Андрей Новиков — продуктовая книга

Head of Engineering / Solution Architect
Бали, Индонезия (UTC+8, это МСК+5)

anovis@anovisoft.com · Telegram @anovis · WhatsApp +62 859 2689 1633 · anovisoft.com

Подробное приложение к трёхстраничному CV. Каждая карточка построена одинаково: в чём была проблема, что я сделал, стек, что изменилось.

Содержание

Кто я

Одиннадцать лет коммерческой разработки, семь из них в позициях Head of Engineering / Solution Architect. Моя работа стоит на стыке архитектуры и доставки: я проектирую систему, часть ядра

пишу сам и делаю так, чтобы остальное команда довела без меня в качестве узкого места.

Необычная часть моего профиля — вертикальная ширина. В одной и той же роли я отвечал за бизнес-домен, продуктовую линейку, архитектуру распределённых систем, бэкенд и слой данных, инфраструктуру и пайплайн доставки, часть фронтенда и стык с реальным трансляционным оборудованием. Большинство инженеров владеет одним-двумя пунктами из этого списка. Большинство менеджеров перестаёт трогать любой из них.

Продолжаю консультировать давнего российского клиента по запросу из пояса МСК+5, так что удалённый режим работы с Россией отлажен, а не теоретический, и ёмкость под новые проекты свободна.

Цифры

Продуктов в моей архитектурной ответственности	**9**, плюс архитектурный надзор ещё над 3
Команды	**5** в зоне ответственности: 3 в прямом ведении, 2 консультативно. **4** собрано с нуля
Люди	**16 инженеров**, менеджер и QA на каждую команду, два дизайнера на всех, свыше 20 человек в периметре
Сервисов в периметре	около **70** под Kubernetes и GitOps
Экономика платформы	новые пакеты фич ~**2×** дешевле, перенастройка ~**4×** дешевле
Масштаб эфиров	**150+** трансляций в неделю в пиковый сезон, **10 000+** за всё время, **6** стран
Облачный рендер	стабильный 1080p на дешёвых CPU-нодах, около **$1** за четырёхчасовой эфир
AI в продакшене	**~40** эфиров в неделю два года; снята статья в **~$100 на матч**
Глубина аналитики	**~295** метрик игрока, **48** командных, **180** миграций базы
Интеграции данных	**20+** внешних поставщиков спортивных данных
Регуляторный энтерпрайз	SaaS отчётности с **~240 000** компаний; **~105** таблиц с MongoDB на PostgreSQL; **7** крупнейших банков страны
Найм и знания	**70+** технических интервью, **25+** архитектурных и стоимостных оценок продуктов, внутренний формат докладов
Аналитика маркетплейса	каталог на **~100 млн** товарных карточек

Часть 1 — VSporte.ru (2019 — настоящее время)

Head of Engineering / Solution Architect. Эфирная графика, спортивные данные, медиаплатформы, computer vision.

vsporte.ru — экосистема технологичных решений в спортивной индустрии.

1.1 Инженерная платформа, построенная с нуля

Проблема. Когда я пришёл, вся инженерная сторона компании — это примерно двадцать файлов исходного кода, запушенных прямо на виртуальную машину, без контейнера. Ни CI/CD, ни оркестрации, ни наблюдаемости, ни воспроизводимого окружения. Каждый релиз был поступком.

Что сделал. Построил платформу, на которой сейчас работает вся линейка, и передал эксплуатацию дальше, вместо того чтобы стать её вечным владельцем.

- Self-hosted **GitLab** с продуктовыми группами, выделенными раннерами на каждую и общими CI-шаблонами, чтобы новому сервису не требовался новый пайплайн
- **Kubernetes** через **Terraform**: два кластера, продовый и dev, тринадцать нод-групп и выделенные пулы под каждый продукт с taint'ами, чтобы одна шумная нагрузка не съедала соседей
- **GitOps на FluxCD** для семи продуктовых тенантов, с автоматическим обновлением образов, чтобы собранный образ доезжал до окружения без ручной правки манифеста
- Платформенный слой под GitOps: ingress, управление сертификатами, **Prometheus и Grafana**, **Thanos** для долгих метрик, **Loki** для логов, алертинг в чат команды и UI для операций с кластером
- CI как продукт, а не набор скриптов: сборка образов без привилегий через **Kaniko**, semantic release, Helm-чарты в OCI-реестр и монорепозиторий, который раскладывается в child-пайплайны по сервисам
- **Sentry** на своих серверах, встроенный в цикл разработки, и **Outline** как база знаний, вместе с процессами ведения документации вокруг неё
- **AI-агенты внутри CI/CD:** по фильтру инцидентов на проде агент создаёт merge request, в котором есть и фикс, и объяснение инцидента, и этот MR уходит на ревью разработчику. Контур закрывается человеком — сознательно
- Мониторинг истечения сертификатов по всей публичной поверхности линейки с алертами в чат
- **DevOps как роль, переданная осознанно:** первые три года вёл сам, потом привлёк аутстаф-инженера, потом обучил девопсу backend-разработчика из основного состава — это сняло зависимость от внешнего подрядчика

Стек. Terraform, FluxCD, Kubernetes, Helm, GitLab CI, Kaniko, ingress-nginx, cert-manager, Prometheus, Grafana, Thanos, Loki, Sentry, Outline.

Результат. Компания прошла путь от одной машины и ручных деплоев до мультитенантной платформы под GitOps с примерно семьюдесятью сервисами в периметре — и от единой точки отказа в голове одного человека до документированной платформы, которую может вести другой инженер.

1.2 Платформа эфирной графики

Флагман. Публичный сайт: smartgfx.ru

Проблема. До платформы никакой унификации не было. Одну лигу снимало множество независимых съёмочных групп, и графику каждая делала по-своему: кто-то передавал файлы графики между командами, кто-то рисовал заново с нуля. Итог — одна лига, один исходный дизайн и заметно разная графика на каждом матче. Данные матча набирались руками во время эфира, поэтому опечатки на экране были нормой. Владение мячом считали шахматными часами. Качество того, что видел зритель, зависело от того, какая бригада вышла на матч.

