Cron больше project: расписание как главный операционный артефакт AI-native компании

2026-04-29

Cron больше project: расписание как главный операционный артефакт AI-native компании

Что показывает crontab команды из трёх человек

В одной AI-native команде, наблюдаемой с весны 2026, GitHub Issues открывают примерно раз в неделю. Crontab смотрят каждый день. Расписание содержит 23 активные записи: проверка воронки в 9:00, ночной digest research в 02:00, health-check агентов каждые два часа, еженедельный отчёт по knowledge base в понедельник в 7:30. Полный список покрывает примерно 80% работы, которая физически происходит в компании за неделю.

Project-доска у той же команды содержит шесть активных карточек. Из них три не двигаются больше двух недель. Это не проблема дисциплины и не «надо подтянуть процессы». Это структурный сдвиг: дискретные задачи перестали быть основной единицей операции. Основной единицей стало расписание.

Почему roadmap перестал отражать реальность

В классической компании roadmap — это карта работы. У каждого проекта есть начало, конец, owner, definition of done. Если хочется понять, чем занят бизнес, достаточно посмотреть в Jira: список карточек примерно равен списку текущей работы.

В AI-native компании эта карта систематически врёт в сторону недооценки. Большая часть операции выполняется не как набор тикетов, а как набор повторяющихся циклов, которые запускаются сами и завершаются сами. Никто не открывает карточку «прогнать research-дайджест за вчера» — эту работу делает агент по расписанию. Никто не назначает owner для «проверить здоровье воронки» — owner это `0 9 * * *`. Сторонний наблюдатель видит шесть карточек и делает неправильный вывод: «команда мало делает». А команда за то же время выполнила 140 routine-запусков, из которых 12 эскалировали человеческое вмешательство.

Этот разрыв растёт по мере того, как managed harness становится стандартной частью стека. Платформы агентов — публичные материалы Anthropic Engineering, анонс OpenAI про новые инструменты для агентов, AWS Bedrock Agents — сводят стоимость запуска нового routine к конфигу. Это значит, что центр операционной массы быстро смещается от «человек делает задачу» к «расписание запускает агента, человек смотрит исключения». При таком рас-

пределении roadmap перестаёт быть основной картой работы — он становится картой исключений.

Похожий сдвиг уже случался в data-инженерии. dbt-core и Apache Airflow превратили аналитику из «возьмите тикет, напишите запрос» в «определите модель, она пересчитывается по расписанию». Граф зависимостей расписания (directed acyclic graph, dag) стал документом, который точнее описывает работу data-команды, чем Jira-проект. AI-native организация повторяет эту траекторию на уровне всей компании, не только аналитики.

Что меняется, когда работа становится расписанием?

Самый короткий способ почувствовать сдвиг — выписать рядом две колонки: как описывается работа в проектной парадигме и как — в парадигме расписания. Различия не косметические; они затрагивают всё, начиная от найма и заканчивая отчётом инвестору.

Аспект	Project-парадигма	Cron-парадигма
Единица работы	Дискретный тикет с началом и концом	Повторяющийся routine с триггером
Главный артефакт	Roadmap, Jira-доска	Crontab, scheduler, реестр routines
Вопрос для статуса	«Что закрыли на этой неделе?»	«Что запускалось и что эскалировало?»
Метрика прогресса	Velocity, story points	Health rate, escalation rate, freshness
Owner	Человек на тикете	Человек на routine + сам routine
Что показывают инвестору	Список будущих фич	Список ежедневной автоматизированной работы
Главный риск	Срыв сроков	Тихая деградация без алерта
Управленческий ритуал	Sprint planning, ретро	Ревью расписания, аудит cron-debt

Эта таблица не означает, что одна парадигма «лучше» другой. У них разные первичные объекты. В компании, где 80% работы — повторяющиеся циклы, описывать её через project-доску — то же самое, что описывать работу аналитической команды через тикеты «написать SQL»: формально можно, операционно — теряет половину сигнала.

Где живёт работа в AI-native компании

Если открыть условный crontab AI-native команды, он распадается на три слоя.

