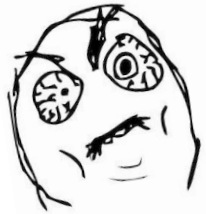


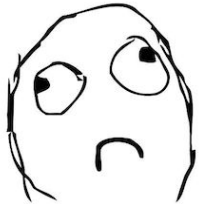
Соглашения и консолидация











Команда

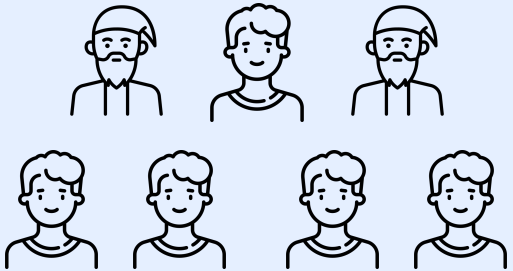
находит взаимопонимание:

- умеет говорить
- и договориться

по настоящему понимает

- что делает,
- с чем работает
- и о чём говорит

Общий знаменатель



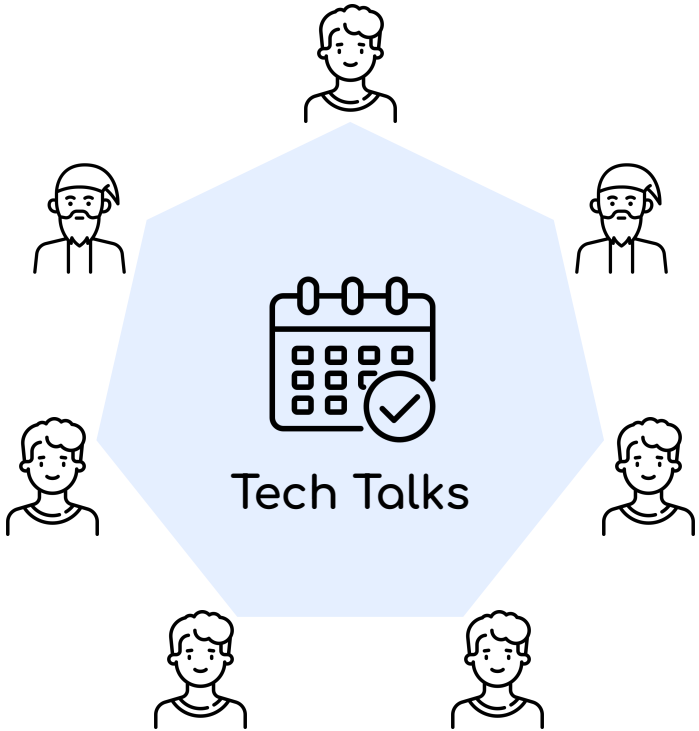
Команда

находит взаимопонимание:

- умеет говорить
- и договориться

по настоящему понимает

- что делает,
- с чем работает
- и о чём говорит



Подвергайте коллективному критическому разбору

- технологии и продукты
- концепции программирования

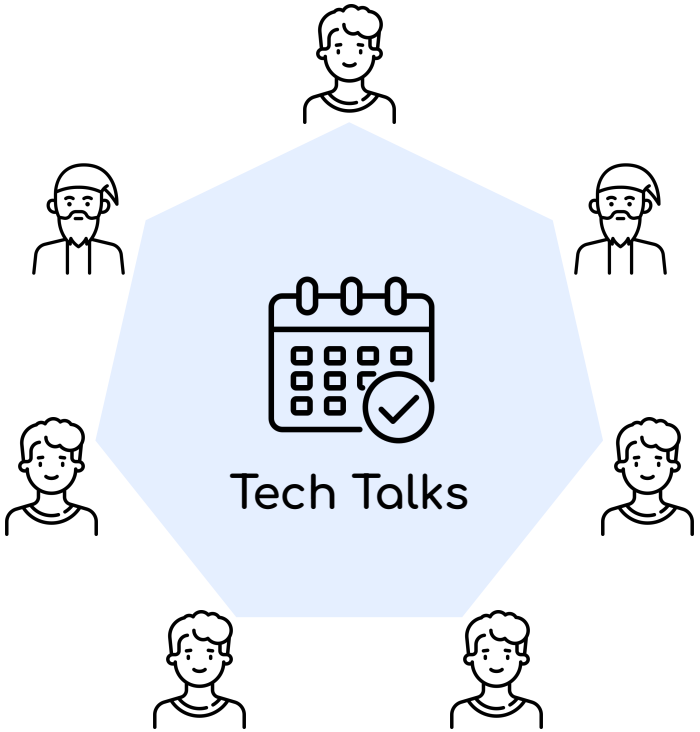
- лучшие и худшие практики
- стандарты и процессы

