УДК 004.9+005.8

***Зиянуров Артём Андреевич**, Магистрант 1 курса*

***Московский физико-технический институт (национальный исследовательский университет)**, г. Москва*

API DRIFT: ОПРЕДЕЛЕНИЕ, КЛАССИФИКАЦИЯ И СВЯЗЬ С GOVERNANCE-РАМКОЙ

Аннотация

В статье вводится и формализуется понятие API Drift — расхождения между представлениями программного интерфейса (спецификация, код, документация, каталог) и его проявлениями (production-трафик, логи, метрики). На основе концептуального анализа существующих определений API предложено интегративное рабочее определение, введено разграничение представлений и проявлений API, сформулирована фасетная классификация расхождений и обоснован принцип анализа drift через governance-рамку из четырёх компонентов. Статья носит концептуально-терминологический характер и закладывает основу для дальнейших эмпирических исследований.

Annotation

The paper introduces and formalizes the concept of API Drift — divergences between API representations (specification, code, documentation, catalog) and API manifestations (production traffic, logs, metrics). Based on conceptual analysis of existing API definitions, we propose an integrative working definition, distinguish between representations and manifestations of an API, develop a faceted classification of drifts, and justify analyzing drift through a four-component governance framework. The paper is conceptual-terminological in nature and lays the groundwork for further empirical research.

Ключевые слова: API Drift, API governance, программный интерфейс, спецификация API, жизненный цикл API, расхождение спецификации и реализации, API как продукт, фасетная классификация.

Keywords: API Drift, API governance, application programming interface, API specification, API lifecycle, specification-implementation divergence, API as a product, faceted classification.

1. Введение

Программные интерфейсы (API) стали ключевым элементом архитектуры современных информационных систем. Рост числа API в организациях — от десятков до тысяч — порождает проблему согласованности: спецификация отстаёт от кода, документация — от спецификации, каталог API — от реальности [1][2]. Это явление в индустрии получило название API drift [3][4].

Однако существующие определения API drift ограничиваются узким техническим контекстом (расхождение OpenAPI-спецификации и runtime-трафика) и не охватывают системно все виды расхождений, возникающих на протяжении жизненного цикла API. Кроме того, отсутствует формальная связь между drift и практиками API governance, что затрудняет принятие обоснованных управленческих решений. В научной литературе проблема расхождения между спецификацией и реализацией исследуется в более широком контексте архитектурного дрейфа (architectural drift) и эрозии архитектуры (architectural erosion) [12][13], однако применительно к API как самостоятельному объекту управления эти концепции до настоящего времени не формализованы.

Цель статьи — ввести расширенное определение API Drift, предложить его фасетную классификацию и обосновать принцип анализа расхождений через governance-рамку из четырёх компонентов.

Задачи: (1) синтезировать интегративное рабочее определение API; (2) разграничить представления и проявления API; (3) на этой основе формализовать понятие API Drift; (4) предложить фасетную классификацию; (5) показать связь drift с governance-пробелами.

2. Обзор литературы

Понятие API governance активно исследуется с начала 2010-х годов. В ранних работах Krintz et al. (2014) определяют API governance как совокупность политик, реализации и функций развёртывания для управления API в масштабе, рассматривая облачные платформы (PaaS) как идеальную среду для deployment-time governance [5]. Haupt, Leymann и Vukojevic-Haupt (2018) предлагают фреймворк структурного анализа REST API и выводят метрики сложности, поддерживающие governance-задачи [1]. Gamez-Diaz et al. (2024) описывают опыт Google по масштабированию API governance через API Improvement Proposals (AIP), автоматический линтинг и программу API Readability [2]. Suescún Monsalve et al. (2024) проводят systematic mapping литературы по API governance в финансовом секторе, подтверждая растущий академический интерес к данной теме [15].

