

В. В. Михайлюк¹✉, Е. Ю. Воронкин¹

Разработка Python-фреймворка для автоматизации релизных процессов с интеграцией в CI/CD-систему Jenkins

¹Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: vitia.mikhailiuk@mail.ru

Аннотация. В статье рассматривается процесс разработки Python-фреймворка для автоматизации релизных процессов с интеграцией в систему непрерывной интеграции и доставки Jenkins. Актуальность работы обусловлена необходимостью вынесения сложной бизнес-логики релизных конвейеров за пределы конкретных CI/CD-оркестраторов и обеспечения ее единообразного исполнения как на локальных компьютерах разработчиков, так и в распределенных сборочных кластерах. В качестве технологической основы выбран язык Python в сочетании с библиотеками Click, Pydantic, Jinja2, Cookiecutter, Loguru и rich. Спроектирована многоуровневая архитектура с разграничением ответственности между слоем командного интерфейса, ядром и подсистемами пайплайнов, плагинов и логирования. Реализованы динамическая загрузка пользовательского кода посредством importlib, типизированные модели параметров и секретов, интеграция с HashiCorp Vault для управления секретами и передача данных наблюдаемости в OpenSearch. Корректность работы подтверждена комплексом из 214 автоматизированных тестов с покрытием исходного кода 80 %. Полученное программное решение пригодно для практического применения и последующего расширения за счет плагиновой архитектуры.

Ключевые слова: python, автоматизация, релизный процесс, ci/cd, jenkins, фреймворк, консольное приложение, click, pydantic, jinja2, cookiecutter, loguru, плагиновая архитектура, hashicorp vault, opensearch

V. V. Mikhaylyuk¹✉, E. Yu. Voronkin¹

Development of a python framework for release process automation with integration into the Jenkins CI/CD system

¹Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: vitia.mikhailiuk@mail.ru

Abstract. The article discusses the process of developing a Python framework for release process automation with integration into the Jenkins continuous integration and delivery system. The relevance of the work is determined by the need to decouple complex business logic of release pipelines from specific CI/CD orchestrators and to ensure its uniform execution both on developers' local machines and in distributed build clusters. Python was chosen as the technological basis together with the Click, Pydantic, Jinja2, Cookiecutter, Loguru and rich libraries. A multi-level architecture has been designed with a clear separation of responsibilities between the command-line interface layer, the core, and the pipelines, plugins and logging subsystems. Dynamic loading of user code via importlib, typed models of parameters and secrets, integration with HashiCorp Vault for secret management, and transmission of observability data to OpenSearch have been implemented. Correctness is confirmed by a suite of 214 automated tests with 80 % source code coverage. The resulting software solution is suitable for practical application and further extension through its plugin architecture.

Keywords: python, automation, release process, ci/cd, jenkins, framework, command-line application, click, pydantic, jinja2, cookiecutter, loguru, plugin architecture, hashicorp vault, opensearch

Введение

Современный цикл выпуска программных продуктов неразрывно связан с практиками непрерывной интеграции и доставки и предполагает оркестрацию десятков внешних систем – трекеров задач, баз знаний, репозиторий артефактов, систем статического анализа и корпоративных мессенджеров [1]. На практике реализация таких межинтеграционных связей часто осуществляется стихийно посредством разрозненных сценариев на Bash, PowerShell или встроенных DSL-языках CI-серверов, что порождает высокую связность кода, отсутствие единых механизмов обработки ошибок, риски утечки конфиденциальных данных в журналы и невозможность локального запуска бизнес-логики для отладки.

В качестве альтернативы данному подходу рассматривается вынесение сложной бизнес-логики из конфигурации CI/CD в самостоятельные консольные приложения на языках общего назначения, в первую очередь на Python [2]. Такой подход позволяет применять к инфраструктурным сценариям классические практики программной инженерии: модульное тестирование [3], объектно-ориентированное проектирование и строгий статический анализ.

Целью настоящей работы является разработка Python-фреймворка для автоматизации релизных процессов с интеграцией в CI/CD-систему Jenkins, обеспечивающего независимое и безопасное выполнение интеграционных сценариев как на локальных компьютерах разработчиков, так и в распределенных сборочных кластерах.

