Каратаев Дмитрий Александрович

Backend Software Developer (C#) · Москва, м. Коломенская

Контакты: +7 (923) 047-13-55 (предпочтительно Telegram) · dkarataev1990@gmail.com · Telegram @Krawler · GitHub · Сайт

Гражданство: Россия · **Опыт в разработке:** 8+ лет

Формат: удалённая работа · полная / частичная / проектная занятость · **Локации (удалённо):** Канада, Минск, Норвегия, США · командировки не рассматриваю

Возможные форматы трудоустройства: трудовой договор, самозанятость, ГПХ

Кратко

Backend-разработчик на **C# / .NET** (8+ лет): микросервисы, REST API, SOA, GraphQL, инженерные и документооборотные системы. Опыт полного цикла SDLC, рефакторинга и перевода монолитов к сервисной архитектуре, внедрения DI, unit-тестов, Agile-практик (DoR/DoD/AC), документации и баз знаний.

Параллельно с основной работой строю **инфраструктуру для AI-агентов**: серия открытых **MCP-серверов** (Model Context Protocol), дающих ИИ-ассистентам доступ к компилятору, отладчику, системе сборки и мультимодальным вводам. Автор методологии **agent-first-learn** — практического руководства по продуктивной работе с AI.

Помимо кода: настройка процессов, работа с требованиями, техническая экспертиза, нетривиальная отладка (дампы памяти, анализ потоков, Reflection).

AI-assisted engineering: 1.5+ года практического применения (Cursor IDE, MCP-серверы собственной разработки).

Опыт работы

ООО «НОВА ЭНЕРДЖИС» (Москва) — эксперт по технической экспертизе и поддержке

Сентябрь 2023 — настоящее время · удалённо

Энергетика · системы управления инженерными данными; по типу задач — **сопоставимо с опытом контура Harvester и корпоративного веб-портала EDW**

(сбор, обработка и публикация данных, контур цифрового двойника EPC). **Тимлид** задаёт приоритеты и общее направление команды; ниже — **личный вклад** и два кейса (**Harvester**, **веб-портал EDW**).

Задачи

1. Разработка и сопровождение сервисов по **сбору, обработке и публикации** инженерных данных.
2. Разработка и сопровождение решений по **публикации инженерных данных** в контексте цифрового двойника EPC: документы, теги, атрибуты, связи и смежные сущности.
3. Анализ архитектуры, предложения по улучшению; развитие функционала и сопровождение системы в production.
4. Внедрение **Dependency Injection**, развитие **unit-тестирования**; ведение документации (модули, онбординг, база знаний техкоманды).
5. Внедрение практик **Agile**: DoR, DoD, acceptance criteria; участие в выстраивании предсказуемого процесса поставки.

Мой вклад — система Harvester (кейс, STAR)

Ситуация: унаследованная система (в интерфейсе и на **IIS** продукт именуется **Harvester**): модули писались в разных стилях и подходах, развитие эволюционное; требования часто противоречили друг другу и менялись по ходу работ. Нужна была устойчивость и производительность на **больших объёмах данных** (порядка **~1 млн** объектов, связей и документов с атрибутикой). При переключении между задачами и модулями росла когнитивная нагрузка; документация и комментарии к коду отсутствовали или устарели.

Цель: ускорить ввод новых и доработку существующих фич, снизить когнитивную нагрузку при смене контекста, повысить прозрачность работы с системой для пользователей и **стабильность** контура подготовки → валидации → публикации в корпоративный веб-портал.

Действия:

- Внедрил **.editorconfig** (за основу взят конфиг **dotnet/runtime**): единый стандарт оформления, снижение «визуального шума» и лишних предупреждений при переходе от модуля к модулю; довёл существующий код в соответствие с правилами.
- Провёл встречи с командой и стейкхолдерами, чтобы обосновать необходимость изменений.
- За **2 дня** предложил и реализовал внедрение **Dependency Injection** на базе **Microsoft.Extensions.DependencyInjection** (совместно с тимлидом).
- Создал и вёл **базу знаний** и **пользовательскую документацию**: логика работы системы, шаблоны оформления обращений в поддержку и др.
- Реорганизовал общую библиотеку (репозитории, объекты предметной области) в **feature-based** структуру.
- **Архитектура:** общая **библиотека ядра** на **.NET 9** (**SQL Server**, **Dapper** / **RepoDb** (в т.ч. bulk), **CsvHelper**, **Serilog**, **feature flags**) и три сервисных

контура — **сервис подготовки файлов и данных**, **сервис валидации** (в т.ч. правила именования тегов и **RDL**), **фоновый сервис публикации** (таймеры проверки согласованности и выгрузки, staging, выгрузка **CSV** в каталог веб-портала по конфигурации). Поверх — **веб-слой**: **ASP.NET Core** + **Angular** (SPA для статусов зарезервированных тегов, процессов загрузки и смежных сценариев). Параллельно развивался доступ к данным через **Linq2db** (контекст в DI, выравнивание имён сущностей со схемой БД) — **рядом** с существующим **Dapper** на отдельных участках. Для совместимости с инженерной моделью **AVEVA** используется слой **воссозданных** API-сборок под контракты модели — без опоры на проприетарные бинарники вендора там, где обходились своим кодом.

