Простая и понятная архитектура автотестов для проекта с opensource ядром

Андрей Березин

Приходилось ли вам?

01 Бороться с дублированием кода в автотестах и самих автотестов?

02 Менять логику тестов в зависимости от среды выполнения?

03 Тестировать продукты с общей функциональностью?

Пара слов о команде

- Разрабатываем распределенную СХД Tatlin.Object
- Состав QA команды — **13** инженеров
- С нуля до первого релиза за полгода

Андрей Березин

SDET в команде QA Tatlin.Object

- Более 8 лет в автоматизации
- С питоном работаю около 2 лет
- Раньше писал автоматизацию на C#

Контекст продукта глазами QA

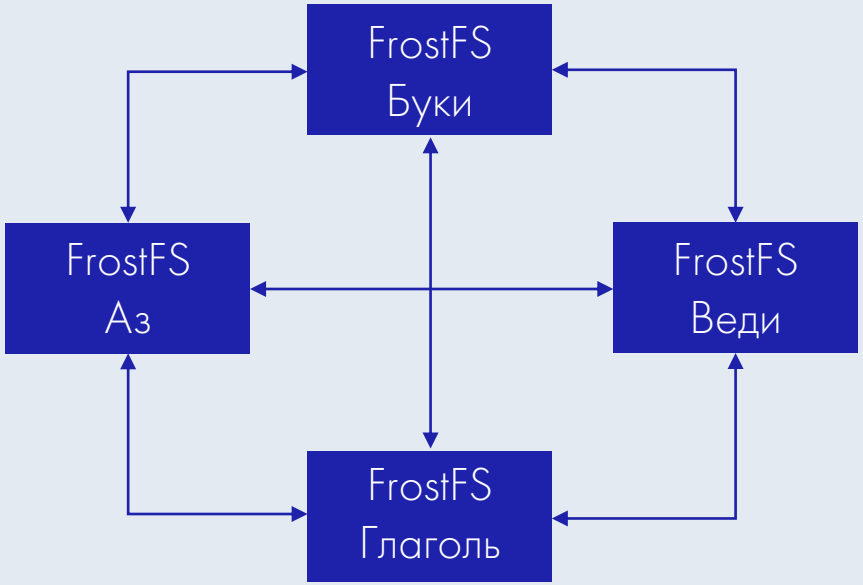
Движок хранения данных FrostFS

- Открытый исходный код
- Распределенный
- Представляет собой набор сервисов
- Поддерживает S3, HTTP, gRPC протоколы

Что такое FrostFS

Базовые возможности

- Загрузить файл в систему
- Скачать файл из системы



Контекст продукта глазами QA

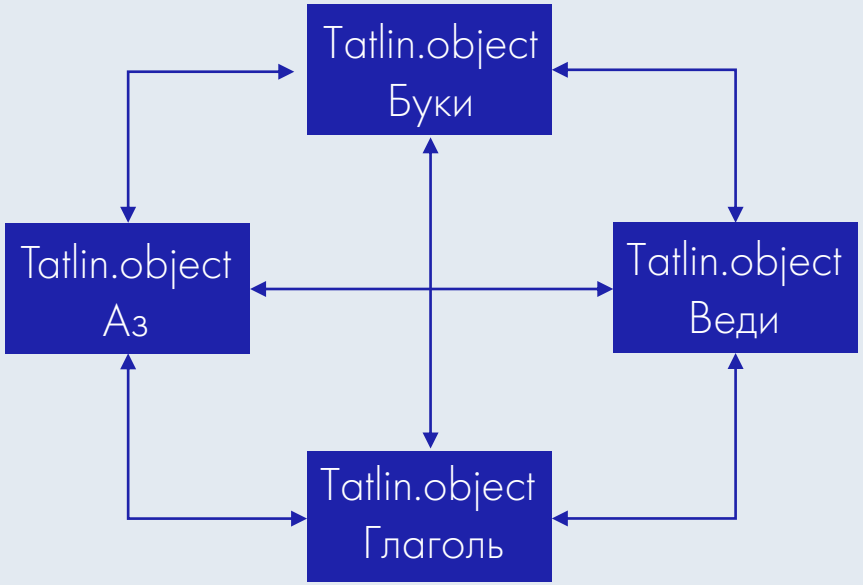
Система хранения данных Tatlin.Object

- Коммерческий продукт
- Использует FrostFS как ядро
- Расширяет FrostFS собственными сервисами

Что такое Tatlin.Object

Базовые возможности

- Загрузить файл в систему
- Скачать файл из системы



В чем уникальность?