Что сделал. Пришёл backend-разработчиком, забрал DevOps, затем фронтенд, затем архитектуру. Первые версии спроектировал и довёл до стабильных продаж практически один, дальше по мере роста команды перешёл к архитектуре, ТЗ и менторству.

- Унифицировал графику по лиге независимо от того, какая группа работает на матче, и заменил ручной набор данных во время эфира автоматическим сбором
- Спроектировал и провёл миграцию с **.NET-монолита на Python-микросервисы** в Kubernetes, не останавливая эфиры. Сегодня в проде восемь Python-сервисов; прежняя версия публичного API осталась на .NET по плану, а не по случайности: помечена в API как устаревшая и заморожена, правки в неё не вносятся, живёт ради клиентов, которые продолжают работать со старой

версией Переход шёл на общей базе, а не в двух изолированных мирах: и старый, и новый код ходили в одну базу. Фичи переезжали на новые рельсы по одной, а не пакетом, а раскатка шла через фича-флаги и канареечные релизы, чтобы держать SLO перед бизнесом всё время перехода

- Построил путь «действие оператора → эфир»: действие уходит в очередь с ключом на матч и на тип потребителя, дальше по **server-sent events** в эфирный оверлей, с прежним realtime-транспортом как fallback и фиче-флагом на матч, на вид спорта и глобально — чтобы переход можно было катить вперёд и назад безопасно
- **Перенёс создание графики с разработчиков на дизайнеров.** Раньше разработчик верстал каждый блок из проприетарного трансляционного формата в код: в среднем полтора дня на блок при двадцати-пятидесяти блоках в пакете, из-за чего планы на полгода рассыпались с приходом нового клиента. После перехода на **Rive** дизайнер рисует блок, разработчик только сопоставляет данные: около половины дня на блок, четыре-восемь блоков в день, и мощность масштабируется аутсорс-дизайнерами, которые в два-три раза дешевле разработчиков
- Параллельная мультиязычность: переводы сущностей на уровне базы, язык как аргумент запроса, одно окно управления отдаёт графику сразу на нескольких языках
- **20+ интеграций** с внешними поставщиками спортивных данных: импорты по расписанию, парсинг HTML там, где у поставщика нет API, алерты об ошибках импорта в чат
- **Оптическое распознавание стадионных часов**, заведённое в контур таймера, чтобы игровое время на экране совпадало с физическим табло
- Персонализация операторов: сохранённые на пользователя хоткеи, кастомные панели ведения трансляции, заготовленные последовательности вывода в эфир
- Публичный API продукта с токенной авторизацией и инкрементальной выдачей изменений по курсору: события матча, изменения состояния, живая статистика ожидаемых голов
- Конвейер ассетов с генерацией нескольких размеров изображений и опциональным удалением фона, и оффлайн-очередь логов на фронтенде, которая досылается при возврате сети

Пять решений, из которых выросло остальное.

- **Бэкенд — это проекция и ничего больше.** Наружу уходят только те поля, что объявлены в схеме графического пакета: из API поставщика на пятьсот полей фронт получает четыре. Все вычисления и вся бизнес-логика живут на фронте. Звучит задом наперёд, пока не увидишь следствие ниже
- **Превью в редакторе и картинка в эфире — один и тот же код.** Раз вычисления принадлежат фронту, то, что оператор настроил, и то, что пошло в эфир, не могут разойтись. Второй реализации, которую надо держать в синхроне, просто нет, поэтому целого класса ошибок «в редакторе выглядело правильно» не существует
- **Контракт с дизайнером сведён к одному булеву полю.** Приложение не знает, как анимируется плашка и сколько длится анимация: оно ставит флаг скрытия в графическом файле, а размонтирование происходит по сигналу от самой графики. Дизайнер остаётся свободен менять анимацию, не трогая код
- **Таймер хранит опорную точку, а не счётчик.** Он помнит момент запуска и значение в этот момент, вместо того чтобы инкрементировать. Нет накопленного дрейфа, время правится в любой момент, состояние восстанавливается после перезапуска. Три режима закрывают три вида спорта: грязное время вверх от нуля, чистое время вниз до нуля с остановкой и обратный отсчёт владения; добавленное время — отдельная величина, а не поправка к основной
- **Консолидация стека была бизнес-решением раньше, чем техническим.** Компания шла к кросс-функциональным командам, а для этого нужен один общий стек вместо двух, так что

выбор языка следовал из оргструктуры, а не из технических предпочтений. Поддерживающая экономика была настоящей, но вторичной: часть работы требовала Python-библиотек, которым не было аналога в .NET, а MVP на Python дешевле в разработке на стадии, где нагрузки, ради которой стоило бы держать более тяжёлый стек, ещё нет. Прослойка на .NET вокруг Python-приложения существовала только как переходный этап, и вся новая разработка ушла на Python. От .NET осталась одна устаревшая версия API: заморожена, а не поддерживается, и держится ровно ради тех клиентов, которые на неё ещё завязаны

Стек. Python 3.12, FastAPI, GraphQL, .NET (одна замороженная устаревшая версия API), Angular в операторском приложении и в оверлее, Rive, PostgreSQL, RabbitMQ, Redis, SSE, Kubernetes с Flux и Helm, S3.

Результат. 150+ трансляций в неделю в пиковый сезон и **10 000+ эфиров** за всё время в шести странах, 40+ графических пакетов, пять с лишним видов спорта. Единая картинка независимо от бригады, меньше ошибок в эфире, потому что данные перестали набирать вручную, и предсказуемые сроки фич, потому что производство графики ушло из кодовой базы.

1.3 Облачный рендер: эфирная графика без студии

Проблема. Классический способ наложить графику на живой поток — студийная станция с GPU на каждый параллельный эфир, в арендованном помещении. Это значит: построить и оборудовать студию, купить машины, платить аренду, посадить людей на место. Каждый дополнительный параллельный эфир — новая закупка железа.

Что сделал. Спроектировал и написал первую рабочую версию сам как эксперимент, потом передал продуктовой команде на развитие.