Для достижения поставленной цели выполнены следующие задачи:

- провести анализ существующих методов и средств создания автоматизационных сценариев и сравнительное рассмотрение представленных на рынке решений;
- обосновать технологический стек реализации;
- проектировать функциональную структуру фреймворка и построить комплекс UML-моделей;
- реализовать ядро системы и подсистемы пайплайнов, плагинов и логирования, а также интерфейс командной строки;
- разработать комплекс автоматизированных тестов и провести количественную оценку покрытия исходного кода.

Методы и материалы

Перед началом программной реализации фреймворка спроектирована его функциональная структура и построен комплекс UML-моделей, охватывающих систему на структурном и поведенческом уровнях [4].

Диаграмма прецедентов формализовала функциональные требования к системе с позиции единственного актора – разработчика – и зафиксировала пять групп сценариев использования: управление проектом, взаимодействие с пай-

пайнами, работу с конфигурацией, расширение через плагины и контроль выполнения (рис. 1).

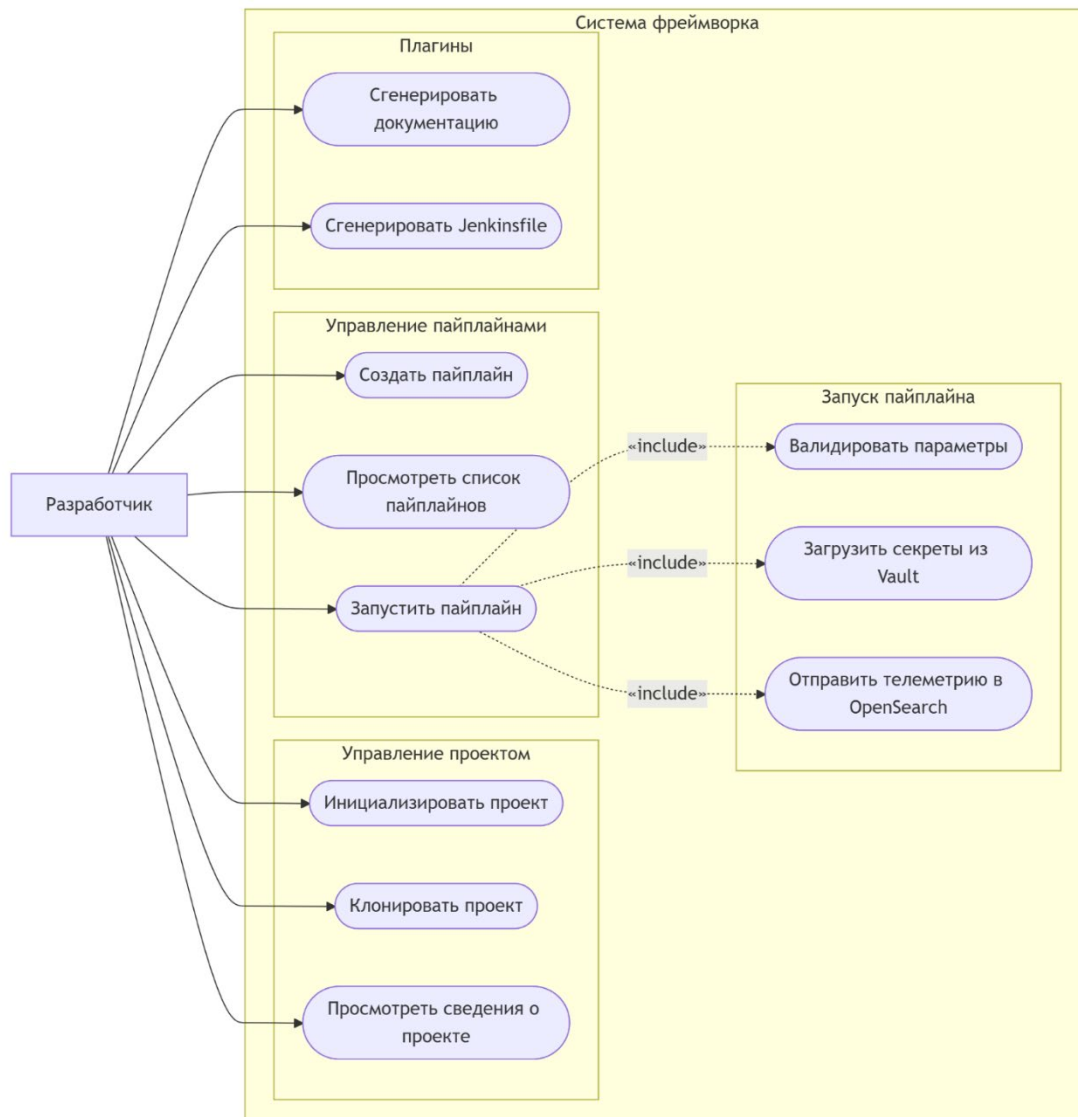


Рис. 1. Диаграмма прецедентов фреймворка

Диаграмма классов описала объектную модель ядра, сгруппировав ключевые классы по функциональному назначению – управление выполнением, передача данных и валидация – и зафиксировав отношения агрегации, композиции, наследования и зависимости (рис. 2).