Все так или иначе используют Opensource

Надо тестировать свой продукт

Мы и пользователи и сами себе разработчики FrostFS

Необходимо тестировать два отдельных продукта с общим функционалом

Задачи автоматизаторов

Основные

- Писать автотесты на FrostFS в публичном доступе
- Писать автотесты на Tatlin.Object
- Поддерживать запуск тестов на разных окружениях
- Не допускать утечки внутренней информации в публичные репозитории

Дополнительные

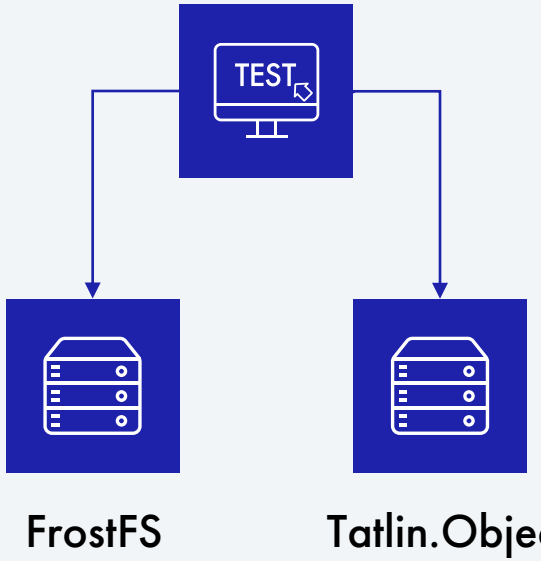
- Минимизировать дублирование
- Сохранить читаемость кода

Пример тестового сценария

01 Загрузить файл

02 Перезагрузить сервер

03 Скачать файл



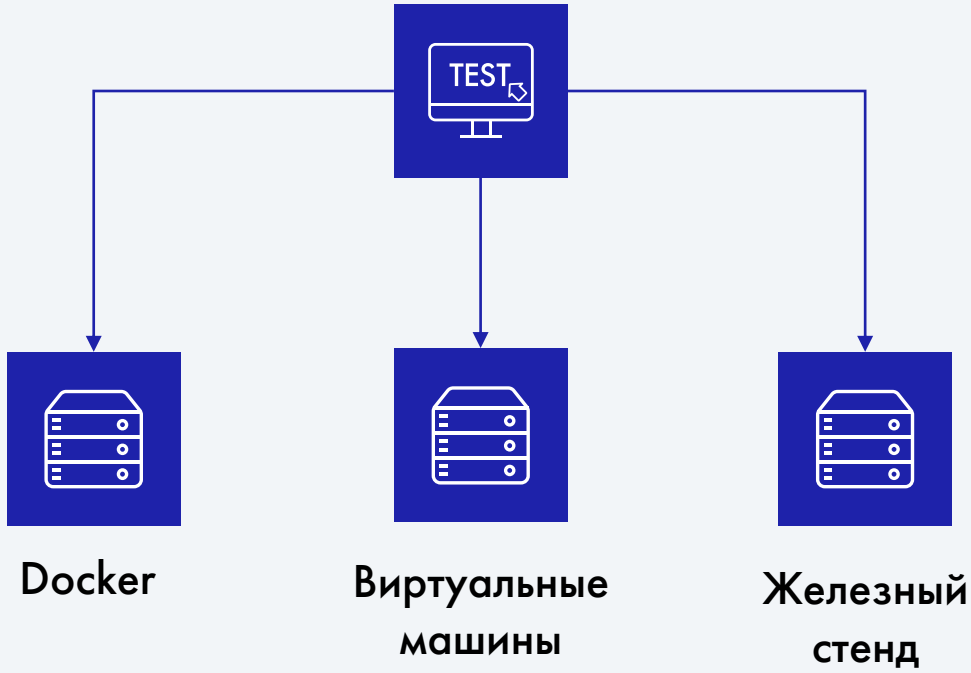
Пример тестового сценария

01 Загрузить файл

02 Перезагрузить сервер

03 Скачать файл

Работает в тестовых средах



Как с этим быть и что делать?