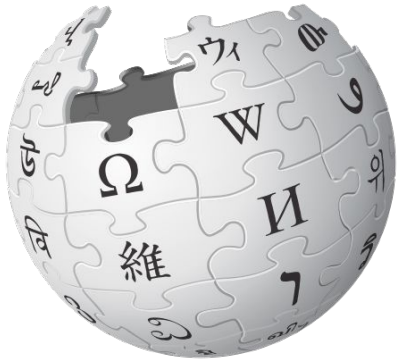
Формируйте собственные договоренности

- подходы и решения
- определения, трактовки и т.д.

- лучшие и худшие практики
- стандарты и процессы

Всё это должно решаться **до ревью,** а не **на ревью!**





Knowledge Base

Командный репозиторий
идей, знаний, понимания
и общей культуры.

(исходники вашего будущего кода)

Knowledge Base

Командный репозиторий
идей, знаний, понимания
и общей культуры.

(исходники вашего будущего кода)

Регулярный командный процесс
Knowledge Review

(аналогично процессу
Code Review для кода)

Консолидация:

желание каждого

участвовать и вовлекаться

Мотивация:

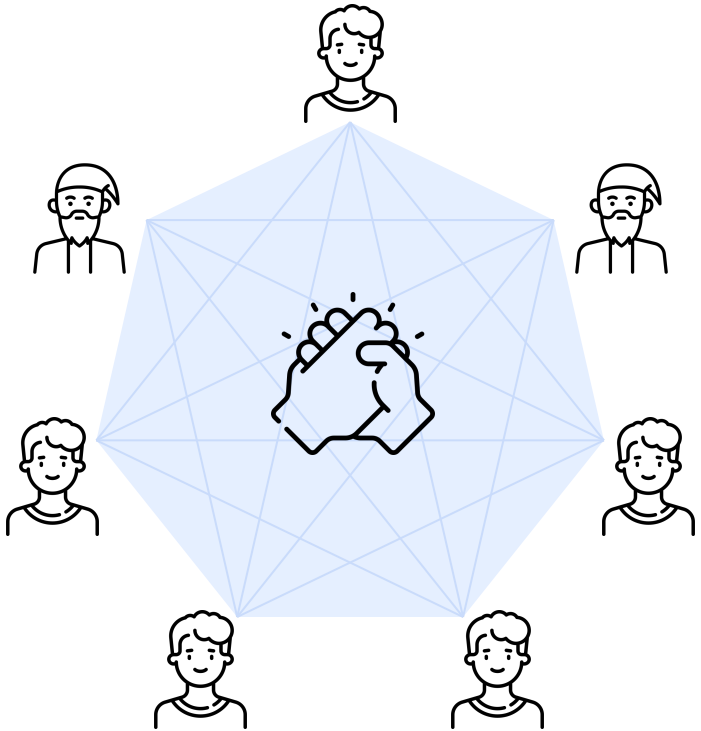
стремление к развитию

и качественному результату

Ответственность

и последовательность

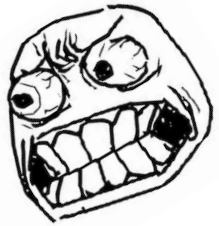
в действиях



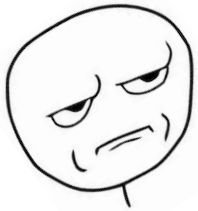
Консолидируйте персональные понимания
в командную культуру

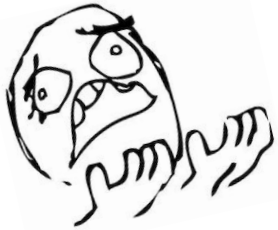
Выстройте командный процесс регулярного
диалога для критического взгляда и переосмысления
окружающих вас идей и технологий