- **Связка с порталом:** потоки публикации учитывают наличие тега «в портале» (валидация и комментарии/ссылки не создают лишнего шума, если карточки в портале нет); выгрузки и имена CSV приводились к **единому шаблону** имён файлов; исправлялись **конкурентные** обращения к общему пайплайну публикации (**lock**-и на стороне воркера, сценарии «готово к тестам») и ошибки конфигурации таймеров (интервал в **секундах**, а не минутах), **кодировка/заголовки CSV** — по следам инцидентов и код-ревью; отдельные крупные доработки по **экспорту** и стабилизации публикации велись в выделенных ветках.
- **Эксплуатация:** PowerShell-скрипты обновления **RDL** под развёртывания **IIS** (отдельные стенды/сайты для веба и сервисов) — в одном контуре с задачами сопровождения.

Результат:

- Единый стиль и предсказуемая структура снизили ментальную нагрузку при подхвате чужой задачи или переходе в другой модуль.
- Внедрение новых фич ускорилось: проще наращивать слои; при **feature-based** доменной модели расширения оказываются в ожидаемых местах.
- Открылась дорога к **unit-тестам** (изначально отсутствовали): проще подменять реализации через интерфейсы; в решении развиты отдельные **тестовые проекты** по ключевым слоям (ядро, публикация, валидация, веб, приёмочные, подготовка).
- Контур **подготовка → валидация → публикация → портал** сопоставим по смыслу с **импортозамещённым веб-порталом** (второй кейс ниже): те же классы задач (теги, документы, атрибуты), другой стек и владение кодом.

Мой вклад — веб-портал EDW (кейс, STAR)

Ситуация: после ухода ряда западных вендоров с рынка прекратилась поддержка продуктов в линейке **AVEVA** (в т.ч. **NET Portal**, **AVEVA ISM**). Для непрерывности работы с инженерными данными потребовалась **замена** и снижение зависимости от недоступной поддержки и обновлений.

Цель: по согласованному плану — **MVP корпоративного веб-портала**: просмотр документов, тегов и **связей** между сущностями, поиск и работа с файлами (в т.ч. превью), чтобы пользователи могли продолжать работу **без привязки к**

исходному вендору.

Действия:

- Участие в **реализации MVP** на стеке **Blazor WebAssembly**, **GraphQL**, **EF Core**, **.NET 8+** (в проде — актуальный **.NET 9** на сервере и клиенте).
- **Архитектура в двух словах:** тонкий **Blazor WASM**-клиент (готовая **UI**-библиотека компонентов, доступ к API через сгенерированный **GraphQL**-клиент) и **ASP.NET Core**-сервер с единой **GraphQL**-точкой: проекции, фильтрация, сортировка, **cursor pagination** — без разрастания набора REST-ресурсов под каждый экран. За сервером — **SQL Server** (база портала), **пул фабрик DbContext** и режим **NoTracking** под нагрузку чтения; отдельно — контроллеры для файлов, экспорта и длительных операций.
- **Разделение модулей:** онлайн-стек (**API + Blazor**) **не ссылается** на общую библиотеку домена — только на контракт по сети. Общая **EF**-модель (контекст БД, сущности) вынесена в **отдельную библиотеку** и используется **фоновыми процессами** (синхронизация, импорт, мониторинг файлов). У сервера — **своя копия** сущностей: осознанное разделение «API/UI» и «интеграции», с понятным **риском рассинхрона** схемы при изменениях (поддерживается дисциплиной ревью и миграций).
- **Интеграция с корпоративным хранилищем инженерных данных:** три независимых конвейера — **инкрементальная** подтяжка метаданных документов, **полная** синхронизация цепочки ревизий и версий, затем **привязка файлов** с диска (staging / fallback), чтобы у версии появился **основной файл** для превью в портале. Участвовал в проработке порядка данных (документ → ревизия → версия → файл), разруливании **дубликатов** в наследуемых данных (представления, дедупликация в коде) и сопутствующей документации.
- **Почему GraphQL:** один типизированный контракт для фронта, запросы под сценарий экрана, меньше «ручного» набора CRUD-эндпоинтов. Минусы известны: возможны тяжёлые запросы (контроль — проектирование схемы, пагинация, при необходимости лимиты стоимости), выше порог входа для новых разработчиков по сравнению с классическим REST.
- **Безопасность и контур:** кроме модели AD/домена, критична **сетевая изоляция** — портал в **корпоративном периметре** и через **VPN**; это сознательное дополнение к техническим мерам на уровне приложения (не полагаться на «открытый в мир» GraphQL без крайней необходимости).