- Компоновщик на **C++** из четырёх взаимодействующих потоков: опрос состояния графики по HTTP, приём **SRT** и декодирование через FFmpeg, рендер графики Rive через **Skia**, композит с альфой и кодирование обратно в SRT
- Именно связка Rive и Skia делает экономику: **стабильный 1080p на дешёвых CPU-машинах**, без GPU и без студии. Там, где нужен 4K на 60 fps, используются GPU-машины в облаке, но это отдельный профиль нагрузки, а не условие работы системы
- Realtime разобран явно: ограниченные очереди кадров с дропом старых, предбуфер оверлея, выдача с фиксированным ритмом и ресинком при дрейфе, аудио транзитом с пересчётом таймстемпов
- Графика приходит JSON-контрактом — файл графики, какую state machine запускать, какие данные привязать, какие ассеты — и кешируется локально; состояние опрашивается раз в секунду
- Прод-профиль: 6 Мбит/с, 30 fps, задержка SRT в диапазоне 200–400 мс, целевой бюджет **меньше секунды** от события до кадра на выходе
- Раскладка на несколько машин: релей кормит узлы оверлея по видам спорта, те кормят приёмники; образы в реестре, compose-файл на каждый хост

Выбор был измерен, а не назначен. Прежде чем брать нативную сборку, я сравнил её с веб-рантаймом, гоняющим Skia-канвас внутри Node, на 1080p, и установил, что тридцать кадров в секунду достижимы на восьми ядрах при четырёх-шести воркерах. То же исследование разобрало, почему в нативной сборке пропадали шрифты и не работал автолэйаут — а такие вещи решают выбор движка куда чаще, чем чистая пропускная способность.

Стек. C++, rive-runtime, Skia, FFmpeg, libx264, SRT, Docker, плюс Python для офлайн-рендера и SRT-релея.

Результат. Рендер графики уехал из студии в облако, и исчезла целая статья расходов: ни помещения, ни GPU-станции на каждый эфир, ни аренды, ни оператора на месте. Рендер обходится примерно в **доллар за четырёхчасовой эфир**, а параллельные эфиры добавляются запуском контейнера, а не покупкой железа. Это же сделало реально возможным удалённое производство — поток обрабатывается в облаке, а не в здании.

Что за этим стоит. Путь сюда не был прямым: сначала скриншоты из headless-браузера, потом рендер на WebAssembly на CPU, потом нативный SRT-компоновщик. Каждый шаг дешевле и стабильнее предыдущего.

1.4 Universal — платформа вместо бесконечных разовых сборок

Это третий подход к одной и той же мысли, а не внезапное озарение. В исходном техническом задании 2021 года на первую версию платформы графики уже стояло понятие графического пакета: одиннадцать групп графических форм под разных заказчиков внутри одного продукта. В 2023-м я набросал гипотезу описывать графику, анимацию и поля данных прямо в дизайн-инструменте и интерпретировать это в продукте — то же самое разделение труда: дизайнер владеет визуалом, приложение владеет данными. В Universal это разделение наконец стало архитектурой, а не договорённостью.

Проблема. Каждый запрос клиента превращался в полуизолированный продукт со своей кодовой базой, своей инфраструктурой и своим куском внимания команды. Линейка росла — доставка замедлялась, оценки перестали держаться.

Идея. Проект равен общему ядру плюс плагины. Ядро отвечает за получение, интерпретацию, синхронизацию и распространение данных. Всё продуктовое живёт в плагинах: элементы управления, источники данных, рендереры, типы справочников, вспомогательные функции. Решение под клиента собирается конфигурацией, а не написанием нового приложения.

Два времени жизни данных. *Проект* — долгий шаблон: справочники, источники, переменные, страницы. *Матч* — короткий экземпляр исполнения с живым реактивным состоянием. Один проект порождает много матчей, и его конфигурация при этом не меняется. Роли следуют из этого разделения: технический специалист настраивает проект, оператор работает внутри матча по готовой конфигурации.

Реактивная модель. Три слоя со строгим направлением потока. **L1** — однородное живое состояние, которое заполняют producers: HTTP- и WebSocket-источники, ручной ввод, таймеры, проекции справочников, сам оверлей. Синхронизируется между клиентами по WebSocket: снапшот при подключении, дальше патчи, — поэтому оператор, судья и эфирный оверлей видят одно состояние. **L2** — производные значения, которые считает движок выражений. **L3** — привязки потребителей: поля оверлея, элементы интерфейса, табло, генерируемые документы. Слои разделены по ответственности, а не по удобству, поэтому замена одного не ломает другие.

Что я в неё заложил.

- **Язык пайплайнов, программы которого — деревья, а не строки.** Пользователь описывает трансформацию коллекции структурированным деревом, и одно и то же дерево исполняется двумя способами: превью прямо в браузере и **компиляция в SQL по JSONB** на бэкенде. Имена полей уходят bind-параметрами, поэтому сгенерированный запрос закрыт от инъекций
- **Граф зависимостей, выведенный из самих деревьев.** Зависимости между справочниками извлекаются из дерева, хранятся явно, проверяются на циклы и пересинхронизируются в топологическом порядке при изменении источника
- Движок выражений поверх живого состояния матча с детектированием циклов и привязки потребителей, которые кормят view model графики напрямую

- **Страница как настраиваемый источник данных.** Указываешь платформе страницу, называешь нужные поля обычными словами, и модель читает страницу и предлагает XPath для каждого. Человек подтверждает, и с этого момента извлечение детерминированное: модель отрабатывает один раз при настройке и не участвует в прогонах. Тот же набор селекторов дальше обходит дерево: страница соревнования отдаёт все страницы клубов, каждая страница клуба отдаёт страницы участников, а страница участника отдаёт саму запись и события, где этот участник заявлен. Обход объявляется, а не программируется, и завести новый источник — работа продуктового специалиста в интерфейсе, а не разработчика в репозитории
- Изолированное исполнение пользовательских трансформаций над данными справочников
- Копирование проекта, которое клонирует справочники, доступы и граф зависимостей вместе, чтобы новый клиент начинал с работающей конфигурации
- Дизайн инкрементального обновления: детект изменений по хешу содержимого, ключ распространения на каждом ребре зависимости, частичные прогоны пайплайна по партиции и статический анализ дерева на глобальные агрегаты, которые вынуждают полный пересчёт
- Вызов модели может стоять внутри графа конфигурации как нода пайплайна: обогащение данных и генерация нарратива без отдельного сервиса

Стек. Python, FastAPI, PostgreSQL, React, TypeScript, WebSocket, Helm, Kubernetes, Prometheus, Rive на стороне рендера.