Отдельным направлением исследований является разработка моделей зрелости для API Management. Overeem, Mathijssen и Jansen (2022) представляют API-m-FAMM — focus area maturity model, охватывающую шесть областей: Lifecycle Management, Security, Performance, Observability, Community и Commercial. Модель построена на основе систематического обзора литературы, экспертных интервью и case studies [14].

Lauret (2022) формализует API governance как систему из четырёх компонентов: политики, институты, процессы, индикаторы [8]. Данная модель используется в настоящей статье в качестве governance-рамки для анализа drift.

Отдельно развивается взгляд на API как продукт с жизненным циклом. Medjaoui et al. (2022) в книге «Continuous API Management» систематизируют стадии жизненного цикла и вводят понятие API-портфеля [9]. Red Hat описывает API lifecycle management как непрерывный процесс от стратегии до вывода из эксплуатации [10]. Nordic APIs рассматривает API как самостоятельный продукт с собственной ценностной моделью [11]. В академической литературе эта концепция находит отражение в исследованиях эволюции Web API: Serbout, Di Lauro и Pautasso (2022) проводят

крупномасштабный эмпирический анализ 42 194 OpenAPI-спецификаций, изучая структуры и модели данных [18]; аналогичное направление развивается в работах по отслеживанию жизненного цикла API-артефактов [19].

Термин API drift используется в индустрии преимущественно в контексте безопасности. Anoop S. (2024) исследует API drift detection как механизм защиты от утечки данных [3]. Wiz Research определяет specification drift как расхождение runtime-поведения и OpenAPI-спецификации [4]. APIscene выделяет несколько типов drift и методы их обнаружения [6]. Независимое исследование APIContext (2024) на большой выборке production API показало, что около 75% обследованных API имеют расхождения между опубликованной OpenAPI-спецификацией и фактическим поведением [7]. Apiiro (2022) рассматривает architecture drift как нарастающую дельту между спецификацией и результатом разработки [17].

В научной литературе проблема расхождения между проектными артефактами и реализацией исследуется в области architecture consistency. Ali et al. (2018) эмпирически изучают состояние практики в отношении согласованности архитектуры, выявляя ключевые барьеры для применения формальных подходов [13]. Anthony et al. (2024) исследуют architectural drift с точки зрения разработчиков, показывая различия в восприятии проблемы между junior- и senior-специалистами [12].

Göransson (2019) предлагает фреймворк для руководства API governance в контексте предприятия, акцентируя внимание на необходимости формализации ролей и процессов [16].

Вместе с тем в научной литературе отсутствует формализованное определение API Drift, охватывающее все виды расхождений (а не только spec vs. runtime), и отсутствует модель, связывающая drift с governance-пробелами. Настоящая статья восполняет этот пробел..

3. Методология исследования

Исследование выполнено методом концептуального анализа и терминологического синтеза. Методология включала следующие шаги: (а)

сбор определений API из научных и профессиональных источников с выделением семантических полей; (б) кластеризацию определений по трём ключевым аспектам (интерфейс, контракт, актив); (в) синтез интегративного рабочего определения на основе выделенных аспектов; (г) введение производных понятий «представление» и «проявление» API; (д) выведение понятия API Drift из логики рабочего определения; (е) построение фасетной классификации по принципу независимых измерительных осей.

На первом этапе проведена систематизация существующих определений API в академической и профессиональной литературе; выделены три устоявшихся взгляда на API (как интерфейс, как контракт, как управляемый актив). На втором этапе на основе синтеза этих определений сформулировано интегративное рабочее определение и введены понятия «представление API» и «проявление API». На третьем этапе из логики рабочего определения выведено понятие API Drift и предложена его фасетная классификация. Для анализа управленческого контекста drift применена четырёхкомпонентная модель API governance, описанная в работе Lauret [8].

Статья носит концептуально-терминологический характер: её цель — не эмпирическая проверка гипотез, а формирование понятийного аппарата и исследовательской рамки для последующей работы. Такой подход соответствует традиции концептуального моделирования в области информационных систем [20] и методологии design science [21].