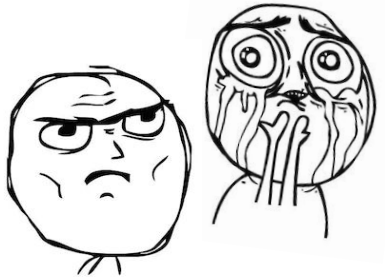
Этот процесс позволит каждому активному
участнику ускоренно расти

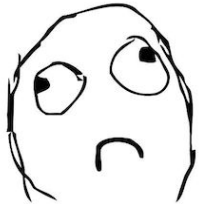


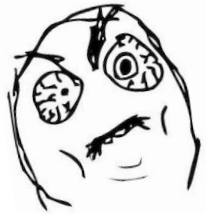
Мой подход к разработке

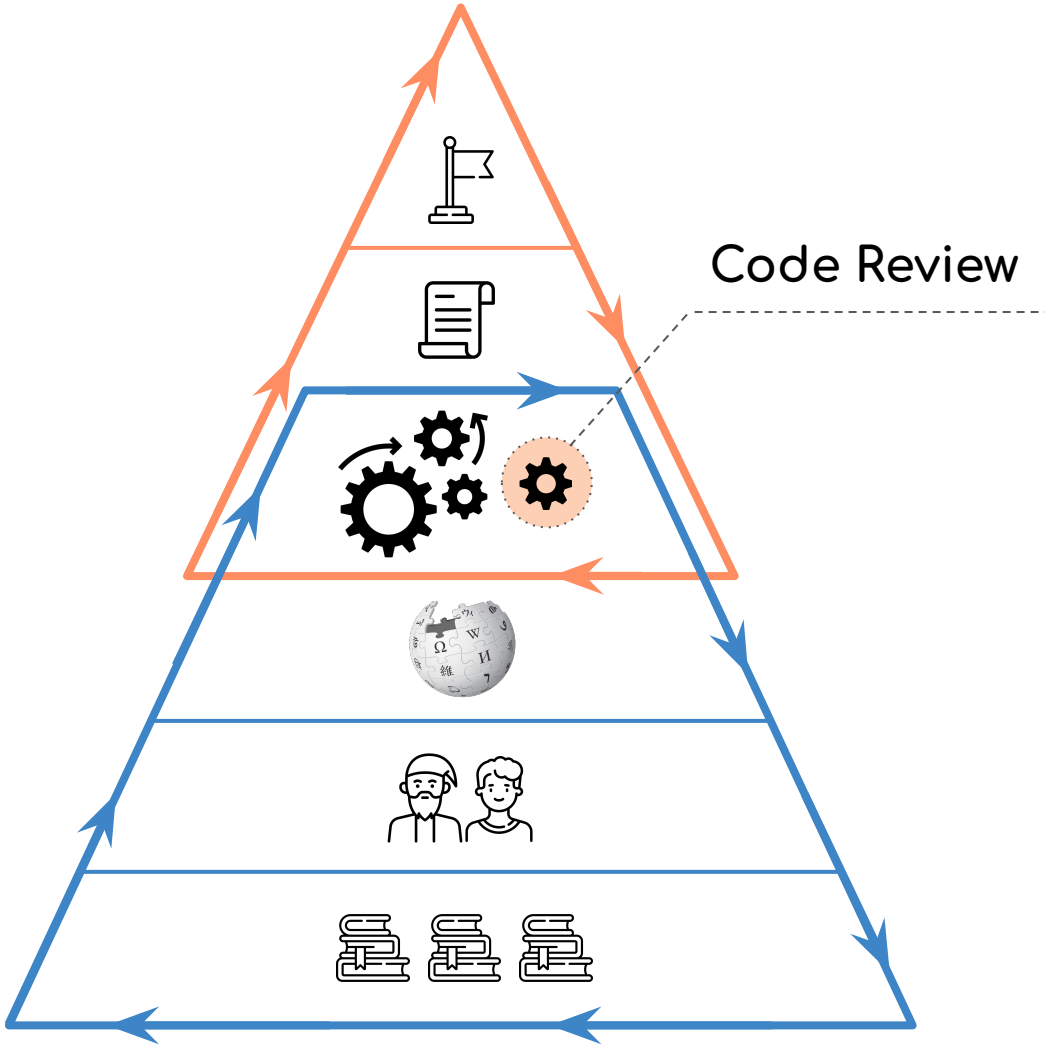












- Цель
- Результат
- Процессы
- Соглашения и документы
- Команда
- Знания и понимание