Результат. Новые пакеты фич под клиента стали примерно **в 2 раза дешевле**, перенастройка существующих продуктов — примерно **в 4 раза**, а настройка ушла с инженеров на технических экспертов продукта, которые работают конфигурацией, а не кодом. **Линейка сейчас переезжает на платформу** — это и есть более трудная половина работы, до которой большинство платформенных проектов не доходит.

1.5 Платформа спортивных данных и аналитики

Сделана внутри партнёрства «Яндекс Плюс» и VSporte — Yandex Plus VSporte. Яндекс Плюс отвечает за технологическую инфраструктуру и нейросети, VSporte выступает официальным поставщиком статистических и фитнес-данных для РПЛ.

Проблема. Вокруг спорта не было аналитического слоя и не было витрины данных. Лиге, её клубам, тренерским штабам и скаутам некуда было пойти за полным матчевым датасетом, и не было способа его нарезать, сравнить или выгрузить. Любой вопрос — сколько пробежал этот игрок, как менялось владение, сделай отчёт по туру — означал ручную работу для кого-то.

Что сделал. Head of Engineering / Solution Architect: архитектура всей системы, hands-on на Python и React, отвечал за доставку, интеграции и инфраструктуру вокруг.

- **Девять сервисов на FastAPI** в проде: клиентский API для клубов, администрирование, публичный API, обработка, приём партнёрских данных, отчётность по расписанию, конфигурация; плюс клубный портал и админка на React и Next.js
- **Событийная обработка на Kafka:** топики на расчёт фитнес-метрик, статистику, генерацию видео и приём партнёрского трекинга. Трекинг приходит чанками по 1500 кадров при 25 кадрах в секунду, а сервис обработки автоскейлится с трёх до семи воркеров под нагрузкой
- **Схема на ~295 метрик игрока и 48 командных** в **180 миграциях**: дистанция с мячом, ожидаемые голы, ожидаемые ассисты, спринты, владение и многое другое. Метрики считаются из разметки событий; скорости — оконными функциями SQL со сглаживанием; ожидаемые голы — детерминированная модель на сетке поля с весами по части тела, сохранённая на каждый удар отдельно, а не как непрозрачное число

- **Отчётность как продукт, а не кнопка «выгрузить».** PDF собирается из живых страниц портала и хранится с версией на язык; очередь задач рассылает три семейства отчётов — индивидуальный, стандартный и по туру — в вариантах TTD, фитнес и таблица; стандартный отчёт уходит через два дня после матча; доставка почтой. Графики и табличный экспорт генерируются на клиенте
- **Фитнес-трекинг** приходит от партнёра через колбэки с подтверждением чанков, параллельно с нашим собственным контуром получения позиций игроков из видео (см. 1.10)
- Разметка принимается из двух независимых источников — внешнего сервиса разметки и нашего собственного инструмента — с таблицей соответствия идентификаторов между системами
- Аналитические дашборды, встроенные в клубный портал

Стек. Python, FastAPI, Kafka, Celery, PostgreSQL, Redis, Kubernetes с Flux, React, Next.js, библиотеки графиков и табличного экспорта, S3.

Результат. У лиги, клубов и их аналитиков появилось единое место, где живёт полный матчевый датасет, автоматическая отчётность вместо ручных запросов и витрина данных, из которой берёт данные вся остальная линейка.

1.6 AI в эфире: автофакты, вероятности и живая модель счёта

Отдельная продуктовая линия поверх датасета из предыдущей карточки.

Проблема. Подготовка интересных фактов к эфиру была ручным конвейером: человек перед каждым матчем копал историю личных встреч и рекорды игроков, примерно **$100 на матч**. При этом эфиру нужны были не только факты, но и живые вероятности, которые двигаются вместе с игрой.

Три механики, которые сознательно разделены.

1. **Автофакты** — нарративные утверждения из истории личных встреч и из личных рекордов игроков и клубов. Генерируются моделью, которая обращается к историческому датасету через извлечение, с метаданными для эфирной плашки: логотип, категория, приоритет.
2. **Инсайты** — вероятности событий, посчитанные на истории: тоталы, ожидаемые голы, карточки, удаления, офсайды, исходы.
3. **Предикт счёта** — **собственная модель**, не букмекерский фид: вероятности исходов, вероятности точных счётов и расхождение с рынком ставок, обновляемые **в live** по ходу матча.

Как это попадало в эфир. До начала матча оператор выбирает, какие факты и инсайты использовать; во время матча иногда подтягивает новые. Всё выводится через платформу графики, поверх обычного эфирного оверлея. Что именно окажется на экране, всегда решает человек.

Что сделал. Head of Engineering / Solution Architect: спроектировал предикт-модель и весь путь от датасета до эфира, часть написал на Python и React, менторил команду, делал code review.

Стек. Python, FastAPI, PostgreSQL, Kafka, Kubernetes. API моделей OpenAI и Anthropic под сценарии генерации, извлечение по собственному историческому датасету, своя предиктивная модель исходов и детерминированные вероятностные модели — живая функция win chance из текущего счёта, владения, сезонных ожидаемых голов, удалений и таймера — для тех значений, которые не имеют права никого удивить в прямом эфире.

Результат. Два года работы на **~40 эфирах в неделю**. Статья расходов в **$100 на матч** исчезла. Живые факты и вероятности стали инструментом вовлечения зрителя, а не хорошей идеей. Позже механика была унаследована и переработана внутри платформы Universal — со слабой связанностью и конфигурацией вместо жёсткой связки.

Чем это отличается от демо. Это не демо чат-бота. Это генеративная и предиктивная работа внутри системы под давлением прямого эфира, с человеком в контуре, со снятой статьёй расходов и с измеримой регулярностью.

