Регламент как код: почему Notion-инструкция никогда не становится операционной политикой

2026-05-13

Регламент как код: почему Notion-инструкция никогда не становится операционной политикой

В любой B2B-команде размером от двадцати человек есть Notion-страница «Как мы работаем». Её открывают на онбординге, проходят по диагонали, ставят галочку и больше не возвращаются. Через шесть месяцев процесс уже не такой, как на странице, а через год расхождение становится системным: новые сотрудники задают одни и те же вопросы в чате, потому что страница врёт, а никто этого не отметил. В материале PagerDuty, посвящённом определению runbook описывается типовой паттерн: команда без исполняемых runbook-ов в инциденте тратит первые минуты не на устранение проблемы, а на восстановление того, кто что должен делать. Регламент существует, но не управляет ничем.

За этой бытовой картиной стоит структурная проблема. Корпоративный регламент в виде документа — это описание мира как он должен быть. Операционная политика — это правило принятия решения в конкретной ситуации с конкретными входными данными. Между ними не косметическая разница, а категориальная. Документ обращается к человеку и предполагает, что человек его прочитает, запомнит и применит. Политика обращается к системе и предполагает, что система её соблюдает, проверяет и фиксирует исключения. Первый формат тиражируем через копипасту. Второй — версионируем, исполняем и аудируем.

Этот сдвиг выходит из узкого инженерного сегмента в массовую практику, а не остаётся эксклюзивом крупного инженерного бизнеса. Дисциплина policy-as-code, выросшая из инфраструктурного мира, доросла до уровня зрелости, при котором её можно применять к управленческим решениям. Архитектура памяти агентов даёт примитив для хранения граф-структуры этих решений. Управляемые среды исполнения для крупных языковых моделей позволяют использовать эти модели как ограниченных по политике исполнителей, а не как свободных текстогенераторов. Эти три слоя — формальные правила в коде, графовая память, контролируемая среда исполнения ИИ-моделей — совпадают к тому моменту, когда скорость принятия решений становится критичной величиной, и документ-регламент перестаёт быть первичным артефактом.

Почему регламент как документ не работает?

Документ-регламент страдает теми же тремя дефектами, что любой документ-как-источник-истины, но с операционными последствиями, которые не списываются на «устарело — обновим».

Первый дефект — **разрыв между описанием и решением**. Регламент описывает шаги, политика описывает условие. Страница «как мы квалифицируем входящего лида» в Notion перечисляет пять шагов; в живой работе менеджер принимает решение «передавать ли в отдел продаж или закрывать как мусор» по семи признакам, четыре из которых не описаны нигде. Эти четыре признака — реальная политика компании. Они живут в голове у трёх старших менеджеров и передаются на онбординге устно. При уходе одного из них часть политики исчезает, и об этом узнают по росту числа жалоб.

Второй дефект — **отсутствие исполнителя**. У документа нет того, кто его соблюдает. Считается, что соблюдает человек, но человек принимает решение в условиях усталости, цейтнота, неполной информации и легко конкурирующих интерпретаций. В инженерной культуре эта проблема решена давно: ни одна команда уровня Netflix или Stripe не полагается на то, что инженер «вспомнит политику безопасности». Политика выражена кодом, валидатор отвергает деплой, нарушающий её, и инженер физически не может проигнорировать правило. В операционных отделах аналогичной дисциплины почти не существует — за исключением узкого сегмента финансовых рабочих процессов, где её требует регулятор.

Третий дефект — **отсутствие истории**. Регламент в Notion не помнит, когда и почему правило изменилось. Эта же проблема в инженерном мире решалась в последние 15 лет через дисциплину commit-сообщений и ADR-документов, фиксирующих контекст решения в момент его принятия. Версионная история страницы есть, но она показывает diff текста, а не контекст решения. На вопрос «почему мы перестали закрывать сделки в пятницу» страница ответит словом «потому что мы решили так делать», а реальная причина — конкретный провалившийся релиз шесть месяцев назад — не зафиксирована нигде, кроме как в памяти трёх человек. При смене этих троих причина исчезает, правило остаётся как карго-культ, и через два года кто-то его молча отменяет, не зная истории.