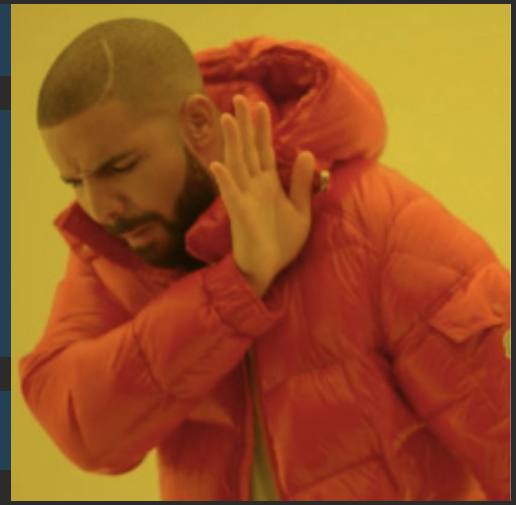
*Требуется **сущность**, отвечающая за управление DNS-записями нод,*
которая позволит

- **записать** DNS-запись
- **очистить** DNS-запись
- **выдать список** DNS-записей
- **вычислять разные представления** для имён DNS-записей

```python
...  # some code

class DnsNodeRecordRepository:

    ...  # trivial init

    def _get_node_ip_address(self, node: models.Node) -> ipaddress.IPv4Address:
        ...  # some code

    def make_name(self, node: models.Node) -> NodeName:
        ...  # some code

    def create_node_record(self, node: models.Node) -> str:
        ...  # some code

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone, ip_address, name.zone,
            )

    def list_a_records(self) -> tuple[RecordA]:
        ...  # some code
```

```
...  # some code

class DnsNodeRecordRepository:

    ...  # trivial init

    def _get_node_ip_address(self, node: models.Node) -> ipaddress.IPv4Address:
        ...  # some code

    def make_name(self, node: models.Node) -> NodeName:
        ...  # some code

    def create_node_record(self, node: models.Node) -> str:
        ...  # some code

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone, ip_address, name.zone,
            )

    def list_a_records(self) -> tuple[RecordA]:
        ...  # some code
```

```
...  # some code

class DnsNodeRecordRepository:

    ...  # trivial init

    def _get_node_ip_address(self, node: models.Node) -> ipaddress.IPv4Address:
        ...  # some code

    def make_name(self, node: models.Node) -> NodeName:
        ...  # some code

    def create_node_record(self, node: models.Node) -> str:
        ...  # some code

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone, ip_address, name.zone,
            )

    def list_a_records(self) -> tuple[RecordA]:
        ...  # some code
```



Перед началом работы над задачей **мы не знаем** конечные:

- архитектуру того, что делаем
- дизайн кода и классов
- код реализации

Большой конечный объём

- идей
- решений
- правок

Agile-разработка

Большинство гибких методик нацелены на минимизацию рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели.

Каждая итерация сама по себе выглядит как программный проект в миниатюре и включает все задачи, необходимые для выдачи мини-прироста по функциональности:

- планирование,
- анализ требований,
- проектирование,
- программирование,
- тестирование
- и документирование.

Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации команда выполняет переоценку приоритетов разработки.

Agile-разработка

Большинство гибких методик нацелены на минимизацию рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели.

Каждая итерация сама по себе выглядит как программный проект в миниатюре и включает все задачи, необходимые для выдачи мини-прироста по функциональности:

- планирование,
- анализ требований,
- проектирование,
- программирование,
- тестирование
- и документирование.

Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации команда выполняет переоценку приоритетов разработки.