4. Определение API: интерфейс, контракт, продукт

В современной литературе и индустриальной практике API определяется с трёх позиций, каждая из которых фиксирует отдельный аспект одного и того же объекта.

API как интерфейс. Наиболее распространённая трактовка рассматривает API как программный интерфейс, посредством которого одна система взаимодействует с другой. Данный взгляд фиксирует техническую сторону: сигнатуры вызовов, протоколы взаимодействия, форматы передаваемых данных [1][9].

API как контракт. Вторая трактовка смещает фокус с технической реализации на согласованность ожиданий между участниками взаимодействия. API описывается как формализованное соглашение между поставщиком и потребителем: при обращении определённым образом гарантируется определённый формат и семантика ответа [9][10]. Контракт в данном контексте — это соглашение о поведении, а не о реализации.

API как управляемый актив. Третья позиция, закреплённая в работе Medjaoui et al. [9] и получившая распространение в практиках Red Hat [10], Nordic APIs [11] и в академической литературе по API management [14], рассматривает API не как статичный артефакт, а как актив, подлежащий управлению на протяжении всего жизненного цикла. Данный взгляд фиксирует организационно-управленческий аспект: API имеет владельца, целевую аудиторию, стадию жизненного цикла, метрики успешности и подлежит портфельному учёту в масштабе организации. В отличие от метафоры «API как продукт», формулировка «API как управляемый актив» акцентирует не рыночную природу, а управленческую: актив необходимо инвентаризировать, отслеживать его состояние и принимать решения о его развитии или выводе из эксплуатации.

Перечисленные трактовки не противоречат друг другу, а дополняют одна другую. На основе их синтеза предлагается следующее интегративное рабочее определение:

API — это программный интерфейс, посредством которого одна система взаимодействует с другой, одновременно являющийся формализованным соглашением между поставщиком и потребителем и управляемым активом в составе API-портфеля организации.

Данное определение подводит к вопросу о том, каким образом API наблюдается и каким образом им управляют. Непосредственно абстракция API не является наблюдаемой — работа ведётся с конкретными артефактами и сигналами, которые целесообразно разделить на две категории.

5. Представления и проявления API

Для описания того, каким образом API становится доступным для наблюдения и управления, вводятся два понятия.

Представление API — это артефакт, фиксирующий API в определённом виде: код реализации, машиночитаемая спецификация (OpenAPI Specification, Protocol Buffers, AsyncAPI), документация для потребителей, запись в каталоге API. Представления создаются участниками процесса разработки и инструментами, версионируются и выступают источниками истины для различных аспектов API. Спецификация фиксирует договор, код определяет поведение, каталог описывает атрибуты API как управляемого актива.

Проявление API — это наблюдаемое состояние работающей системы: production-трафик, логи вызовов, runtime-поведение, метрики использования, трейсы. Проявления не создаются намеренно — они возникают как побочный эффект функционирования системы; их можно регистрировать, агрегировать и анализировать.

Три стороны API соотносятся с представлениями и проявлениями следующим образом:

API как интерфейс наблюдается через код реализации и production-трафик — это фактическое поведение системы.

API как контракт фиксируется через машиночитаемую спецификацию — это декларация намерений, формализованное обещание поставщика потребителю.

API как управляемый актив становится доступным для управления через каталог API, документацию, назначение владельца, фиксацию стадии жизненного цикла и отслеживание метрик использования. Проявлением данной стороны являются, в частности, adoption rate (доля потребителей, использующих данную версию API), retention (удержание потребителей при смене версий) и time-to-deprecation (время от объявления устаревшей версии до полного вывода из эксплуатации).

У каждой стороны API существуют как представления, так и проявления. Представления описывают, каким API задуман; проявления —

каким образом он функционирует в действительности. Расхождение между ними составляет сущность явления, определяемого в следующем разделе.