Наш путь

01. Начало пути

Независимые автотесты для двух проектов

02. Robot Framework

Keyword driven testing

03. Библиотека для тестирования

Весь общий код в библиотеку

04. Библиотека для тестирования с плагинами

Весь общий код в библиотеку
Дополняющий закрытый код в плагины

Абсолютно независимые автотесты

Два репозитория

- FrostFS-Testcases
- TO-Testcases

Всё общее дублируется

Код и тестовые сценарии

Разные тестовые среды

Ветвление логики в коде

Всё общее дублируется

FrostFS-Testcases

- Автотест
  - Загрузить файл
  - Перезагрузить сервер
  - Скачать файл
- Реализация методов
  - Загрузки файлов
  - Скачивания файлов
  - Перезагрузка серверов

TO-Testcases

- Автотест
  - Загрузить файл
  - Перезагрузить сервер
  - Скачать файл
- Реализация методов
  - Загрузки файлов
  - Скачивания файлов
  - Перезагрузка серверов

Наш тестовый сценарий

```
CONFIG: dict[str, Any] = yaml.safe_load(“config.yaml”)

@pytest.fixture
def server():
    return CONFIG[“server”]

def test_reboot_server(server):
    upload_file()
    reboot_server(server)
    download_file()
```

Наш тестовый сценарий

```
CONFIG: dict[str, Any] = yaml.safe_load(“config.yaml”)

@pytest.fixture
def server():
    return CONFIG[“server”]

def test_reboot_server(server):
    upload_file()
    reboot_server(server)
    download_file()
```

Наш тестовый сценарий

```
CONFIG: dict[str, Any] = yaml.safe_load(“config.yaml”)

@pytest.fixture
def server():
    return CONFIG[“server”]

def test_reboot_server(server):
    upload_file()
    reboot_server(server)
    download_file()
```

Наш тестовый сценарий

```
CONFIG: dict[str, Any] = yaml.safe_load(“config.yaml”)

@pytest.fixture
def server():
    return CONFIG[“server”]

def test_reboot_server(server):
    upload_file()
    reboot_server(server)
    download_file()
```

Разные тестовые среды – ветвление логики в коде

```
CONFIG: dict[str, Any] = yaml.load(“config”)
ENVIRONMENT_TYPE = CONFIG["env_type"] # virtual

def test_reboot_server(server):
    upload_file()
    if ENVIRONMENT_TYPE == "hardware":
        reboot_hardware(server)
    elif ENVIRONMENT_TYPE == "virtual":
        reboot_vm(server)
    else:
        restart_docker()
   download_file()
```

Сложно добавить новую среду выполнения

```
ENVIRONMENT_TYPE = CONFIG["env_type"] # virtual
def test_reboot_server(server):
    upload_file()
    if ENVIRONMENT_TYPE == "hardware":
        reboot_hardware(server)
    elif ENVIRONMENT_TYPE == "virtual":
        reboot_vm(server)
    elif ENVIRONMENT_TYPE == "новый тип среды":
        reboot_vm(server)
    else:
        restart_docker()
    download_file()
```

Если таких функций много – надо менять каждую

Плохо читаемый код

```
# Может быть virtual/hardware/docker/aio/local/…
ENVIRONMENT_TYPE = "virtual"
def test_reboot_server(server):
    upload_file()
    if ENVIRONMENT_TYPE == "hardware":
        reboot_hardware(server)
    elif ENVIRONMENT_TYPE == "virtual":
        reboot_virtual(server)
    <…> Еще 10-ток ветвлений
    else:
        restart_docker(server)
    download_file()
```

Если окружений много –
легко потеряться

Плохо читаемый код

```python
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Абстракции – чуть лучше,
но все равно не то

Плохо читаемый код

```python
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Абстракции – чуть лучше,
но все равно не то

Плохо читаемый код

```python
@pytest.fixture
def server() → IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Абстракции – чуть лучше,
но все равно не то

Оцениваем подход

Аспект	Полное дублирование
Автотесты FrostFS	🟢
Автотесты Tatlin.Object	🟢
Разные окружения	🟡
Отсутствие утечки	🟢
Дублирование кода	🔴
Читаемость кода	🔴

Наш путь

01. Начало пути

Независимые автотесты
для двух проектов

02. Robot Framework

Keyword driven testing

03. Библиотека для тестирования

Весь общий код в библиотеку

04. Библиотека для тестирования с плагинами

Весь общий код в библиотеку
Дополняющий закрытый код
в плагины

Robot Framework – keyword driven testing

Два репозитория

- FrostFS-Testcases
- TO-Testcases

Всё общее дублируется

Код и тестовые сценарии

Разные тестовые среды

Через отдельные модули

Тестовые среды через отдельные модули

```
# Модуль server.hardware
class HardwareServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка железа через контроллер

# Модуль server.virtual
class VirtualServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка виртуальной машины через API
```

Тестовые среды через отдельные модули

```
# Модуль server.hardware
class HardwareServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка железа через контроллер