Методология гибкой разработки как часть программирования

- декомпозировать задачу на маленькие кусочки
- “кусочек” — 1 минимальное логическое изменение
- для каждого “кусочка” чётко определены:
  - цель
  - scope
  - важность
- мы не знаем (даже если *"очень догадываемся"*):
  - как делать каждый отдельный “кусочек”
  - как они соберутся в конечное решение
- работаем с динамичной цепочкой маленьких гипотез
- итеративно разрабатываем каждый “кусочек”

Этапы развития “кусочка”

1 этап — 1 Merge Request

- **очертить дизайн**
  - сигнатура
  - набор сущностей
  - псевдокод процедуры
- **черновая реализация** (избегая тестов)
- **чистовая реализация** (с тестами)

Мой подход к разработке

- молниеносные итерации
- микро-патчи
- качественное ревью за 3-5 мин.
- нерабочий код
- feature-ветки
- реактивные нотификации в канал с mention-ами:
  - до 4 переписок в GitLab
  - (возможно) чат в канале под сообщением
  - (возможно) ad-hoc командный созвон для brainstorm-а и консультаций

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        raise NotImplementedError
```

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        raise NotImplementedError
```

Этап очертить дизайн

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        raise NotImplementedError
```

- паттерн Repository
- почему это вообще Repository?

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        raise NotImplementedError
```

- интерфейс этого репозитория
- типы аргументов и результата
- анти-паттерн “Generic Repository”

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

Этап черновая реализация

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

- тип объекта “name”
- выделяем отдельную задачу на метод “make_name”

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

- “поднимаем на поверхность” техдолг
- формулируем план или задачу для доработки
- приоритезируем в беклоге

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

- по новому смотрим на интерфейс “dns_client”
- обсуждаем и планируем работы

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        self._dns_client.delete_record_a(
            name.without_zone, ip_address, name.zone,
        )
```

думаем о поведении при ошибках

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone , ip_address, name.zone,
            )
        # + logging
```

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone , ip_address, name.zone,
            )
        # + logging
```

Этап чистовая реализация

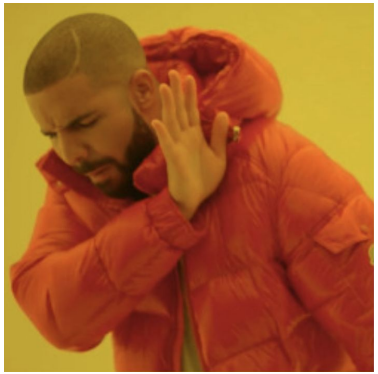
```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone , ip_address, name.zone,
            )
        # + logging
```

автор учитывает все замечания предыдущих этапов

```python
class DnsNodeRecordRepository:

    def create_node_record(self, node: models.Node) -> str:
        raise NotImplementedError