6. Определение API Drift

API Drift (далее — drift, расхождение, дрейф) определяется как расхождение между представлениями API либо между представлениями и проявлениями API.

Данный термин выводится как прямое следствие интегративного определения: абстракция, существующая одновременно в множестве представлений (код, спецификация, документация, запись в каталоге) и множестве проявлений (трафик, логи, метрики, трейсы), по определению допускает рассогласование между ними.

Различные участники — проектирующие, реализующие, поддерживающие и интегрирующие API — работают с различными артефактами на различных стадиях жизненного цикла. При отсутствии специальных механизмов согласования расхождения возникают неизбежно.

Для иллюстрации приведём несколько типовых примеров drift, иллюстрирующих различные виды расхождений:

Spec ↔ Code (расхождение контракта и реализации в виде кода): в OpenAPI-спецификации поле email помечено как required: true, а код эндпоинта POST /users не выполняет валидацию и пропускает запросы без email.

Spec ↔ Traffic (расхождение намерений и реального трафика): спецификация определяет эндпоинт DELETE /items/{id}, но анализ логов за месяц показывает ноль обращений; другой пример: метод, помеченный как @Deprecated, продолжает активно вызываться.

Doc ↔ Spec (расхождение документации и контракта): в документации для разработчиков на портале поле userId описано как строка, а OpenAPI-спецификация определяет его как integer.

Catalog ↔ Reality (расхождение каталога и реального состояния): в каталоге API организации актив числится на стадии «Deprecated» и имеет

владельца Петрова, но фактически продолжает обрабатывать production-трафик, а Петров покинул компанию.

Ecosystem drift (расхождение между API портфеля): API «Users» возвращает ошибку валидации в формате {«error»: «Invalid email»}, а API «Orders» — в формате {«code»: 422, «detail»: «Invalid email»}, при том что политика оформления ошибок на уровне организации едина.

В индустриальной практике термин «API drift» используется в продуктах Optic, Speakeasy, Stainless, Wiz, однако, как правило, в узком значении — как расхождение спецификации и runtime-поведения [3][4][6][7]. В настоящей работе данному термину придаётся более широкое и строгое определение, охватывающее все виды расхождений между артефактами и наблюдаемыми состояниями API.

В определение закладываются следующие принципы.

Принцип 1. Drift является явлением, а не проблемой. Утверждение о том, что расхождение представляет собой проблему, само по себе является гипотезой и требует обоснования в каждом конкретном случае. Drift может быть нейтральным, намеренным или являться следствием осознанного компромисса. Проблемой drift становится только при наличии governance-пробела, делающего расхождение неуправляемым (см. раздел 8).

Принцип 2. Drift существует на двух уровнях. На уровне единичного API — как расхождение между его собственными артефактами и проявлениями. На уровне экосистемы — как расхождение между различными API в одном портфеле (например, в стилях проектирования, соглашениях об именовании, форматах ошибок). Второй случай обозначается как ecosystem API drift и является частным случаем общего понятия.

Принцип 3. Drift анализируется через governance-рамку. Для ответа на вопросы «является ли данное расхождение проблемой» и «какие действия необходимы» drift необходимо соотнести с компонентами governance, что рассматривается в разделе 8..

7. Фасетная классификация API Drift

Для систематической работы с расхождениями предлагается фасетная классификация, в которой каждый наблюдаемый drift описывается набором независимых признаков, а не единственной позицией в иерархии. Каждый фасет отвечает на определённый тип вопроса (табл. 1): онтологический (что это), структурный (что именно расходится), атрибутивный (как себя ведёт), управленческий (где именно сломалось), функциональный (к чему приводит).

Таблица 1. Фасетная классификация API Drift.