# Модуль server.virtual
class VirtualServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка виртуальной машины через API
```

Тестовые среды через отдельные модули

```
# Модуль server.hardware
class HardwareServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка железа через контроллер

# Модуль server.virtual
class VirtualServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка виртуальной машины через API
```

Тестовые среды через отдельные модули

```
# Модуль server.hardware
class HardwareServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка железа через контроллер

# Модуль server.virtual
class VirtualServer:
    @keyword('Reboot server')
    def reboot(self):
        # Перезагрузка виртуальной машины через API
```

Тестовые среды через отдельные модули

```python
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Тестовые среды через отдельные модули

```python
@pytest.fixture
def server() → IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)


def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Тестовые среды через отдельные модули

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}
```

Нужный модуль грузится
по необходимости

```
def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Тестовые среды через отдельные модули

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}
```

Нужный модуль грузится
по необходимости

```
def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Тестовые среды через отдельные модули

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}


*** Test Cases ***
Reboot server
    Upload file
    Reboot server
    Download file
```

Нужный модуль грузится
по необходимости

Оцениваем подход

Аспект	Robot Framework
Автотесты FrostFS	
Автотесты Tatlin.Object	
Разные окружения	
Отсутствие утечки	
Дублирование кода	
Читаемость кода	
Технологическая сложность	

Пример сценария посложнее

Проверка репликации файлов

01 Загрузить файл

02 Подождать репликацию файла до 2 копий

03 Выключить сервер с копией файла

04 Подождать репликации файла до 2 копий еще раз

Пример сценария посложнее

Проверка репликации файлов

01 Загрузить файл

02 Подождать репликацию файла до 2 копий

03 Выключить сервер с копией файла

04 Подождать репликации файла до 2 копий еще раз

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =       Set Variable 0
    WHILE ${copies} < 2       limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
               ${copy_found}=  Find file on  ${server}
               IF  ${copy_found}
               Append To List  ${servers_with_copy} ${server}
               ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =       Set Variable 0
    WHILE ${copies} < 2       limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
               ${copy_found}=  Find file on  ${server}
               IF  ${copy_found}
               Append To List  ${servers_with_copy} ${server}
               ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =         Set Variable 0
    WHILE ${copies} < 2         limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
                ${copy_found}=  Find file on  ${server}
                IF  ${copy_found}
                Append To List  ${servers_with_copy} ${server}
                ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =         Set Variable 0
    WHILE ${copies} < 2         limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
               ${copy_found}=  Find file on  ${server}
               IF  ${copy_found}
               Append To List  ${servers_with_copy} ${server}
               ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =       Set Variable 0
    WHILE ${copies} < 2       limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
               ${copy_found}=  Find file on  ${server}
               IF  ${copy_found}
               Append To List  ${servers_with_copy} ${server}
               ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Сложные случаи с Robot Framework

```
Verify file replication
    Upload file
    ${copies} =       Set Variable 0
    WHILE ${copies} < 2       limit=120 seconds
       @{servers_with_copy}  Create List
        FOR  ${server}  IN ${servers}
               ${copy_found}=  Find file on  ${server}
               IF  ${copy_found}
               Append To List  ${servers_with_copy} ${server}
               ENDIF
        END
        ${copies} =    Get Length  ${servers_with_copy}
```