Результат: пользователи продолжают вести проекты, **загружать и актуализировать** инженерную информацию несмотря на **прекращение поддержки** прежних продуктов и внешние ограничения на поставку ПО; цепочка **интеграция → база портала → интерфейс** закрывает сценарии просмотра и обновления данных в привычном для EPC виде.

Итоги команды (помимо кейсов выше; под руководством тимлида)

Снижение рисков vendor lock-in и рост модульности; развитие тестирования и

CI/CD; улучшение производительности и использования ресурсов на отдельных контурах после профилирования и целевых доработок.

Wissance — Co-founder / Senior Backend Developer (.NET)

Февраль 2022 — сентябрь 2023 (1 год 8 месяцев)

- Кроссфункциональные задачи: анализ, документирование, разработка, оценка затрат — по большей части цикла **SDLC**.

ООО «Электрон-сервис» — техник-программист

Август 2020 — февраль 2022 (1 год 7 месяцев)

- Стек: **.NET Core**, **Entity Framework Core**, **REST API**, GitLab, TFS.
- Разработка веб-сервисов и служб; **SOA**; разработка контроллеров **ASP.NET**.
- Отладка и доработка хранимых процедур **MS SQL Server**; SQL-скрипты для тестирования.
- Проект на стадии поддержки и развития (приложению ~6 лет на момент работы).

Газпром нефть — Цифровые решения — ведущий разработчик .NET

Сентябрь 2017 — март 2020 (2 года 7 месяцев)

Сфера: система электронного хранения документов (ЭХД) на базе **IBM Document Manager**; решение закрывало **полный цикл**: от **сканирования первичной документации** (бухгалтерские и смежные документы) — собственное ПО вокруг **TWAIN** и **потоковых сканеров** — до **интеграции с SAP** и контроля **проведения документов** в учётной системе.

Задачи

1. Разработка и сопровождение модулей ЭХД: **WCF**-службы, интеграции, по мере развития — выделение **микросервисов** из монолитов.
2. Рефакторинг, приведение кода к **SOLID**; многопоточность (сканирование, загрузки, фоновые сценарии).
3. Использование **.NET Reflection** для нетривиальной отладки и архитектурных обходов (в т.ч. взаимодействие с драйвером сканера и **UI**-потоками).

Архитектура (контекст)

- Серверная часть ECM на **Java**; для доступа к классам платформы из **.NET** использовался **IKVM.NET** (мост JVM ↔ CLR).
- Два режима работы с хранилищем: **высокоуровневый клиентский сценарий** (удобнее, но требует **запущенного клиента ECM в браузере**) и **низкоуровневое API доступа к содержимому** (без браузера, сложнее в реализации, больше контроля).

- У платформы — свои средства форм и **плагинов** на действия пользователя (например, нажатие кнопки на форме); этим механизмом активно пользовались.
- Система была **унаследованной** (развивалась несколько лет до прихода); документация была, но **скудная** и полностью **ручная**; контроль версий — **TFS**, работа через экосистему **Visual Studio** / **Visual Studio Team Services**.

Стек: .NET Framework 2.0–4.8, **MongoDB**, **Quartz.NET**. На **MongoDB** и **Quartz** был выстроен **конвейер обработки документов**: в коллекциях создавались задачи, **фоновые сервисы** переносили их между коллекциями, продвигая пакеты по **бизнес-процессу**.

Кейс: назначение «куратора» пакета документов (STAR)

Ситуация: у пакета документов есть **ответственный (куратор)** по срокам обработки; нужно было **вычислить** этого человека по **оргструктуре**: для линейного сотрудника — **куратор его непосредственный руководитель**; если руководителя нет — взять **следующий уровень** вверх по иерархии.

Цель: корректное определение куратора по **дереву подчинённости** без дублирования логики при появлении нового клиентского интерфейса.