Первый слой — **observation routines**. Циклы, которые периодически снимают состояние мира: воронка, sales-звонки за сутки, состояние клиентского контура, свежие сигналы из research, ошибки в продакшене. Они не производят решений, они производят структурированный snapshot. Этот snapshot потом потребляется либо человеком, либо следующим routine. У наблюдаемой команды на этот слой приходится примерно треть записей crontab.

Второй слой — **decision routines**. Циклы, в которых агент применяет SOP: триггерит follow-up, эскалирует зависшие тикеты, помечает stale-документы, переоценивает приоритеты задач. Это та работа, которая в классической компании размазана по людям и редко документируется явно. В AI-native варианте она кодифицирована — потому что иначе её не запустить по расписанию. Здесь живёт основной operating layer компании, тот самый, про который мы писали в заметке про harness как commodity: institutional SOP перестаёт быть документом и становится исполняемой политикой.

Третий слой — **maintenance routines**. Health checks, проверки целостности knowledge base, ротация логов, аудиты broken links, перепроверка устаревшего research. Этот слой невидим для бизнеса, но его отсутствие проявляется через два-три месяца как тихая деградация: dashboard начинает показывать неправду, агенты ссылаются на исчезнувшие документы, freshness падает. Когда речь идёт про cockpit, который «не должен врать», именно эти routine отвечают за то, чтобы он не врал.

Все три слоя имеют общее свойство: они описываются расписанием. Не «когда сделаем», а «когда запускается». Project-логика «у задачи есть начало и конец» к ним применяется плохо. Их жизнь — это не trajectory от старта к финишу, а пульс. Поэтому crontab точнее описывает, что делает компания, чем roadmap.

Почему расписание — это политический документ

В классической компании главный политический документ — roadmap. Он отвечает на вопрос «что мы делаем в этом квартале» и согласовывается днями, потому что от него зависит, что инвестор увидит на следующей встрече и что команда будет делать в понедельник. В AI-native компании главный политический документ — расписание.

Каждая cron-запись — это явное утверждение: «эта работа достаточно важна, чтобы запускать её каждый день, час или неделю автоматически, даже если никто не просил». Добавление записи — это решение примерно того же веса, что найм человека: запись будет работать, потреблять ресурсы и производить выводы, пока её явно не выключат. Удаление записи — это решение примерно того же веса, что увольнение: что-то, что компания делала каждый день, перестаёт делаться, и последствия проявятся через недели.

Из этого следует операционная гигиена, которая в классической компании не нужна, а в AI-native — критична. Каждая запись в crontab должна иметь явного owner-человека, документированную цель, явный output (куда пишет-

ся результат) и явные условия деактивации. В наблюдаемой команде такой реестр живёт как отдельный документ; без него за полгода накапливается «cron-debt» — записи, про которые никто уже не помнит, зачем они нужны, но они продолжают что-то писать в файлы и иногда что-то ломать.

Переход от прототипов к продакшен-системам в агентной индустрии — это переход к надёжности и наблюдаемости, не к новым фичам. Применительно к агентам продакшен — это надёжное расписание плюс мониторинг исключений. AI-native организация делает то же самое на уровне организационного дизайна: расписание становится first-class объектом управления, а не служебной деталью.

Почему cron, а не очереди событий?

Резонный вопрос: если работа повторяется, почему не описать её как реактивную систему — очереди, события, webhooks? Событийная архитектура отлично работает там, где есть внешний триггер: пришёл клиент, изменился документ, упал сервис. Но большая часть operating layer AI-native компании запускается не от внешнего события, а от внутренней необходимости периодически проверять состояние мира. Никто не присылает webhook «пора аудитнуть knowledge base». Эти routine инициируются временем, не данными.

Как отмечает Andrej Karpathy в публичных выступлениях про операционные особенности агентов:

> «Большинство интересной работы агента — это не реакция на пользователя, а его собственный внутренний цикл размышления и проверки.» — Andrej Karpathy, ex-Director of AI, Tesla

Если этот цикл живёт в коде, его естественная форма — расписание. Очереди событий хороши как транспорт между routine, но не как замена самим routine. На практике AI-native стек обычно сочетает обе модели: cron инициирует «такт сердца», очереди и durable execution платформы вроде Temporal обеспечивают надёжность отдельных шагов внутри. Расписание задаёт ритм; очереди — связность.