Проверка копий реплицированных файлов

Наш путь

01. Начало пути

Независимые автотесты
для двух проектов

02. Robot Framework

Keyword driven testing

03. Библиотека для тестирования

Весь общий код в библиотеку

04. Библиотека для тестирования с плагинами

Весь общий код в библиотеку
Дополняющий закрытый код
в плагины

Отдельная библиотека для тестирования

Боремся с дубликацией

Три репозитория

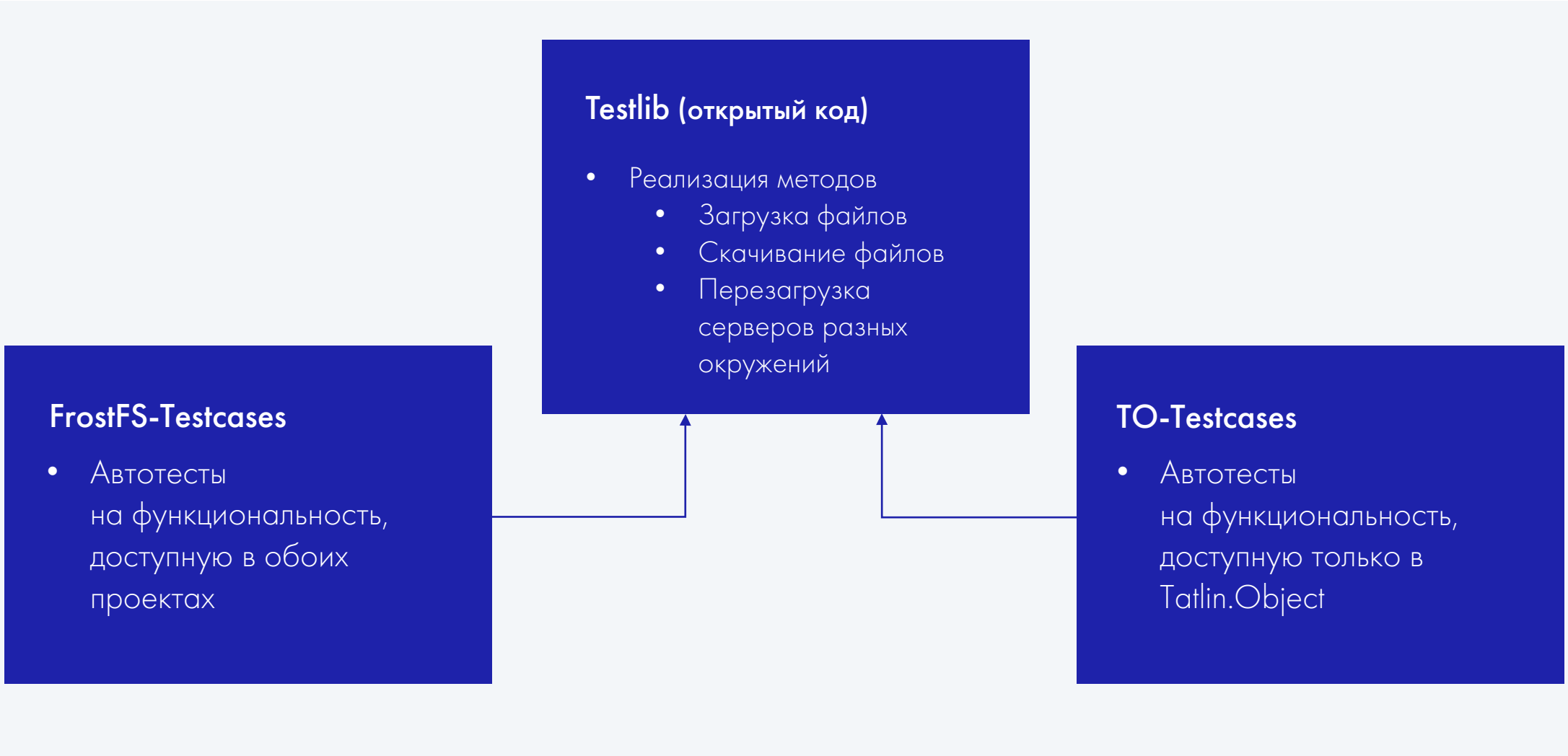
- FrostFS-Testcases
- TO-Testcases
- Testlib

Весь общий код в библиотеку

Общие тесты в публичном доступе

Разные тестовые среды – через абстракцию

Схема устройства тестов



Модули для тестовых сред

```
# Модуль testlib.server
class IServer:
    def __init__(self, config):
        # код инициализации общий для всех

    @abstractmethod
    def reboot(self):
         pass
```

Модули для тестовых сред

```
# Модуль testlib.server
class IServer:
    def __init__(self, config):
        # код инициализации общий для всех

    @abstractmethod
    def reboot(self):
        pass
```

Модули для тестовых сред

```
# Модуль testlib.server.hardware
class HardwareServer(IServer):
    def __init__(self, config):
        # код инициализации для железного стенда


    def reboot(self):
        # Код перезагрузки железного сервера
```