1.7 Инструмент разметки событий матча

Проблема. Чтобы посчитать любую спортивную аналитику, кто-то сначала должен превратить видео матча в структурированные данные: кто, что, где и когда сделал. Внешние поставщики такую разметку продают, но глубина у них фиксированная, а цена и сроки — их. Нам нужны были данные той глубины, которую требовали наши собственные продукты.

Что делает. Оператор смотрит матч и в реальном времени фиксирует действия каждого участника, а после эфира спорные моменты можно уточнить. Не «был удар», а действие конкретного игрока против конкретного соперника, **с координатами точки действия и точки назначения**, с указанием части тела и периода. Для футбола в интерфейсе 29 типов действий, 5 частей тела, 7 типов игры и 28 позиций игрока.

Что сделал. Архитектура на основе платформы графики, ТЗ и планы разработки, контроль соответствия, code review, менторство.

- Операторский интерфейс на Angular, унаследованный от клиента платформы графики, два бэкенда на FastAPI, PostgreSQL на 33 таблицы, четыре прод-релиза под GitOps
- Realtime-раздача по server-sent events с семантикой добавления, правки и удаления и выделенной очередью на каждое соединение, чтобы один зависший потребитель не ронял остальных
- Конфигурация вида спорта хранится в базе структурированно, а не только статикой, поэтому новый вид спорта не требует деплоя
- Импортёр в аналитическую платформу с сопоставлением идентификаторов, что делает этот инструмент одним из источников центрального датасета

Результат. Глубина разметки стала продуктовым решением, а не ограничением вендора, а аналитическая платформа получила независимый источник данных, который контролирует сама. Поддержано четыре вида спорта.

1.8 Медиаплатформа: прямой эфир и видео по запросу

Проблема. Партнёрам и зрителям нужно смотреть спортивное видео — и прямой эфир, и записи — с брендированным плеером и возможностью сразу перейти к важным моментам, а правообладателю нужно отдать один поток и получить несколько адресатов раздачи.

Что сделал. Архитектура, первая рабочая версия, ТЗ и планы, code review, менторство.

- Приём сигнала по **RTMP и SRT** с multicast-раздачей, живая выдача через **HLS** и записи в MP4
- Веб-платформа просмотра: клиентское приложение и административная консоль
- **Свой встраиваемый пакет плеера** — либо React-пакет, либо embed через iframe, с оформлением под спонсора: логотипы, цвета, стилистика
- Расширения плеера: переход к ключевым моментам вроде голов, режим закладок, временные метки и seek по ним
- Транскодирование без перекодирования там, где достаточно ремукса, рендер клипов через облачные функции, управление баннерами и объектное хранилище под всем этим

Стек. Python, FastAPI, nginx-RTMP, HLS, FFmpeg, React в пакете плеера, PostgreSQL, RabbitMQ, Redis, S3, облачные функции, Kubernetes.

Результат. 10M+ просмотров видео суммарно по эфирам и записям, и пакет плеера, который переиспользуется в партнёрских встройках, а не собирается заново под каждого партнёра.

1.9 Оценка стоимости спонсорских интеграций по видео эфира

Публичный продукт: eyedy.io

Проблема. У спонсора нет честного ответа, что купили его деньги внутри трансляции. Носители разные — LED-борта по периметру, статичные баннеры, форма, напольная графика — и стоимость контакта зависит от типа носителя, площади, позиции в кадре, чёткости изображения, времени на экране и аудитории за всем этим. До системы это оценивалось на глаз и решалось в переговорах.

Что сделал. Head of Engineering / Solution Architect: архитектура продукта, ТЗ и планы разработки, менторство, code review.

Продукт. Покадровый разбор трансляции, в котором computer vision фиксирует появление логотипа или носителя в кадре; математическая модель media value на **15+ параметрах** — аудитория и время контакта, длительность интеграции, охват, позиция и площадь в кадре, тип носителя, чёткость изображения, стоимость охвата; и отчётность по телевидению, соцсетям и веб-ресурсам с рекомендациями и разрезами по матчам, месяцам и командам.

Интересная инженерия. Точность всей системы держится на двух решениях.

- **Детекция на правильном масштабе.** Логотипы на широком плане мелкие, поэтому детекция идёт не по кадру целиком, а по нарезке кадра на перекрывающиеся тайлы. В пайплайне YOLOv8 со срезанным выводом
- **Ракурс камеры без ручной калибровки.** Система выделяет поле маской по цвету, отдельная модель предсказывает матрицу гомографии, кадр разворачивается в схему поля, и по этой схеме считается, **какая доля каждого из двенадцати сегментов периметра реально попала в кадр**. Именно это превращает «баннер было видно» в число

Стек. Python, computer vision, YOLOv8 со срезанным выводом, PyTorch, OpenCV, проективная геометрия, system design. Мой вклад здесь архитектурный, а не обучение моделей.

Результат. Стоимость спонсорской интеграции перешла из переговоров и оценки на глаз в измеримое покадровое число и в отчёты, которые могут читать и правообладатель, и спонсор.

1.10 Трекинг игроков по видео трансляции

Проблема. Данные о перемещениях игроков либо покупаются у поставщика, либо собираются носимыми трекерами на игроках. И то и другое стоит денег, и ни то ни другое недоступно на каждом матче. Мы хотели получать координаты из того, что уже есть, — из картинки трансляции.

Что делает. Берёт потоки с камер, детектирует и трекает игроков, переводит пиксельные координаты в **метры на реальном поле 105 на 68** через гомографию и **сшивает идентификаторы игроков между камерами** на линии центра поля, чтобы один игрок не превратился в двух при переходе между кадрами. На выходе траектории, из которых считаются дистанции и скорости. Работает и **в live, и как постобработка** записи.

Стек. Python, детекция YOLO на высоком входном разрешении с батчингом, многообъектный трекинг, модель повторной идентификации под футбол, матрицы гомографии на каждый сетап, сшивка идентификаторов между камерами с порогом по расстоянию.

Результат. Данные о перемещениях стали выводимыми из самой трансляции — а это разница между «трекаем каждый матч» и «трекаем те матчи, которые кто-то оплатил приборами».

1.11 Графика дополненной реальности поверх живого видео

Проблема. Нарисовать 3D-объект поверх живой картинки просто. Заставить его вести себя как часть сцены — нет: он не должен перекрывать игроков, которые находятся перед ним, должен стоять в правильном месте поля и не должен плавать при движении камеры.