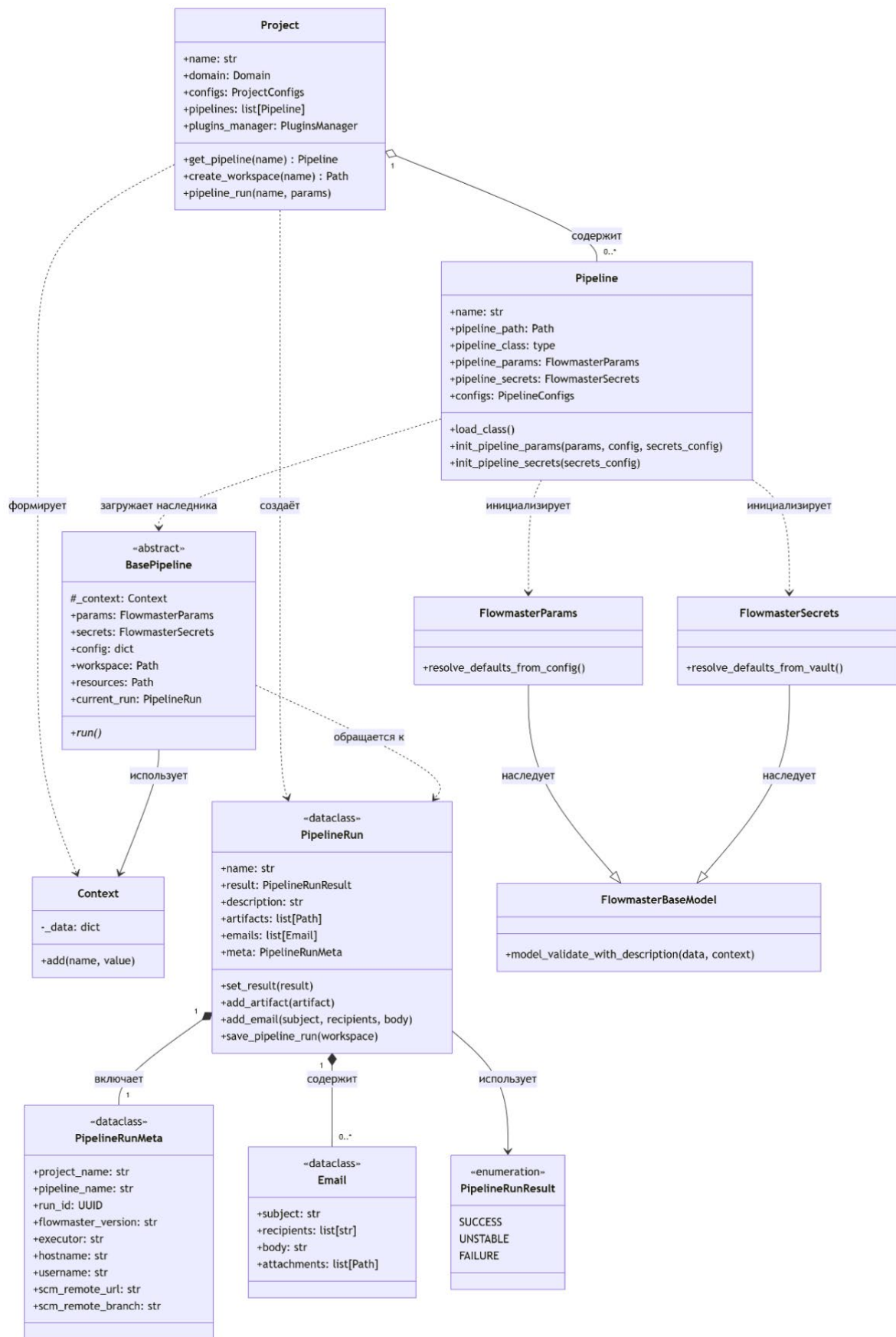


Рис. 2. Диаграмма классов фреймворка

Диаграмма последовательностей смоделировала временной порядок взаимодействия компонентов в ходе типового запуска пайплайна, отразив инициализацию проекта, подготовку запуска, динамическую загрузку пайплайна, валидацию параметров, разрешение секретов, формирование контекста исполнения и сохранение результатов (рис. 3).