Event log показывает, что произошло; crontab — что компания делает всегда.

Как отличить этот сдвиг от обычного DevOps?

Существует контраргумент: «scheduled jobs всегда были, это просто DevOps в новой обёртке». Контраргумент верен наполовину. Технически — ничего нового. Концептуально — отличие в статусе и составе того, что попадает в расписание.

В классическом девопс-сценарии в crontab живут служебные задачи: ротация логов, бэкап БД, health checks инфраструктуры. Это «системные функции», их пишут инженеры по надёжности, они редко обсуждаются на продуктовых син-

хронах. Routine ничего не решает за бизнес; они поддерживают существование системы.

В AI-native сценарии в расписание попадает бизнес-логика: оценка лидов, follow-up клиентов, дайджест research, апдейт cockpit, аудит SOP. Это уже не системная функция, а часть продукта. Owner таких routine — не SRE, а тот же человек, который раньше владел соответствующим бизнес-процессом вручную. Расписание становится местом, где живёт «как мы делаем бизнес», а не «как мы поддерживаем сервер».

Эту разницу хорошо описывают практики наблюдаемости агентских систем. Как пишет Charity Majors в блоге Honeycomb, наблюдаемость нужна не за инфраструктурой, а за поведением системы — и в агентной операции это означает наблюдаемость за тем, что routine реально делает с бизнесом, а не только за тем, что процесс не упал. Тот же сдвиг описан и в материалах Мартина Фаулера про эволюционную архитектуру и в практиках SRE из Google SRE Book: когда поведение системы определяется не одним релизом, а постоянным фоном изменений, главным артефактом становится механизм этих изменений, а не их слепок.

Как говорит Charity Majors:

> «Observability нужна, чтобы задавать системе новые вопросы в продакшене, а не подтверждать заранее известные.» — Charity Majors, CTO, Honeycomb

Перенося это на нашу задачу: cron в AI-native компании — это не «папка с скриптами», а механизм, через который компания задаёт миру свои вопросы каждое утро. Поэтому он и становится политическим, а не служебным.

Что это меняет для трёх типов читателей

Фаундер. Если на питче инвестор спрашивает «расскажите про roadmap», полезный второй ответ — «вот наш crontab». Не вместо, а вместе. Разговор сдвигается от будущих обещаний к тому, что компания делает каждый день уже сейчас и где живут точки контроля. Тестовый вопрос: можете ли вы за 60 секунд показать список из 10–20 routine с явным owner и явным output? Если ответ «у нас всё в Jira» — компания операционно не AI-native, что бы ни было написано на сайте.

Операционный руководитель. Главная управленческая работа в AI-native компании — не приоритизация бэклога, а ревью расписания. Раз в две недели — пройти по crontab или scheduler, отметить каждую запись как «оставляем», «удаляем», «меняем триггер», «передаём другому owner». Это занимает 30–60 минут и заменяет несколько часов sprint planning. Тестовый вопрос: знаете ли вы, какая запись в расписании за последние два месяца ни разу не привела к человеческому действию? Если знаете — это либо ценная защита от ложно-

отрицательного, либо мёртвый routine, который пора удалять. Если не знаете — у вас нет наблюдаемости над собственной операцией.

Инженер. Большая часть кода в AI-native компании — это не features в продукте, а routines: маленькие, надёжные, с понятным контрактом ввода-вывода, с idempotency и с graceful degradation. Это близко к парадигме data-инженерии, далеко от парадигмы веб-разработки. Если выбираете команду по техническому стеку, проверьте, как у них устроен scheduler, есть ли наблюдаемость за выполнением routine, как они откатывают сломанный cron, и сколько уходит времени от «нужен новый routine» до «он в проде и работает». Хороший ответ — часы. Плохой — недели. Это часть representation layer, которую AI-native команда умеет строить, а классическая — нет.