Модули для тестовых сред

```
# Модуль testlib.server.virtual
class VirtualServer(IServer):
    def __init__(self, config):
        # код инициализации для виртуального стенда


    def reboot(self):
        # Код перезагрузки виртуального сервера
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Файл testlib/pyproject.toml
# Задаем классы под окружения
[project.entry-points."testlib.servers"]
hardware = "testlib.servers.hardware:HardwareServer"
virtual = "testlib.servers.virtual:VirtualServer“
docker = "testlib.servers.docker:DockerServer“
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Файл testlib/pyproject.toml
# Задаем классы под окружения
[project.entry-points."testlib.servers"]
hardware = "testlib.servers.hardware:HardwareServer"
virtual = "testlib.servers.virtual:VirtualServer“
docker = "testlib.servers.docker:DockerServer“
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Файл testlib/pyproject.toml
# Задаем классы под окружения
[project.entry-points."testlib.servers"]
hardware = "testlib.servers.hardware:HardwareServer"
virtual = "testlib.servers.virtual:VirtualServer“
docker = "testlib.servers.docker:DockerServer“
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Файл testlib/pyproject.toml
# Задаем классы под окружения
[project.entry-points."testlib.servers"]
hardware = "testlib.servers.hardware:HardwareServer"
virtual = "testlib.servers.virtual:VirtualServer“
docker = "testlib.servers.docker:DockerServer“
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Грузим класс через importlib
from importlib.metadata import entry_points

def get_env_class(group: str, name: str) -> Type:
    classes = entry_points(group=group)
    if name not in classes.names:
        raise Exception(“Not supported env found: {name}”)
    cls = classes[name]
    return cls.load() # вернет нужный класс
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Грузим класс через importlib
from importlib.metadata import entry_points

def get_env_class(group: str, name: str) -> Type:
    classes = entry_points(group=group)
    if name not in classes.names:
        raise Exception(“Not supported env found: {name}”)
    cls = classes[name]
    return cls.load() # вернет нужный класс
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Грузим класс через importlib
from importlib.metadata import entry_points

def get_env_class(group: str, name: str) -> Type:
    classes = entry_points(group=group)
    if name not in classes.names:
        raise Exception(“Not supported env found: {name}”)
    cls = classes[name]
    return cls.load() # вернет нужный класс
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Грузим класс через importlib
from importlib.metadata import entry_points

def get_env_class(group: str, name: str) -> Type:
    classes = entry_points(group=group)
    if name not in classes.names:
        raise Exception(“Not supported env found: {name}”)
    cls = classes[name]
    return cls.load() # вернет нужный класс
```

Великолепный importlib

Спецификация docs.python.org/3/library/importlib.metadata.html

```
# Грузим класс через importlib
from importlib.metadata import entry_points

def get_env_class(group: str, name: str) -> Type:
    classes = entry_points(group=group)
    if name not in classes.names:
        raise Exception(“Not supported env found: {name}”)
    cls = classes[name]
    return cls.load() # вернет нужный класс
```

Было

```python
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer()
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer()
    return DockerServer()

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Абстракции
из модифицированного
первого подхода

Было

```python
@pytest.fixture
def server() → IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer()
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer()
    return DockerServer()

def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Абстракции
из модифицированного
первого подхода