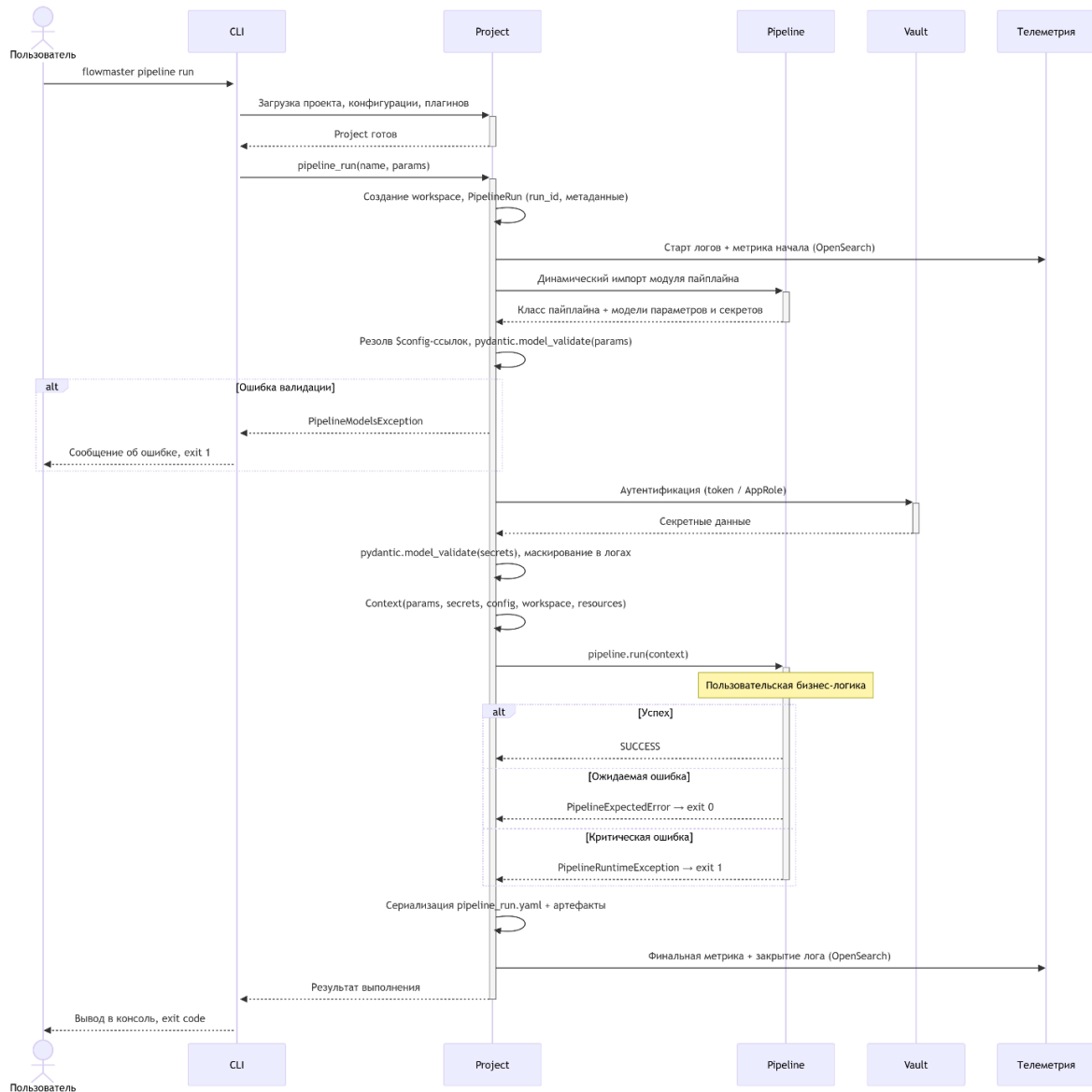


Рис. 3. Диаграмма последовательностей выполнения пайплайна

В качестве серверной части и точки входа выбран язык программирования Python с богатой объектно-ориентированной моделью и развитой системой типизации. Интерфейс командной строки реализован средствами библиотеки Click, обеспечивающей маршрутизацию команд через декораторы и автоматическую генерацию справочной документации. Валидация параметров пайплайнов и секретов выполняется на основе типизированных моделей библиотеки Pydantic [5]. Для генерации конфигурационных файлов и шаблонов документации применя-

ется шаблонизатор Jinja2; для разворачивания типовой структуры новых проектов и пайплайнов используется библиотека Cookiecutter. Подсистема журналирования построена на основе библиотеки Loguru с поддержкой потокобезопасной записи через очередь, форматированный табличный вывод в терминал реализован средствами библиотеки rich.

Хранение и доступ к секретам осуществляются через HashiCorp Vault [6], что обеспечивает централизованное управление учетными данными и их изоляцию от исходного кода пайплайнов. Журналы и метрики выполнения направляются в систему централизованной телеметрии на базе OpenSearch [7] и используются для последующего анализа и построения дашбордов.

Результаты

В соответствии со спроектированной архитектурой реализованы все предусмотренные подсистемы фреймворка.

Ядро системы оформлено в пакете flowmaster.core. Класс Project выступает единой точкой оркестрации и инкапсулирует жизненный цикл запуска пайплайна. Подсистема конфигурации реализует доменно-специфичную загрузку параметров с разделением на базовый и доменный уровни, а иерархия исключений обеспечивает дифференцированную обработку ошибок различных категорий. Вспомогательные утилиты – рекурсивное слияние словарей, безопасный доступ к вложенным ключам и контекстный менеджер смены рабочей директории – формируют переиспользуемые примитивы для всех последующих подсистем.