Что делает система. Принимает живой поток по **NDI**, то есть стоит внутри реального продакшн-контура, а не рядом с ним. Сегментирует игроков и трекает их, стабилизирует створ ворот, выводит геометрию поля по ключевым точкам. Дальше кормит игровой движок: маски игроков кодируются в компактный поток и уходят по сокету, треки и геометрия ворот передаются рядом. Маски и есть ответ на вопрос, что рисуется перед игроком, а что за ним.

Моя роль. Архитектурные консультации для команды — этот продукт вырос без меня, и я советовал по его устройству, а не строил его.

1.12 Программно-аппаратный комплекс наложения рекламы на несколько потоков

Сделан для Кинопоиска.

Проблема. Один входящий поток должен порождать несколько исходящих, у каждого своя реклама, и таких потоков идёт несколько параллельно. До системы на каждый входящий поток нужна была своя машина и свой оператор, который смотрит эфир и решает, что выводить. Масштабирование означало найм.

Что сделал. Первая рабочая версия, архитектура, ТЗ и планы разработки, code review, менторство.

Решение. Программно-аппаратный комплекс: восемь машин под трансляции плюс одна операторская станция. Каждый узел отдаёт чистый выход и выход с графикой; операторская станция показывает их все на одном экране. Один человек настраивает ретрансляцию, число выходов, рекламную вёрстку и внешние материалы и ведёт эфир в live, а система ему подсказывает.

Автоматика. Приложение опрашивает внешний сигнальный API по состоянию матча — периоды, овертайм, окончание — и **планирует показ рекламы с настроенной задержкой по условию**, то есть перерыв ловится без человека, который его высматривает.

Стек. Electron и React в операторском приложении, часть бэкенда на .NET, управление vMix по HTTP, NDI для раздачи потоков.

Результат. Один оператор заменил восьмерых. В проде комплекс обычно держал **пять одновременных матчей при потолке восемь**; при масштабировании на **три комплекса, каждый под своё направление, суммарно доходило до 12 параллельных эфиров**.

1.13 Облачное наложение рекламы как продукт самообслуживания

Проблема. Комплекс из предыдущей карточки показал, что подстановка рекламы в поток — это не разовая интеграция под одного заказчика, а продуктовая потребность. Рекламодатель хочет сам заводить кампанию, видеть, где и когда она вышла, и получать отчёт. Правообладатель хочет отдать один поток и получить несколько адресных версий. Оператору нужно рабочее место, где всё это видно.

Что построил. Облачное наложение рекламы с **порталом самообслуживания для рекламодателя** и операторским приложением. Портал: кампании, расписание, креативы, дашборд, отчётность. Операторское приложение: контроль потоков и вывод в эфир. Между ними — спецификация интеграции с realtime-каналом.

За этим: полный набор требований и спецификаций — бизнес-требования, техническое задание, постраничная раскладка интерфейсов, контракт интеграции web с desktop и дизайн-система, —

работающее операторское приложение на Electron, React и TypeScript, десятистраничный административный интерфейс и поэтапный план с первым релизом за 8–10 недель.

Как это устроено внутри. SRT-эндпойнт создаётся динамически на каждую сессию, а не выделяется заранее. Наружу отдаются два потока раздельно: **clean** — релей с минимальной задержкой под превью, и **dirty** — композит с буфером в одну-две секунды, потому что наложение требует буфера, а превью не должно за него платить. Команды маршрутизируются в поды рендер-сервиса через **Redis Pub/Sub**, а для биллинга считаются фактические показы, а не расписание.

Две оценки, и это сознательно. Настольное демо на Electron и FFmpeg посчитано отдельно от настоящей облачной платформы, с честно помеченными мокапами интерфейса и отдельной строкой «инструкция для отдела продаж». Продать демо как продукт — стандартный способ убить такой проект.

Стек. React и Next.js, FastAPI, PostgreSQL, Redis, S3, CDN, FFmpeg, WebSocket, Electron.

Что это показывает. Это самый наглядный отдельный пример presale-архитектуры, которая стоит за строкой «25+ архитектурных и стоимостных оценок»: требования, раскладка интерфейсов, контракт интеграции, поэтапная оценка и прототип, который можно потыкать.

1.14 Быстрая нарезка хайлайтов в live

Проблема. Не было нормальных инструментов, чтобы резать видео **быстро и точно, пока эфир ещё идёт**. Требовалась точность порядка **половины секунды** на границах фрагмента, и требовалась скорость, потому что момент нужен в соцсетях и в эфире сейчас, а не после матча. Обычные инструменты либо режут по ключевым кадрам, то есть неточно, либо перекодируют весь файл, то есть медленно, либо требуют, чтобы запись сначала закончилась.

Что сделал. Архитектура, ТЗ и планы разработки, code review, менторство.

Продукт. Веб-система нарезки видео из живого потока: выбор фрагмента с субсекундной точностью по идущему эфиру, наложение текста на клип и автоматические подсказки периодов для нарезки.

Инженерия под этим. Параллельное скачивание сегментов сразу в кодировщик, точная резка фильтрами вместо границ по ключевым кадрам, быстрый путь с копированием потока там, где перекодирование не нужно, и короткие сегменты на приёме — именно они делают точные границы возможными.

Стек. Python, FastAPI, облачные функции, React, FFmpeg, HLS, Kubernetes.

1.15 Речь в действие

Проблема. Комментатор и оператор произносят вслух то, что только что произошло, за секунды до того, как это доедет до данных и до графики. Параллельно живому эфиру нужны субтитры, а записям — очистка.

Продукт. Распознавание речи из эфирного звука, превращённое в действия и текст: живые субтитры, поиск ключевых слов по записи матча для быстрой навигации и нарезки, автоматическое цензурирование субтитров.