    def delete_node_record(self, node: models.Node) -> None:
        name = self.make_name(node)
        ip_address = self._get_node_ip_address(node)
        with contextlib.suppress(dns_excs.DnsRecordNotFoundError):
            self._dns_client.delete_record_a(
                name.without_zone , ip_address, name.zone,
            )
        # + logging
```

код следует всем требованиям
по оформлению чистового кода
(такие как логирование и прочее)

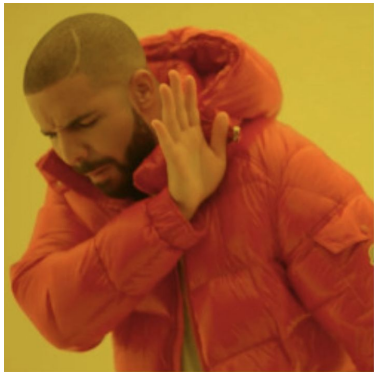
Перед стартом ревью



- общая продуктовая цель
- **якобы** представление технического решения

- общая продуктовая цель
- **гипотеза** технического решения

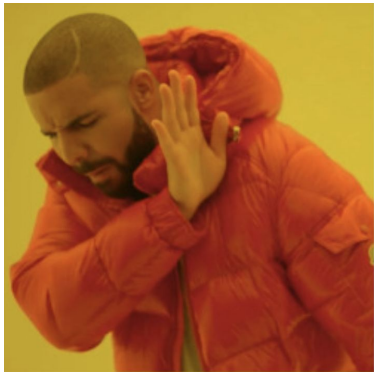
Во время ревью



- **разные взгляды** на техническое решение
- **отстаивание/защита** представленного изменения/кода (как конечного и правильного)

- **разные взгляды** на один из вариантов реализации гипотезы
- **диалог и совместные поиск** лучшего решения
- (легко отказаться от предложенного варианта)
- (легко отказаться от изначальной гипотезы)

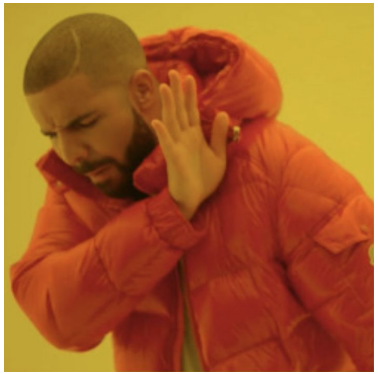
После ревью



- продукт получил **какое-то решение**
- **различное видение** технического решения
- техдолг выявлен и обработан в **ограниченном** объёме

- продукт получил **решение, приближенное к наилучшему**
- **единое командное видение** технического решения
- техдолг выявлен и обработан в **максимальном** объёме

После ревью



повышение одного из следующих **рисков:**

- эта часть кода **замкнётся на первом написавшем**
- этот код **попробует переписать следующий разработчик**, который столкнется с этой частью кода в следующий раз

в следующий раз при работе с этой частью кода **любой участник продолжит развитие ранее сформированного командного видения** по улучшению этого модуля

Декомпозиция задачи
на минимальные логические изменения

Декомпозиция разработки логического изменения
на этапы:

- очертить дизайн
  - сигнатура
  - набор сущностей
  - псевдокод процедуры