Подсистема пайплайнов построена вокруг базового класса BasePipeline и метаклассового механизма копирования атрибутов вложенного класса Meta. Динамическая загрузка пользовательского кода выполняется через стандартный модуль importlib [8], что позволяет отделить пользовательский код от ядра фреймворка. Параметры и секреты пайплайна описываются типизированными моделями FlowmasterParams и FlowmasterSecrets с автоматическим разрешением ссылок на значения из конфигурации и из HashiCorp Vault. Класс Context формирует окружение исполнения, а классы PipelineRun и PipelineRunMeta обеспечивают трассируемое сохранение результатов запуска в файле pipeline_run.yaml.

Подсистема логирования реализована в пакете flowmaster.logging на базе библиотеки Loguru. Глобальный фильтр маскирования конфиденциальных данных реализован с применением паттерна Singleton [9] и гарантирует обработку секретов во всех выходных потоках независимо от точки журналирования. Передача журналов во внешнюю систему агрегации выполняется через очередь и фоновый поток, что выносит сетевое взаимодействие за пределы основного потока исполнения и сохраняет работоспособность пайплайна при временной недоступности OpenSearch.

Подсистема плагинов основана на разделении контракта Plugin/CliPlugin и менеджера PluginsManager, выполняющего динамическую загрузку классов плагинов по путям, указанным в конфигурации. Такое решение реализует принцип открытости/закрытости: добавление новой интеграции сводится к созданию класса-наследника CliPlugin и регистрации его в конфигурации без изменений в

ядре фреймворка. На основе данного контракта разработаны два штатных плагина – генератор документации пайплайнов средствами Jinja2 и интеграционный плагин Jenkins, обеспечивающий формирование Jenkinsfile и общей разделяемой библиотеки.