Стало

```
@pytest.fixture
def server() -> IServer:
    cls = get_env_class(group="testlib.servers",
                        name=ENVIRONMENT_TYPE)
    return cls(CONFIG)



def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Выдаем нужный класс в тестах

Оцениваем

Аспект	Общая библиотека
Автотесты FrostFS	🟢
Автотесты Tatlin.Object	🟢
Разные окружения	🟢
Отсутствие утечки	🔴
Дублирование кода	🟢
Читаемость кода	🟢
Технологическая сложность	🟢
Зависимость репозиториев	🟡

Наш путь

01. Начало пути

Независимые автотесты
для двух проектов

02. Robot Framework

Keyword driven testing

03. Библиотека для тестирования

Весь общий код в библиотеку

04. Библиотека для тестирования с плагинами

Весь общий код в библиотеку
Дополняющий закрытый код
в плагины

Наш путь

Много репозиториев

- FrostFS-Testcases
- TO-Testcases
- Testlib
- Плагины

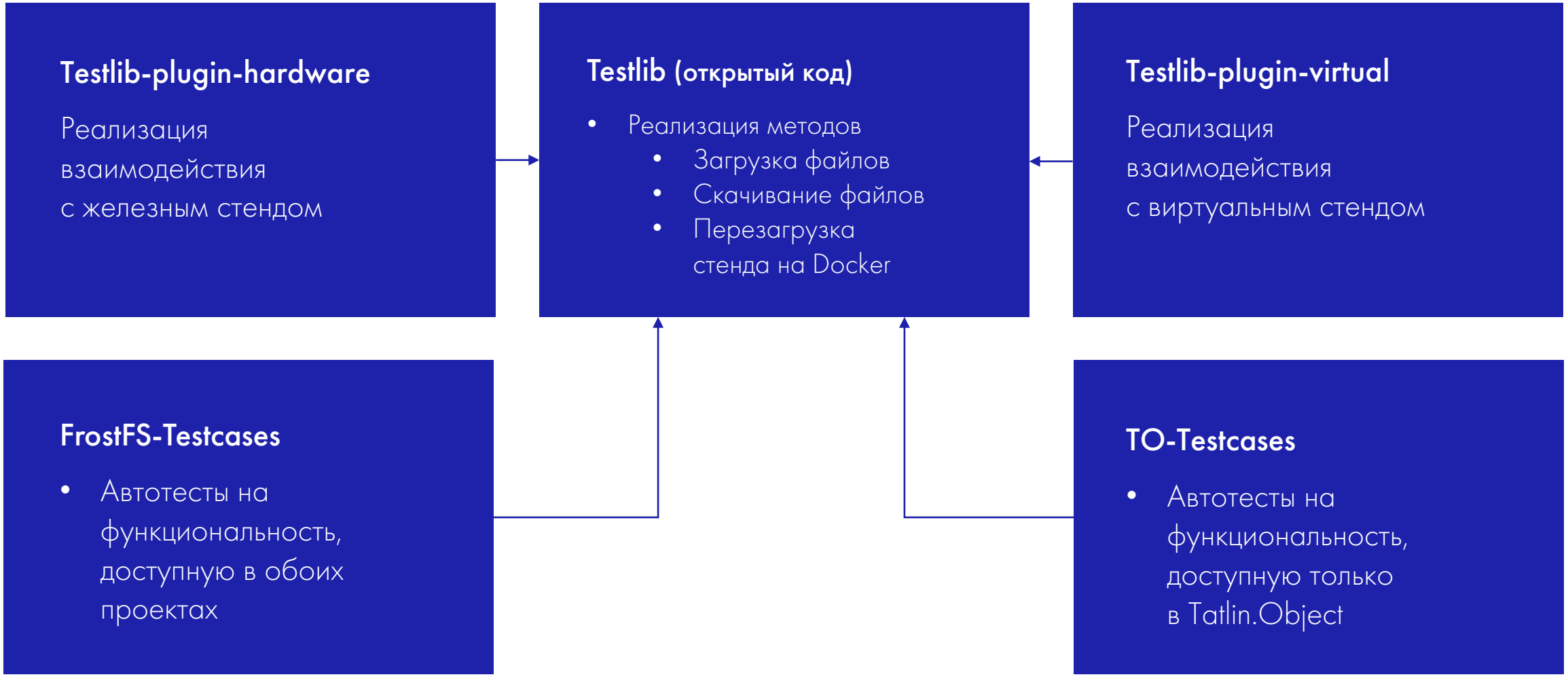
Весь общий код в библиотеку

Коммерческий код в плагины

Общие тесты в публичном доступе

Разные тестовые среды – через абстракцию с Importlib

Схема устройства тестов



Что изменилось?

В плагинах есть свои собственные entry points

```
# Файл testlib-plugin-hardware/pyproject.toml
[project.entry-points."testlib.servers"]
hardware = "testlib_plugin_hardware:HardwareServer"


# Файл testlib-plugin-virtual/pyproject.toml
[project.entry-points."testlib.servers"]
virtual = "testlib_plugin_virtual:VirtualServer"
```

Что изменилось?

В плагинах есть свои собственные entry points

```
# Файл testlib-plugin-hardware/pyproject.toml
[project.entry-points."testlib.servers"]
hardware = "testlib_plugin_hardware:HardwareServer"


# Файл testlib-plugin-virtual/pyproject.toml
[project.entry-points."testlib.servers"]
virtual = "testlib_plugin_virtual:VirtualServer"
```

Что изменилось?

В плагинах есть свои собственные entry points

```
# Файл testlib-plugin-hardware/pyproject.toml
[project.entry-points."testlib.servers"]
hardware = "testlib_plugin_hardware:HardwareServer"


# Файл testlib-plugin-virtual/pyproject.toml
[project.entry-points."testlib.servers"]
virtual = "testlib_plugin_virtual:VirtualServer"
```

Решаем проблему верификации

В тестовых репозиториях

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./tests

validation: lint
        pytest --collect-only
```

Решаем проблему верификации

В тестовых репозиториях

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./tests

validation: lint
        pytest --collect-only
```

Решаем проблему верификации

В тестовых репозиториях

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./tests

validation: lint
        pytest --collect-only
```

Решаем проблему верификации

В тестовых репозиториях

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./tests

validation: lint
        pytest --collect-only
```

Решаем проблему верификации

В плагинах

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./src

unit_test:
         pytest ./tests

validation: lint unit_test
```

Решаем проблему верификации

В плагинах

```
# Makefile
lint: create_venv
        pip install -e ../frostfs-testlib
        pylint ./src

unit_test:
        pytest ./tests

validation: lint unit_test
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Решаем проблему верификации

В тестлибе – запускаем валидацию с неё