Действия:

- Реализовал **плагин** для ECM: обход **XML** с корпоративной иерархией через **System.Xml**, классический **обход дерева**.
- При появлении **настольного клиента** на **WinForms** с **System.Reactive** и сценариями работы с **пакетами документов** тот же функционал потребовалось встроить в новый контур. По согласованию с **тимлидом** вместо переписывания логики предложено **переиспользовать** уже существующий плагин: он был по сути **консольным приложением** с чётким контрактом вызова — достаточно было **корректно вызвать** его из нового UI.

Результат: команда **сэкономила время** на повторной реализации бизнес-правил; единая логика куратора осталась в одном месте.

Кейс: собственный просмотрщик пакетов и unit-тесты (STAR)

Ситуация: со временем стало ясно, что **исходный UX платформы** сильно **ограничивает** сценарии: появилось требование, чтобы **пакет документов** мог содержать не только **файлы** (в т.ч. **PDF**), но и **вложенные пакеты** — встроенными средствами **высокоуровневого API ECM** это было неудобно или невозможно в нужном виде.

Цель: уйти от опоры на **высокоуровневое API** и реализовать **свой настольный просмотрщик пакетов** поверх **низкоуровневого API хранилища** (того же контура, что описан в архитектуре выше), с приемлемым **UX** под новые правила композиции пакетов.

Действия:

- Спроектировал и участвовал в реализации клиента: работа с деревом пакетов и вложенностью на **низкоуровневом API**, без прежних ограничений интерфейса «из коробки».
- Параллельно встала задача **unit-тестирования**: изначально **автотестов не было**, проверки — **вручную**. В ходе экспериментов удалось настроить **Microsoft Fakes** так, что стабы «увидели» **COM-интерфейсы** из библиотеки вендора; на этом построен **слой абстракции** над API и **большой набор тестов**, который заметно **снизил** регрессии и ручной труд.
- В перспективе тот же слой абстракции позволил перейти к **MVVM** — хорошо сочетается с **System.Reactive** (**Rx.NET**) в настольном клиенте.

Результат: управляемая архитектура UI и тестируемость вместо «только ручных прогонов»; основа для дальнейшего развития клиента без жёсткой привязки к деталям COM/API в коде представления.

Кейс: зависание, дамп памяти и Reflection (STAR)

Ситуация: сканирование строилось на библиотеке **TWAIN.NET** (обёртка над **TWAIN**), вызовы шли **из фонового потока**. Приложение начало **зависать** «в никуда»: в логах тишина, **исключений нет**, стабильно **воспроизвести** на месте не получалось; первым подозревали **свежие изменения** в коде. Оформлен **инцидент наивысшего приоритета**; расследование тянулось **несколько дней**.

Цель: установить причину **взаимоблокировки** и убрать её без многомесячного «угадывания».

Действия: предложил **снять дамп памяти** зависшего процесса и разобрать **стеки потоков**. По дампу вскрылось: через цепочку зависимостей приложение **подписывалось на уведомления об изменении пользовательских настроек Windows**; по **документации WinAPI** такие сообщения должны обрабатываться **главным (UI) потоком**, а фактически цепочка приводила к тому, что обработка оказывалась **не там** — и при наступлении события всё **стопорилось**. Подписку удалось найти через **.NET Reflection** и **отключить несколькими строками** (отписка от обработчика).

Результат: зависание устранено. **Воспроизведение** оказалось неочевидным: достаточно было включить **слайдшоу обоев рабочего стола** — при **смене картинки** система посылает нужное уведомление, и баг проявлялся стабильно.

Итоги (помимо кейсов)

Рефакторинг монолитов и движение к **микросервисной** модели там, где это оправдано; опыт интеграции **ECM (Java)** с **.NET**, фоновых пайплайнов на **MongoDB** / **Quartz** и учётных контуров (**SAP**); **настольный клиент** пакетов с **MVVM**, **Rx.NET** и **unit-тестами** на базе **Microsoft Fakes** и абстракции над **COM**-доступом к API хранилища; **отладка по дампу памяти**, сочетание **многопоточности**, **WinAPI**/UI-потока и **Reflection** для поиска скрытых подписок в стороннем коде.

Образование

Омский государственный университет им. Ф.М. Достоевского, Омск.