На какие сигналы смотреть дальше

Несколько публичных индикаторов покажут, насколько сдвиг становится мейнстримным. Первый — появление в основных PM-инструментах (Linear, Jira, Asana) явной поддержки «recurring agent task» как отдельного типа сущности, наравне с issue и project. Когда станет встроенной концепцией, формализация «cron как объект управления» закрепится. Второй — публичные кейсы компаний, которые ведут открытый список своих routines как часть «about/engineering» страницы. Это AI-native эквивалент «we use Postgres and Kubernetes». Третий — рост MRR у инструментов класса orchestrator-for-agents, заточенных не под data pipelines, а под agent workflows.

Если эти сигналы придут в ближайшие 12 месяцев, операционный язык индустрии успеет перестроиться. Позже — фаундеры, которые уже строят компанию вокруг расписания, получат окно фор-старта. Главный вопрос один: что у вас в crontab завтра в 9 утра, и почему именно это.

Главное

- В AI-native компании основная единица работы — повторяющийся routine, а не дискретная задача. Roadmap отражает исключения; cron отражает реальность.
- Crontab распадается на три слоя: observation, decision, maintenance. Все три описываются расписанием, не сроками.
- Расписание — политический документ. Добавление записи — это решение масштаба найма; удаление — масштаба увольнения.
- Управленческая работа смещается от приоритизации бэклога к регулярному ревью расписания. Без этого появляется cron-debt.
- Crontab точнее показывает, чем компания реально занимается каждый день, чем любой roadmap или sprint-доска.

FAQ

Это значит, что project-менеджмент больше не нужен в AI-native компании?

Нужен, но в другой форме. Проекты остаются для дискретных инициатив с началом и концом — запуск нового продукта, миграция, исследование вертикали. Просто проекты больше не описывают всю работу. Они описывают исключения над расписанием. Соотношение в наблюдаемых командах — примерно 20% работы в проектах, 80% в routines.

Что если бизнес по природе плохо ложится на расписание — например, консалтинг или редкие сделки?

Тогда часть работы остаётся в проектах, как и раньше. Но даже в таких бизнесах observation- и maintenance-слой обычно поддаётся cron-логике: ежедневный сбор сигналов о клиентах, еженедельный аудит pipeline, ночная синхронизация CRM. Routine не обязан охватывать core-бизнес, чтобы быть основным операционным артефактом — достаточно, чтобы он покрывал большую часть повторяющейся работы вокруг ядра.

Чем cron-задача отличается от обычного scheduled job, который и в классических компаниях есть?

Технически — ничем. Концептуально — статусом. В классической компании scheduled jobs — служебная деталь, спрятанная в DevOps. В AI-native компании расписание поднято на уровень операционного артефакта: оно документировано, обсуждается на синхронах, имеет явных owner-ов и реестр. Сдвиг — не в технологии, а в том, что считается «настоящей работой».

Как защититься от cron-debt — записей, которые никто не помнит, зачем нужны?

Минимум три практики: регулярное ревью расписания (раз в две недели достаточно), явный owner и описание для каждой записи в едином реестре, и метрика «когда запись последний раз привела к действию». Записи, которые полгода ничего не триггерили, — кандидаты на удаление. Это та же дисциплина, что в founder operating cockpit: операционный layer работает только если за ним есть health loop.

Можно ли называть компанию AI-native, если у неё в crontab пять записей и команда из 50 человек?

Дело не в количестве записей, а в том, какая доля повторяющейся работы кодифицирована и запускается без участия человека. AI-native — не маркетинговый ярлык, а поддающееся измерению свойство: какая доля операционной работы живёт в расписании, а не в людях.

Что отличает хороший routine от плохого?

Хороший routine идемпотентен (можно безопасно перезапустить), имеет явный output в наблюдаемое место (файл, dashboard, канал), имеет явный fallback при ошибке (тихо в лог, не падает на пользователя) и имеет owner-человека, который получит сигнал, если routine начнёт врать. Плохой routine — это скрипт, про

который через три месяца невозможно ответить, что он делает и куда пишет результат.