```
DIRECTORIES := $(sort $(dir $(wildcard ../testlib-plugin-*/ ../*-testcases/)))
lint: create_venv
	pylint ./src
lint_dependent: $(DIRECTORIES)
$(DIRECTORIES):
	@echo checking dependent repo $@
	$(MAKE) validation -C $@
validation: lint unit_test lint_dependent
```

Оцениваем

Аспект	Библиотека + плагины
Автотесты FrostFS	●
Автотесты Tatlin.Object	●
Разные окружения	●
Отсутствие утечки	●
Дублирование кода	●
Читаемость кода	●
Технологическая сложность	●
Зависимость репозиториев	●

Эволюция тестового сценария

Начало пути

```
CONFIG: dict[str, Any] = yaml.load(“config”)
ENVIRONMENT_TYPE = CONFIG["env_type"] # virtual

def test_reboot_server(server):
    upload_file()
    if ENVIRONMENT_TYPE == "hardware":
        reboot_hardware(server)
    elif ENVIRONMENT_TYPE == "virtual":
        reboot_vm(server)
    else:
        restart_docker()
   download_file()
```

Эволюция тестового сценария

Улучшаем дело абстракцией

```
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)


def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Эволюция тестового сценария

Улучшаем дело абстракцией

```
@pytest.fixture
def server() -> IServer:
    if ENVIRONMENT_TYPE == "hardware":
        return HardwareServer(CONFIG)
    elif ENVIRONMENT_TYPE == "virtual":
        return VirtualServer(CONFIG)
    return DockerServer(CONFIG)


def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Эволюция тестового сценария

Пробуем Keyword-Driven Testing подход

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}
```

```
def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Эволюция тестового сценария

Пробуем Keyword-Driven Testing подход

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}
```

```
def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Эволюция тестового сценария

Пробуем Keyword-Driven Testing подход

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}



*** Test Cases ***
Reboot server
    Upload file
    Reboot server
    Download file
```

Эволюция тестового сценария

Пробуем Keyword-Driven Testing подход

```
***Settings***
Library         server.${ENVIRONMENT_TYPE}




*** Test Cases ***
Reboot server
    Upload file
    Reboot server
    Download file
```

Эволюция тестового сценария

Нынешний вариант

```
@pytest.fixture
def server() -> IServer:
    cls = get_env_class(group="testlib.servers",
                        name=ENVIRONMENT_TYPE)
    return cls(CONFIG)



*** Test Cases ***
Reboot server
    Upload file
    Reboot server
    Download file
```

Эволюция тестового сценария

Нынешний вариант

```
@pytest.fixture
def server() -> IServer:
    cls = get_env_class(group="testlib.servers",
                        name=ENVIRONMENT_TYPE)
    return cls(CONFIG)
```

```
*** Test Cases ***
Reboot server
    Upload file
    Reboot server
    Download file
```

Эволюция тестового сценария

Нынешний вариант

```
@pytest.fixture
def server() -> IServer:
    cls = get_env_class(group="testlib.servers",
                        name=ENVIRONMENT_TYPE)
    return cls(CONFIG)


def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Эволюция тестового сценария

Нынешний вариант

```
@pytest.fixture
def server() -> IServer:
    cls = get_env_class(group="testlib.servers",
                        name=ENVIRONMENT_TYPE)
    return cls(CONFIG)


def test_reboot_server(server: IServer):
    upload_file()
    server.reboot()
    download_file()
```

Сравниваем

Аспект	Независимые тесты	Robot Framework	Общая библиотека	Библиотека + плагины
Автотесты FrostFS	🟢	🟢	🟢	🟢
Автотесты Tatlin.Object	🟢	🟢	🟢	🟢
Разные окружения	🟡	🟢	🟢	🟢
Отсутствие утечки	🟢	🟢	🔴	🟢
Дублирование кода	🔴	🔴	🟢	🟢
Читаемость кода	🔴	🟢	🟢	🟢
Технологическая сложность	🟢	🔴	🟢	🟢
Зависимость репозиториев	🟢	🟢	🟡	🟢

Итоги

01 Существенно снизили дороговизну поддержки автотестов

02 Смогли выпустить СХД практически с нуля за полгода

Выводы

Не бойтесь строить коммерческий продукт с использованием opensource

Используйте абстракции для разделения логики в коде тестов

Выносите общий код в библиотеки, а закрытый код в плагины

Подгружайте нужные абстракции на лету с помощью importlib

Testlib

Testcases

yadro.com