Слой командного интерфейса реализован в пакете flowmaster.cli на основе библиотеки Click и объединяет все точки взаимодействия пользователя с фреймворком. Группа команд project отвечает за управление каркасом автоматизационного проекта, группа pipeline – за работу со списком пайплайнов, их инициализацию из шаблона и запуск с передачей параметров в формате JSON. Декоратор @pass_obj_with_project выполняет ленивую инициализацию объекта Project и пробрасывает его в обработчики команд. Команды плагинов встраиваются в общее дерево CLI наряду со штатными группами, что обеспечивает единообразный пользовательский опыт. Пример интерфейса представлен на рисунках 5, 6, 7, 8.

```
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development
• $ mkdir test-project
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development
• $ cd test-project
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
• $ flowmaster project init -n test-project
2026-04-30 21:56:05 | INFO | Генерация нового проекта test-project в E:\Development\test-project.
2026-04-30 21:56:05 | SUCCESS | ✔ Проект test-project успешно инициализирован.
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
• $ flowmaster project current
2026-04-30 21:56:13 | INFO | Информация о проекте test-project
```

Параметр	Значение
name	test-project
description	Описание проекта test-project
created_at	30-04-2026
default_domain	test
plugins	flowmaster.plugins.jenkins, flowmaster.plugins.docs

```
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
○ $
```

Рис. 5. Пример работы команд интерфейса фреймворка

```
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
• $ flowmaster pipeline init -n test_pipeline
2026-04-30 22:00:36 | SUCCESS | ✔ Пайплайн test_pipeline успешно инициализирован.
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
• $ flowmaster pipeline run -n test_pipeline
2026-04-30 22:00:42 | INFO | Инициализация параметров пайплайна 'test_pipeline'.
2026-04-30 22:00:42 | SUCCESS | ✔ Инициализация параметров пайплайна 'test_pipeline' завершена успешно.
2026-04-30 22:00:42 | INFO | Инициализация секретов пайплайна 'test_pipeline'.
2026-04-30 22:00:42 | SUCCESS | ✔ Инициализация секретов пайплайна 'test_pipeline' завершена успешно.
2026-04-30 22:00:42 | INFO | Запуск пайплайна 'test_pipeline'.
2026-04-30 22:00:42 | INFO | Тестовый запуск.
2026-04-30 22:00:42 | INFO | Завершена работа пайплайна 'test_pipeline'.
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project
○ $
```

Рис. 6. Пример работы команд интерфейса фреймворка

```

vitia@viktor-pc MINGW64 /e/Development/test-project
● $ flowmaster pipeline run -n test_pipeline
2026-05-01 00:07:23 | INFO | Инициализация параметров пайплайна 'test_pipeline'.
2026-05-01 00:07:23 | SUCCESS | ✓ Инициализация параметров пайплайна 'test_pipeline' завершена успешно.
2026-05-01 00:07:23 | INFO | Инициализация секретов пайплайна 'test_pipeline'.
2026-05-01 00:07:23 | SUCCESS | ✓ Инициализация секретов пайплайна 'test_pipeline' завершена успешно.
2026-05-01 00:07:23 | INFO | Запуск пайплайна 'test_pipeline'.
2026-05-01 00:07:23 | INFO | Pydantic-модель с тестовым секретом из HashiCorp Vault: test_key='[***HIDDEN***]'
2026-05-01 00:07:23 | INFO | Завершена работа пайплайна 'test_pipeline'.

```

Рис. 7. Пример работы команд интерфейса фреймворка

```

vitia@viktor-pc MINGW64 /e/Development/test-project (master)
● $ flowmaster jenkins init
2026-05-01 01:14:23 | INFO | Генерация flowmaster-shared-library
2026-05-01 01:14:23 | SUCCESS | ✓ Генерация flowmaster-shared-library завершена успешно!
(flowmaster-py3.12)
vitia@viktor-pc MINGW64 /e/Development/test-project (master)
● $ flowmaster jenkins generate pipeline -n test_pipeline -j test_instance
2026-05-01 01:14:25 | SUCCESS | ✓ Jenkinsfile для пайплайна test_pipeline успешно обновлен.