Второй путь из того же распознавания, и он интереснее. Текст комментария уходит в языковую модель, которая возвращает структурированные события матча в JSON. Эти события кормили графический оверлей и журнал событий матча, а до оператора, ведущего эфирные титры, доходили как подсказки: какой маркер добавить следующим, какую плашку вывести. Это работало на реальных эфирах, что и отличает историю от демонстрации. Распознавание отстаёт от живого звука примерно на две секунды, то есть укладывается в окно, в котором оператор и так работает, и

подсказка приходит, пока она ещё нужна. Уходит шаг, где человек должен услышать и набрать руками, пока матч продолжается.

Честное ограничение, и оно определило конструкцию: комментатор не всегда сразу называет автора гола, а судья отменяет решения. Поэтому вывод модели — это подсказка человеку, а не прямая запись в графику.

Стек. Python, FastAPI, распознавание речи семейства Whisper, раздача по WebSocket с коротким окном распознавания, чтобы текст появлялся по ходу речи, а не в конце фразы, захват звука, FFmpeg.

1.16 Внутренние системы и консультативная работа

- **BMS, внутренняя система учёта** — сотрудники и внешние сотрудники, заказчики, оборудование, заказы и проекты, мероприятия и назначения специалистов и техники на них. Вход по одноразовому коду в SMS. Четыре сервиса на FastAPI и React-приложение под GitOps. Моя роль: архитектура на первых этапах и дальше консультирование и менторство команды
- **Генерация контента для соцсетей** — система подготовки фото и видео для спортивных социальных сетей. Архитектура, ТЗ, code review, менторство
- **Судейская система** для водных видов спорта — архитектурное консультирование
- **Проверка офсайда для системы видеопомощи судьям** — по видео проверить офсайд и подсветить часть тела, перешедшую за линию. Раннее поколение было построено на проективной геометрии в веб-стеке; моя роль — первая версия, архитектура и ТЗ

1.17 Руководство, найм и обмен знаниями

- **Четыре команды разработки собраны с нуля**, в каждой от трёх до шести разработчиков плюс QA. Всего в зоне ответственности пять команд: три в прямом ведении, две консультативно. 16 инженеров плюс менеджер и QA на каждую команду
- **70+ технических интервью** по .NET, Python, Angular, React и DevOps, включая структурированную воронку тестовых заданий, которую я вёл волнами. Интервью не импровизируется: есть written-рубрика из восьми критериев по .NET — от жизненного цикла объекта в ASP и работы с памятью до ORM, многопоточности и обработки ошибок, — каждый оценивается от одного до десяти с зафиксированным порогом, плюс четыре текстовых поля, включая факапы кандидата в тестовом задании. Отдельный банк вопросов по React несёт по каждой теме теорию, уточняющие вопросы и практическую задачу с эталонным решением. Часть оценки — способность кандидата починить собственную ошибку по наводящему вопросу, и это говорит больше, чем сам факт ошибки
- **25+ технических оценок потенциальных продуктов**: архитектура, стоимость, сроки и риски; часть из них превратилась в продукты
- **Поставил и вёл внутренний формат технических докладов и митапов** — не просто выступал, а завёл практику, в которой инженеры учат друг друга. Программу написал сам: цикл раз в две недели с чередованием подготовленных докладов и открытого микрофона, регламент шестьдесят минут с поимённой структурой слотов и одно правило, которое важнее прочих, — доклады записываем, открытый микрофон не записываем, потому что он работает только как safe space. В документе программы прямо назван и режим отказа: если идеи снизу не превращаются в решения, практика приносит вред. Стоимость объявлена заранее — час на разработчика раз в две недели. Один из моих докладов был полноценной сессией про то, как заменить бэкенд самим PostgreSQL с REST-слоем поверх него и построчной безопасностью, с запускаемым демо

- **Отправлял команду разработки титровать реальный матч.** Прямой эфир вскрывает отказы, которые не воспроизводятся на тестовом стенде, и инженер, поработавший с продуктом под цейтнотом, спорит о нём потом иначе
- **Написал runbook передачи дел уходящего инфраструктурного инженера**: сначала полные дампы состояния — все namespace, секреты, ConfigMap, тома, все базы, вся кодовая база, состояние Terraform, — затем ротация доступов по девяти системам двумя волнами. В runbook отдельно помечено единственное место, которое нельзя восстановить из дампов, объектное хранилище, и ротация его ключей вынесена в критические пункты. Когда я передавал собственную работу, та же дисциплина дала распределение моих задач и зон знания по четырём людям, у каждого назначения — обоснование
- Практики ведения документации, культура code review и материалы для онбординга новых инженеров
- Presale-архитектура: выступал голосом CTO или Head of Engineering / Solution Architect на звонках с клиентами и по интеграциям

1.18 Эксплуатация: дежурства и стоимость

Дежурства. У платформы был график дежурств. Это легко не написать в архитектурном документе, и это ровно та часть, которая решает, переживёт ли архитектура столкновение с сезоном.

Облачная стоимость как своя цифра. Инфраструктурный счёт я разношу по продуктам помесячно, а не смотрю на него одной суммой: только так видно, какой продукт сколько стоит на самом деле. Рядом живёт бэклог задач на оптимизацию затрат, где у каждой позиции стоит и ожидаемый выигрыш в месяц, и себестоимость самой работы, потому что часть оптимизаций обходится дороже того, что они экономят. Приоритет ставится по этой паре, а не по размеру экономии.

Часть 2 — SaaS государственной отчётности (2016–2019)

Контур — Full Stack Developer. Контур.Бухгалтерия, российский продукт для учёта и сдачи государственной отчётности, которым пользуются около **240 000 компаний**; в Экстерн он влился уже после моего ухода. Пришёл инженером на четвёртом курсе, вырос до senior full stack.

Проблема. Регуляторный продукт, в котором правила меняются под тобой. Форматы документов, требования к отчётности и государственные реестры задаются законодательством, и когда закон меняется, прод должен продолжать работать в тот же день.

За что отвечал.