Фасет	**Значения**	**Принцип деления**
Уровень	single-API / ecosystem	онтологический
Что расходится	конкретная пара A ↔ B (spec↔code, spec↔traffic, doc↔spec, catalog↔reality, naming-style-across-APIs, …)	структурный
Характер	управляемый / неуправляемый	атрибутивный
Governance-gap'ы	policy / process / institution / indicator	управленческий
Последствия	security / reliability / APX-DX / cost / compliance / none	функциональный

Фасеты являются независимыми: один drift может одновременно относиться к ecosystem-уровню, быть управляемым, иметь пробел в метриках и оказывать влияние на API experience (APX) и developer experience (DX). Такой подход обеспечивает более точное описание расхождений по сравнению с иерархической классификацией и допускает комбинаторное расширение по мере появления новых типов артефактов и проявлений.

8. Анализ drift через governance-рамку

Для определения управленческой значимости обнаруженного drift предлагается его разложение по четырём компонентам API governance [8].

Политики. Существует ли формализованное правило (политика), которое нарушает данное расхождение? При отсутствии такого правила drift не может быть квалифицирован как нарушение — лишь как отклонение, требующее оценки.

Процессы. Какой процесс должен был предотвратить возникновение drift или обеспечить его раннее обнаружение? Существует ли он? Если существует — по какой причине не сработал?

Институты. Кто является владельцем API? Кто отвечает за его поддержку и развитие? Кто несёт ответственность за соответствие общим правилам на уровне портфеля?

Индикаторы (метрики). Измеряется ли drift количественно? Каким образом? При отсутствии метрик невозможно судить ни об улучшении или ухудшении ситуации, ни о влиянии drift на продуктовые показатели.

Отсутствие ответа по любому из перечисленных компонентов квалифицируется как governance-пробел (governance gap). Именно такие пробелы, а не drift сам по себе, становятся предметом управленческой работы команды.

Данный подход операционализирует принцип 1. Если расхождение является управляемым, governance-пробелы отсутствуют, а последствия оцениваются как приемлемые — drift фиксируется как осознанное решение и не требует корректирующих действий. Если же обнаруживается хотя бы один пробел — появляется конкретное направление governance-работы, независимо от субъективной оценки «опасности» самого расхождения.

9. Заключение

В статье введено расширенное определение понятия API Drift, охватывающее расхождения как между различными представлениями API (спецификация, код, документация, каталог), так и между представлениями и проявлениями (production-трафик, логи, метрики). Предложена фасетная классификация по пяти независимым измерениям: уровень, пара

расходящихся артефактов, характер управляемости, governance-пробелы, последствия. Обоснован принцип: drift сам по себе не является проблемой — он становится предметом управления только при обнаружении пробела хотя бы в одном из четырёх компонентов governance.

Практическая значимость работы состоит в формировании словаря и аналитической рамки, которые могут использоваться командами разработки и архитекторами для систематической оценки расхождений в API-портфеле. Предложенная фасетная классификация позволяет описывать drift с точностью, достаточной для принятия управленческих решений.

Дальнейшие направления исследования: (1) разработка количественных метрик для измерения различных типов drift; (2) эмпирическое исследование влияния drift на developer experience и API experience; (3) процессное моделирование возникновения drift на различных стадиях жизненного цикла API; (4) валидация предложенной классификации на реальных API-портфелях промышленных организаций.

Литература

1. Haupt F., Leymann F. Vukojevic-Haupt K. API governance support through the structural analysis of REST APIs // Computer Science. – 2018. – № 3. – P. 291–303. – DOI: 10.1007/s00450-017-0384-1.
2. Gamez-Diaz A., et al. Gamez-Diaz A. et al. API Governance at Scale // Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24). – 2024. – DOI: 10.1145/3639477.3639713.
3. Anoop S. API Drift Detection Enhancing Data Protection // International Journal of Computer Applications. – 2024. – DOI: 10.5120/ijca2024924168.
4. What is API drift and how do you prevent it? [Электронный ресурс]. // Wiz. – URL: https://www.wiz.io/academy/api-security/api-drift (дата обращения: 15.06.2025).
5. Krintz C., Jayathilaka H., Dimopoulos S., Pucher A., Wolski R., Bultan T. Cloud Platform Support for API Governance // Proceedings of the 2014 IEEE

International Conference on Cloud Engineering (IC2E). – 2014. – P. 615–618. – DOI: 10.1109/IC2E.2014.30.