```

Рис. 8. Пример работы команд интерфейса фреймворка

Корректность реализации подтверждена разработанным комплексом автоматизированных тестов на базе фреймворка `pytest`. Набор включает 214 тестов, распределенных по двадцати одному модулю, и сочетает модульное тестирование изолированных единиц поведения с функциональными end-to-end сценариями, выполняющими реальные CLI-команды в изолированной временной директории. Изоляция тестового окружения достигается стандартными фикстурами `pytest` и подменой внешних зависимостей средствами модуля `unittest.mock`. Автоматизированный прогон завершается без отказов; измеренное средствами `coverage.py` покрытие исходного кода составляет 80 % от 1621 исполняемой инструкции фреймворка, при этом подсистемы ядра, моделей параметров и секретов, логирования и плагиновых контрактов покрыты полностью.

Заключение

В статье рассмотрен процесс разработки Python-фреймворка для автоматизации релизных процессов с интеграцией в CI/CD-систему Jenkins. На основании проведенного анализа существующих подходов обоснована необходимость вынесения бизнес-логики релизных конвейеров за пределы конкретных оркестраторов в самостоятельное программное решение. Спроектирована многоуровневая архитектура с разграничением ответственности между слоем командного интерфейса, ядром и подсистемами пайплайнов, плагинов и логирования; согласованность принятых решений зафиксирована комплексом UML-моделей.

Реализация фреймворка обеспечила динамическую загрузку пользовательского кода, типизированную валидацию параметров и секретов, централизованное управление секретами через HashiCorp Vault, асинхронную передачу журналов в OpenSearch и расширяемость через плагиновый контракт. Корректность работы подтверждена прогоном 214 автоматизированных тестов без отказов и достигнутым покрытием исходного кода в 80 %.

Полученное программное решение пригодно к практическому применению в качестве унифицированного инструмента описания и исполнения многокомпонентных автоматизационных сценариев, обеспечивающего единообразный запуск как в локальном окружении разработчика, так и в кластере Jenkins. Перспективы дальнейшего развития связаны с расширением набора штатных плагинов для распространенных CI/CD-платформ и трекеров задач, реализацией параллельного исполнения нескольких пайплайнов, подключением альтернативных систем управления секретами и телеметрии, а также с развитием инструментальной поддержки разработки пользовательских пайплайнов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Хамбл Д., Фарли Д. Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ / Пер. с англ. А. Г. Сысолюка. – М.: Вильямс, 2011. – 432 с. – ISBN 978-5-8459-1739-3.
2. Гифт Н., Берман К., Деза А., Георгиу Г. Python и DevOps: ключ к автоматизации Linux / Пер. с англ. – СПб.: Питер, 2022. – ISBN 978-5-4461-2929-4.
3. Майерс Г., Баджетт Т., Сандлер К. Искусство тестирования программ. 3-е изд. / Пер. с англ. – М.: Вильямс, 2012. – 272 с. – ISBN 978-5-8459-1796-6.
4. Фаулер М. UML. Основы: краткое руководство по стандартному языку объектного моделирования. 3-е изд. / Пер. с англ. – СПб.: Символ-Плюс, 2005. – 192 с. – ISBN 978-5-93286-060-1.
5. Pydantic. Официальная документация. – URL: <https://docs.pydantic.dev/> (дата обращения: 08.04.2026).
6. HashiCorp Vault. Официальная документация. – URL: <https://developer.hashicorp.com/vault/docs> (дата обращения: 16.04.2026).
7. OpenSearch. Официальная документация. – URL: <https://opensearch.org/docs/latest> (дата обращения: 16.04.2026).
8. Python Software Foundation. Документация стандартной библиотеки Python 3: модуль importlib. – URL: <https://docs.python.org/3/library/importlib.html> (дата обращения: 17.04.2026).
9. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Паттерны объектно-ориентированного проектирования / Пер. с англ. – СПб.: Питер, 2021. – 448 с. – ISBN 978-5-4461-1595-2.

© В. В. Михайлюк, Е. Ю. Воронкин, 2026