- **Электронный документооборот** — отправка и получение счетов-фактур, накладных, актов и универсальных передаточных документов в нескольких обязательных форматах, подписанных электронной подписью, в полном соответствии с законом, включая поддержку соответствия при изменениях законодательства
- **Сдача отчётности** — интерфейс и веб-компоненты подачи в налоговую и смежные ведомства, встраиваемые в другие продукты линейки компании
- **Банковские интеграции** — выписки и платежи с **семью крупнейшими банками страны** по объёму активов
- **Миграция базы** — MongoDB на PostgreSQL, **~105 таблиц**, на живом регуляторном продукте
- **Адресная подсистема** — перевод с одного государственного формата адресов на сменивший его, когда государство поменяло формат

- **Распознавание формализованных документов** — PDF или изображение в Excel, Word или JSON. Лейауты размечались руками, разбор полей был на правилах и шаблонах; модели там не было, и работало это именно потому, что документы формализованные
- **Тестирование полного цикла** — эмуляция реального поведения пользователя в браузере, проверки того, что интерфейс действительно показал, работа с тестовыми базами и подмена внешних интеграций
- **Производительность** — оптимизация запросов и поиск утечек памяти в продукте, который вырос за пределы своих исходных допущений

Стек. C#, ASP.NET и ASP.NET Core, JavaScript, TypeScript, Angular, MongoDB, PostgreSQL.

Что это доказывает. Это сильнейшее доказательство в моём опыте для регуляторной, интеграционно-насыщенной работы на .NET: соответствие, которое не имеет права поехать, интеграции с институтами, которые под тебя не подстроятся, и миграции без простоя.

Часть 3 — govtech-венчура (2019–2020)

Global Intellect Systems — со-основатель / CTO. Стартап оптимизации работы служб благоустройства Екатеринбурга, который я вёл параллельно с основной работой.

- Архитектура платформы и руководство разработкой
- Программно-аппаратный комплекс: GPS-мониторинг техники и интеграция видеонаблюдения
- Техническая документация и требования для государственного заказчика
- Прямая работа с администрацией города как техническая сторона

Заморожен через семь месяцев. Держу его в документе, потому что работа с муниципальным заказчиком над системой из железа и софта — это другая дисциплина, чем выпуск SaaS, и потому что понимать, когда венчуру надо остановить, тоже часть работы.

Часть 4 — независимая инженерная работа

Аналитика поверх каталога Wildberries *(примерно с 2020, параллельно работе над вещанием, ~7 месяцев активной разработки)*
Платформа, отвечающая на вопрос «что изменилось» по каталогу Wildberries примерно в **100 миллионов товарных карточек**: цены, наличие, позиции, характеристики. Полный снапшот каждый день на таком объёме не живёт — хранение растёт линейно, а запросы по нему умирают. Каждый ответ парсился в набор сконфигурированных полей и складывался в **JSONB в PostgreSQL**, поэтому различие означало ровно одно и означало это на уровне полей, а не страницы. Страница, вернувшаяся заблокированной, не давала полей вообще, то есть была отказом сбора, а не изменением. Сырые ответы лежали в объектном хранилище, так что любой перепарс, пересчёт или откат шёл от исходных байтов, а не от того, что парсер понимал на тот момент. Поверх этого append-only лог изменений с партиционированием по дате и материализованные агрегаты, чтобы отвечать на вопросы **по группам, по фильтрам и по произвольным периодам дат**, а не только по отдельной карточке. Рядом с инкрементальным путём есть резервный, способный обойти и обновить **весь** каталог целиком внутри часа — на случай, когда штатный медленный поллинг деградирует или источник начинает его отбивать. Сбор шёл через пул соединений поверх **Tor**, а не через платный пул прокси, и именно это держало расходы на трафик в разумных пределах при банах адресов. Python и PostgreSQL.

Computer vision и машинное обучение
Модели с обучением с учителем для распознавания объектов и анализа поведения в потоковом

видео, сделанные для лабораторных экспериментов: мышь в лабиринте с посторонними предметами, мышь в ведре с водой, разделённом на светлую и тёмную части, рыба в аквариуме.

Оптимизация маршрутов
Построение кратчайшего пути для вертолёта, обследующего состояние водных магистралей: задача коммивояжёра на реальных картах с ограничением по времени полёта до следующей дозаправки, десктопный интерфейс настройки точек облёта и отрендеренная карта маршрута на каждый день.

Embedded
Автоматизированный инкубатор для куриных яиц на Arduino с регулированием температуры и влажности на каждом этапе развития. До сих пор самое странное задание, которое я брал.

Геймдев
Двенадцать с лишним модов для Kerbal Space Program, три из них в топ-10.

Преподавание *(2015–2016)*
Преподавал в летней школе программирования для старшеклассников. Частное репетиторство по C++, Java и .NET.

С чего это началось *(2011–2012)*
Система складского учёта для розничного магазина, Delphi и SQLite. Один оплаченный проект, сделанный ещё в школе. Профессиональный стаж я считаю не отсюда, но оплачиваемая работа начинается здесь.

Часть 5 — личные продукты

Продукт	Что это	Стек	Статус
Push100	тренер по отжиманиям с автоматическим счётчиком повторений	React Native	в App Store
Swippo	очистка фотогалереи свайпами по подписочной модели	React Native с нативными модулями iOS и Android	в App Store
Entrimo	визовый сервис: Telegram-бот и Telegram Mini App, лендинг entrimo.com	Telegram Bot API, Mini App	на старте

Сделаны, чтобы честно понимать мобильную разработку: доведение до магазина, включая нативные модули на обеих платформах, учит тому, чему чтение об этом не учит.

Публикации

- How our broadcast graphics left vMix, one bottleneck at a time — dev.to, сентябрь 2026. Три шага прочь из настольного графического пакета и узкое место, которое создавал каждый следующий. На английском языке.

Образование

Уральский федеральный университет — бакалавр, фундаментальная информатика и информационные технологии, 2012–2017.
Магистратура по компьютерным наукам (2017–2019) и, в экономическом институте, по искусственному интеллекту
в управлении капиталом (2021–2023). Ни одна не завершена; вторая шла параллельно с полной занятостью
на архитектуре.
Регулярно посещаю HighLoad и DotNext, лично или в трансляции.

Открыт к работе по solution architecture, техническому лидерству, бэкенду и платформам, архитектурным ревью и консультированию. Гражданство РФ. Удалённый штат или договор с действующим ИП. Пояс МСК+5: московская первая половина дня приходится на мою вторую, устойчивое пересечение четыре-пять часов каждый день.