- черновая реализация (избегая тестов)
- чистовая реализация (с тестами)

Живое Code Review

- консолидация экспертов
- активный рост экспертизы неопытных (выход из категории начинающих)
- техдолг под надежным контролем
- технический Roadmap

Живое Code Review

- *проектирование и планирование*

  лучшее — враг хорошего

- *реалистичность и реализация*

  контролируемое качество

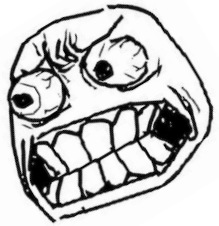
- *общие цели*

  понимание и прозрачность

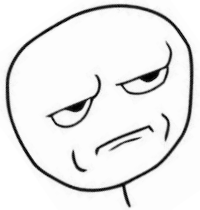
python™

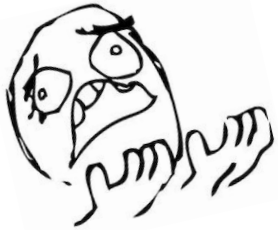
Язык практичного проектирования

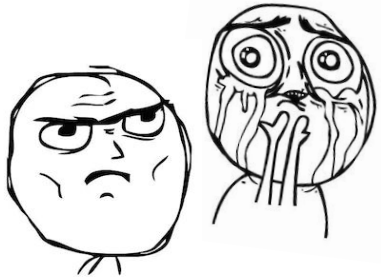
- лаконичный, элегантный, наглядный код
- мощность языка + стандартная библиотека
- хороший дизайн на Python — это 50% реализации

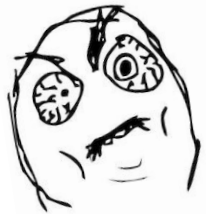


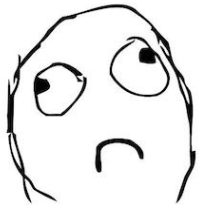
Итоги

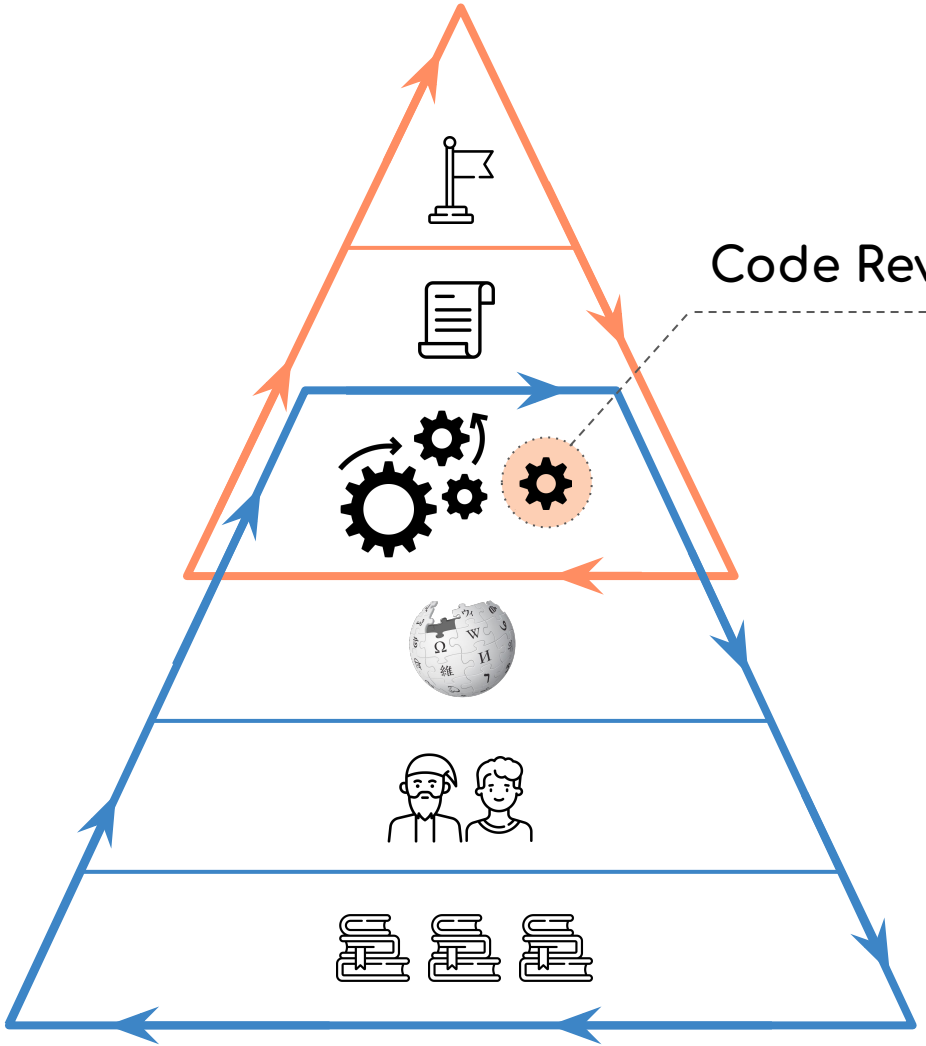












- Цель
- Результат
- Процессы
- Соглашения и документы
- Команда
- Знания и понимание

Собрать минимальное Code Review — **несложно**

Выстроить эффективное взаимодействие — **сложно**

Code Review

- точка самого плотного и **чувствительного** контакта
- **боль,** если команда атомизирована, и каждый отстаивает собственные взгляды

Три столпа эффективности вашего Code Review

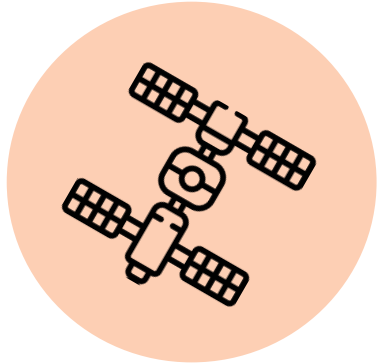
- взаимопонимание
- реальные знания
- консолидация

В Code Review больше **социального,** чем технического

Лучшие решения и лучший код

— это **результат командной работы,**

а не индивидуальное решение



Code Review — это

часть разработки,

а не преграда на её пути

Превратите

“ревью-боль”

в “ревью-развитие”

Code Review – инструмент

роста и взаимопонимания

ТИНЬКОФФ

Code Review

Процесс командного производства
надежного поддерживаемого кода

- постоянное развитие команды
- непрерывная поставка качественных фич
- параллельное техническое развитие

Slides and **BONUS** inside!

Dima
Burmistrov

@pyctrl

dimaburmistrov