Эти три дефекта не лечатся «лучшим Notion» по той же причине, по которой не лечится документ как примитив знания в целом: они вшиты в природу документа как акта рефлексии после события. Регламент пишется тогда, когда уже принято решение, как должно быть. Политика работает тогда, когда решение ещё принимается. Чтобы перевести одно в другое, нужно изменить единицу — со страницы на правило-в-исполнении.

Что такое исполняемая политика как артефакт

Исполняемая политика (executable policy) — это правило принятия решения, выраженное в формальном языке, которое исполняется средой исполнения

и порождает аудиторский след с временем, актором, входными данными и результатом. Не текст про правило, а само правило, доступное для машинной проверки.

Разница укладывается в три практических критерия, которые можно проверить в любой команде:

Критерий	Регламент в Notion	Исполняемая политика
Что описывает	Мир как должно быть	Правило принятия решения
К кому обращается	К человеку (читает + запоминает)	К системе (проверяет + фиксирует)
Исполнение	Надежда на дисциплину	Машинное принуждение с эскалацией
История изменений	Diff текста страницы	Контекст решения с временем и автором
Аудит	Ручной сэмплинг	Автоматический, 100% решений

Первые три строки — категориальные; последние две — экономические и как раз объясняют, почему первые три выливаются в операционную бесполезность документа.

В инженерном мире эта концепция реализована в проекте Open Policy Agent (OPA), который принят в CNCF на уровне graduated в феврале 2021 года, через 5 лет после запуска проекта в 2016 году и используется крупными технологическими компаниями для проверки облачных конфигураций, прав доступа Kubernetes и контрактов API. OPA задаёт язык Rego, в котором правило описывается как «при таких входных данных вернуть такое решение». Правило живёт в репозитории на GitHub рядом с тестами, проходит ревью пулл-реквестом и развёртывается как любой код. У этой архитектуры есть конкретные свойства, которые отсутствуют у Notion-страницы: правило проверяется автоматически на каждом релевантном событии, его нарушение блокируется или эскалируется, история изменений живёт в коммитах с обязательным описанием.

Тот же подход в виде Sentinel у HashiCorp применён к управлению инфраструктурой: прежде чем Terraform создаёт ресурсы, набор политик проверяет, не нарушает ли изменение требования безопасности, расходов или соответствия. Решение «можно ли развернуть эту конфигурацию» принимается не инженером и не страницей в Confluence, а исполняемой политикой с аудиторским следом.

В операционных отделах того же уровня дисциплины почти нет. Не потому что задача отличается принципиально, а потому что не было дешёвого

способа описывать управленческие правила в формальном виде, доступном для машинной проверки. Управленческое правило обычно содержит ссылки на свободный текст: «эскалировать клиенту с признаками недовольства», «закрывать сделку, если шансы низкие». Раньше эти фразы было нельзя вычислить иначе, как заставить человека читать и решать. Сейчас крупные языковые модели делают этот шаг технически возможным — но именно как исполнители политики, а не как авторы.

Граф решений как форма политики

Линейный список правил перестаёт работать, как только правил становится больше 30–40. Зависимости между ними не описываются плоской таблицей: «эскалировать инцидент» зависит от типа клиента, тарифа, истории взаимодействия, текущей загрузки команды и времени суток. Это не таблица, это граф. И именно как граф решений политика становится управляемым артефактом.

Графовые системы для агентной памяти, такие как Graphiti — открытый временной граф знаний, решают близкую задачу: фиксируют сущности, отношения и время их валидности. Тот же примитив применим к политике. Узел графа — это правило с условием и результатом. Ребро — это зависимость одного правила от другого. Двойная метка времени, как у Graphiti, отвечает на вопрос «с какого момента это правило действует и когда оно перестало действовать». Изменение правила не затирает старое — добавляет новый узел с новыми границами валидности. На вопрос «как мы решали этот случай в марте» система отвечает не догадкой, а извлечённым состоянием графа на марте.

Этот сдвиг меняет саму единицу, с которой работает регламент. Документ остаётся — но как проекция графа, рендер для конкретного читателя в конкретный момент. Онбординг новичка читает «текущее состояние политики на сегодня» как сгенерированный из графа документ; аудит получает diff между двумя моментами; владелец процесса работает не с текстом, а с самими узлами и ребрами. Документ перестаёт быть источником и становится одной из возможных поверхностей чтения.