6. API Drift Detection [Электронный ресурс]. // APIscene. – URL: https://www.apiscene.io/api-security-identity/api-drift-detection/ (дата обращения: 15.06.2025).
7. API Drift White Paper [Электронный ресурс]. // APIContext. – URL: https://apicontext.com/resources/api-drift-white-paper/ (дата обращения: 15.06.2025).
8. Lauret A. The 4 components of API governance [Электронный ресурс]. // API Handyman. – URL: https://apihandyman.io/the-4-components-of-api-governance/ (дата обращения: 15.06.2025).
9. Medjaoui M., Wilde E., Mitra R., Amundsen M. Continuous API Management: Making the Right Decisions in an Evolving Landscape. – 2nd ed. – Sebastopol: O'Reilly Media, 2022. – 300 P.
10. Full API lifecycle management: A primer [Электронный ресурс]. // Red Hat Developer. – URL: https://developers.redhat.com/blog/2019/02/25/full-lifecycle-api-management/ (дата обращения: 15.06.2025).
11. API as a Product [Электронный ресурс]. // Nordic APIs. – URL: https://nordicapis.com/wp-content/uploads/API-as-a-Product-v2.1.pdf (дата обращения: 15.06.2025).
12. Anthony E., Berntsson A., Santilli T., Wohlrab R. We're Drifting Apart: Architectural Drift from the Developers' Perspective // Proceedings of the 21st IEEE International Conference on Software Architecture (ICSA 2024). – 2024. – P. 101–111. – DOI: 10.1109/ICSA59870.2024.00018.
13. Ali N., Baker S. O'Crowley R., Herold S., Buckley J. Architecture consistency: State of the practice, challenges and requirements // Empirical Software Engineering. – 2018. – P. 224–258. – DOI: 10.1007/s10664-017-9515-3.
14. Overeem M., Mathijssen M., Jansen S. API-m-FAMM: A focus area maturity model for API Management // Information and Software Technology. – 2022. – T. 106890. – DOI: 10.1016/j.infsof.2022.106890.

15. Suescún Monsalve E., Tabares Betancur M., González Palacio L., Vásquez Escobar M. Towards an API Governance Model: Systematic Mapping of the Literature // Perspectives Scientific Journal. – 2024. – № 2. – DOI: 10.47187/perspectivas.6.2.219.
16. Göransson K. A Framework for Guidance of API Governance: Master's Thesis. – University of Gothenburg, 2019. – 78 P.
17. Detect Application Architecture Drift Early in the SDLC [Электронный ресурс]. // Apiiro. – URL: https://apiiro.com/blog/detect-application-architecture-drift-early-in-the-sdlc/ (дата обращения: 15.06.2025).
18. Serbout S., Di Lauro F., Pautasso C. Web APIs Structures and Data Models Analysis // Proceedings of the 19th IEEE International Conference on Software Architecture (ICSA 2022). – 2022. – DOI: 10.1109/ICSA53651.2022.00012.
19. Pautasso C., Serbout S. Towards Large-Scale Empirical Assessment of Web APIs Evolution // Web Engineering (ICWE 2021). – 2021. – P. 123–138. – DOI: 10.1007/978-3-030-74296-6_10.
20. W, Y., Weber R. Research commentary: Information systems and conceptual modeling – a research agenda // Information Systems Research. – 2002. – № 4. – P. 363–376. – DOI: 10.1287/isre.13.4.363.69.
21. Hevner A.R., March S.T., Park J., Ram S. Design science in information systems research // MIS Quarterly. – 2004. – № 1. – P. 75–105. – DOI: 10.2307/25148625.