Где это уже работает в управленческой плоскости?

Узкий, но показательный сегмент, где исполняемая политика управляет операцией — финансовые рабочие процессы в крупных компаниях. Правила соответствия (compliance) описаны не страницей в Confluence, а исполняемыми правилами в системах вроде ServiceNow или внутренних оркестраторах. Заявка на договор проходит через граф проверок, каждая из которых имеет аудиторский след; отклонение фиксируется как событие; политика версионируется и пересматривается раз в квартал. Это работает по принуждению регулятора — но архитектурно то же самое применимо к любому операционному процессу, где решения принимаются часто: скидка выше порога без согласования начальника — такое же policy rule, как порог выделения ресурсов в Kubernetes-кластере.

Второй кейс — инцидент-менеджмент в зрелых технических командах. Норма последних лет, отражённая в операционных руководствах по runbook-практикам инцидент-менеджмента, сводится к одному требованию: runbook проходит путь от описательного документа к исполняемому артефакту — связанным с триггерами, выполняемым системой и эскалирующим человеку только то, что требует решения. Шаги не пишутся в свободной форме — они описываются как набор автоматизированных действий с явными условиями перехода. Регламент инцидента превращается в исполняемый граф.

Третий сегмент — корпоративные политики использования крупных языковых моделей. Модель контекстного протокола (Model Context Protocol) от Anthropic сам по себе не является policy-framework — это протокол подключения модели к источникам данных и инструментам. Но он создаёт точку, в которой политика может сработать: какие серверы разрешены конкретному агенту, какие действия он может выполнять, какие фильтры наложены на данные. Именно эта слойность — протокол внизу, исполняемые правила вверху — превращает корпоративный вопрос «по каким правилам наш агент имеет право действовать» из теоретического в инженерный.

Объединяет эти три сегмента одно: в каждом из них регламент в виде документа физически не справляется со скоростью и плотностью операции. Документ работает, пока решений мало и они принимаются медленно. Когда решений тысячи в сутки и они должны приниматься за секунды, документ исчезает из контура — либо как формальный артефакт без операционной нагрузки, либо целиком уступая место исполняемой политике.

Что меняется для трёх типов читателей

Для основателя на стадии 15–50 человек тест простой. Возьмите три решения, которые ваша команда принимает чаще всего — квалификацию лида, эскалацию клиента, согласование скидки — и спросите, где живёт правило. Если ответ «у нас есть страница в Notion», откройте её и сравните с тем, как реально решает старший менеджер. Если расхождение очевидно после трёх минут разговора — у вас нет операционной политики, у вас её описание шестимесячной давности. Конкретное действие: возьмите одно из этих правил и опишите его как граф — узлы условий, узлы результатов, явные исключения. Это уже политика, даже если граф пока живёт в одной таблице. Дальше — выбор между человеческим исполнением и автоматическим — становится инженерной задачей, а не управленческим спором.

Для руководителя операционного блока полезен другой вопрос: какой процент решений в вашем отделе принимается «по практике», а не «по регламенту». Если ответ выше 20% — у вас есть устная политика, которой нет в письменном виде. Этот сегмент несёт двойной риск: уход носителя практики обнуляет часть политики, и аудит не может проверить её исполнение. Перевод этого сегмента в исполняемые правила имеет более высокую отдачу, чем редактура существующих документов. Начинать стоит с правил, у которых уже есть ав-

томатический триггер — то есть с тех, где наступление условия фиксируется системой, а не человеком.

Для инженера, оценивающего проект, тест третий: посмотрите, есть ли в продукте отдельный артефакт «policy» — репозиторий, рендер графа, версионная история правил с описанием контекста изменений. Если есть и он живёт как код — продукт устойчив к смене людей: правила переживают уход носителя практики, и накопленный опыт каждой итерации остаётся в системе. Если политика живёт в свободном тексте промптов и инструкций для команды поддержки — любая смена ключевого менеджера переписывает половину поведения системы, и проектная работа не накапливается.

На что мы будем смотреть дальше

Если тезис верен, в течение ближайших 12–18 месяцев в управленческом сегменте должны появиться три явления. Первое — публичные открытые наборы политик для типовых операционных решений по отраслям, аналог того, как для инженерного мира появились библиотеки готовых правил OPA для Kubernetes и облаков. Сейчас каждый бизнес пишет свою политику квалификации лида с нуля; первая попытка стандартизации станет сигналом зрелости рынка. Второе — появление продуктов, где граф политики является самостоятельным артефактом, а не модулем поверх привычного офисного стека. Это другой вид инструмента, а не «Notion с автоматизацией». Третье — рост сегмента аудита и переноса политик: когда правила компании выражены как граф, миграция между поставщиками операционных систем становится не «выгрузить документы», а «перенести граф». Появление переносимых форматов для управленческой политики скажет, что сегмент перестал быть привязан к одному вендору.

Регламент как документ проживёт долго — у него есть инерция и чувство контроля у руководителей, привыкших работать с текстом. Но в той части бизнеса, где операционная скорость определяет экономику, документ уже не первичен — его место занимает исполняемая политика как самостоятельный артефакт. Первым сигналом будет не публичный манифест, а появление открытых библиотек операционных политик в одной из плотных отраслей — так, как это произошло в инженерной дисциплине, когда общие Kubernetes-политики внезапно обнаружились на GitHub в виде воспроизводимых наборов, а не внутренних wiki-страниц. Какой сегмент сделает этот шаг первым — открытый вопрос; ответ на него заметен только постфактум, по появлению переносимых форматов между вендорами.

Главное

- Корпоративный регламент в виде Notion-страницы описывает мир как должно быть; операционная политика описывает правило принятия решения в конкретной ситуации. Это два разных артефакта, и первый никогда не становится вторым автоматически.

- Документ-регламент имеет три структурных дефекта: разрыв между описанием и решением, отсутствие исполнителя, отсутствие истории контекста. Все три не лечатся «лучшим Notion».
- Исполняемая политика — это правило в формальном виде, которое исполняется средой исполнения с аудиторским следом. В инженерной плоскости это реализовано через policy-as-code (OPA, Sentinel); в управленческой — пока узко, но архитектурно готово.
- Граф решений с двойной меткой времени заменяет линейный список правил. Документ остаётся как проекция графа, а не как источник истины.
- Сдвиг наблюдается там, где скорость операции превышает скорость ручной интерпретации регламента: финансовое соответствие, инцидент-менеджмент, политики использования агентов. Первый отраслевой сегмент, где это станет нормой, окажет большее влияние на рынок операционных инструментов, чем любой обобщённый прогноз в тех же границах.

FAQ

Чем исполняемая политика отличается от хорошо написанного регламента? Регламент описывает, как должно быть и предполагает, что человек его прочитает и применит. Политика выражена в формальном языке, исполняется средой и фиксирует каждое решение с временем и входными данными. Разница не в качестве текста, а в том, кто является исполнителем.

Нужно ли переписывать все Notion-страницы в исполняемый вид? Нет. Документы остаются полезными как референс и обучающий материал. Переводить в исполняемый вид нужно те правила, где решение принимается часто (10+ раз в день), быстро (быстрее 60 секунд на решение) и имеет проверяемый результат. На практике это не все сотни внутренних регламентов, а узкий поднабор «горячих» правил, вокруг которых разбивается основное операционное время команды.

Как понять, что у нас есть ±устная политика», не отражённая нигде? Простой тест: выберите 5 частых решений (эскалации, скидки, обработка жалоб, приоритизация заявок) и попросите двух старших менеджеров отдельно описать, как они принимаются. Если расхождение выше 30% хотя бы по 2 из 5 — у вас есть устная политика, и это норма, а не исключение.

Как это совместимо с ИИ-агентами? Исполняемая политика отвечает на вопрос «что агенту разрешено делать в этом контексте» — без неё агент либо работает в свободном режиме и иногда уходит в поведение, которое никто в компании не планировал, либо блокируется жёсткими входными фильтрами и перестаёт быть полезным. Рабочая промежуточная форма — явный граф политики, в котором агент является одним из исполнителей, а не единственным носителем правил.

Где живёт это в бизнес-модели? Соседний сюжет — в разборе service-as-software: когда выручка билингуется за исход, а не за лицензию, исполняемая

политика становится опорной точкой, в которой провайдер отвечает за SLA, а не за факт доступа.