Математический факультет — до 2019

Прикладная математика и информатика · неоконченное высшее

Физический факультет — до 2017

Прикладная математика и физика · неоконченное высшее

Факультет компьютерных наук — до 2013

Вычислительные машины, комплексы, системы и сети · неоконченное высшее

Навыки

Языки и платформы: C# (8+ лет), .NET Framework 2.0–4.8, .NET Core, .NET 9, ASP.NET Core, ASP.NET MVC, Blazor WebAssembly, Entity Framework / EF Core, ADO.NET, LINQ, SQL

Архитектура и практики: микросервисы, SOA, REST API, GraphQL (HotChocolate), WCF, SOLID, Design Patterns, DDD (элементы), feature-based архитектура, Dependency Injection, unit / integration тестирование, Agile (DoR / DoD / AC), BPMN

Данные и интеграции: SQL Server, MongoDB, Dapper, RepoDb, Linq2db, RabbitMQ, CSV / XML / XSD пайплайны, SAP-интеграции

AI и инструментарий агентов: разработка MCP-серверов (Model Context Protocol), Roslyn API, Debug Adapter Protocol (DAP), Whisper-интеграция, prompt engineering, structured context management, AI safety guardrails

Отладка и системный уровень: анализ дампов памяти, thread debugging, .NET Reflection, COM interop, TWAIN, фоновая обработка (Quartz.NET), реактивное программирование (Rx.NET / System.Reactive), MVVM

Инструменты: Git, GitHub, GitLab, Docker (базово), CI/CD, PowerShell, Cursor IDE, VS Code, Visual Studio

Языки: русский — родной; **английский — B2** (средне-продвинутый)

AI-assisted engineering environment

Я выстроил рабочую среду, в которой AI используется как **ускоритель инженерной работы**, а результат остаётся **проверяемым и воспроизводимым**. Фокус — не «генерация кода ради генерации», а **процесс и инфраструктура**: контекст, знания, ограничения, проверки качества и повторное использование решений.

Что именно построено

- **База знаний и «память» среды:** единое место для решений, соглашений, разборов проблем, паттернов, примеров, чеклистов и known pitfalls. Это снижает повторяемость вопросов и ускоряет возврат в контекст после переключений/пауз.
- **Стандарты работы с контекстом:** правила «что считается фактом», как фиксировать решения и причины (decision log), как дозировать контекст, чтобы получать стабильный результат без лишнего шума.
- **Guardrails (безопасность и качество):** анонимизация и запрет на чувствительные данные; явное разделение «гипотеза / проверено»; обязательная валидация результата через сборку/тесты/диагностики и ручную проверку там, где это необходимо.
- **Интеграция в SDLC:** AI используется внутри обычного цикла разработки — формулировка задач, PR-описания, тест-планы, технические заметки, разбор ошибок сборки/тестов и регрессий.
- **Повторное использование решений:** шаблоны формулировок, структуры документов и переиспользуемые «строительные блоки», чтобы новые задачи решались быстрее и в едином стиле.

Выводы, к которым пришли в ходе совместной работы

- AI наиболее эффективен для **рутины и структуры** (черновики документов, систематизация, triage, шаблоны и однотипные изменения), а не для «магического» решения сложной бизнес-логики без контекста.
- Качество результата определяется не моделью, а **дисциплиной проверки**: тесты/сборка/диагностики + чёткие критерии готовности дают предсказуемость.
- В долгую выигрывает не «одна удачная подсказка», а **среда**: база знаний + стандарты + guardrails → меньше когнитивной нагрузки, меньше повторных ошибок, быстрее доставка.

Что это демонстрирует

Knowledge management, инженерная дисциплина, системное мышление, автоматизация процесса разработки, формализация требований/DoD, качество и безопасность при использовании AI.

Память KB: слои L0–L3 и домены

Сжатое описание **многослойной** модели и контракта загрузки: как во входном обзоре к базе знаний (без full load), но **без** перечисления файлов и путей.

Ниже — **mermaid**-схемы: в **HTML** и в **PDF, напечатанном из браузера**, они отрисовываются; в **DOCX** Pandoc обычно не выполняет JS — смысл продублирован списком.

- **L0 — всегда в силе:** целостность, эпистемика, ядро при крахе барьеров, принципиальная ясность; **не** зависит от текущей задачи.
- **L1 — оперативная память по срезам scope:** цепочка **status → playbook → matrix → kb**; сначала компактные артефакты, тяжёлые `kb-*` только по

явному запросу.
- **L2 — архив и evidence:** ревизии, батчи правил, evidence-документы — когда не хватает фактов или нужна история (без «поднять всё при старте»).
- **L3 — семантика:** роутинг по **граням контекста**; перенос между **мирами** (доменами) — только через **явные границы**.
- **Домены и связность:** Git, PR review, HCI, Developer Experience, IT, Knowledge Engineering, психология, авиация, чтение, целостность под давлением и др. — связаны через **единый индекс** и тот же контракт загрузки.

Открытые проекты (MCP)

Публичные репозитории на GitHub — серверы **Model Context Protocol** (MCP) для Cursor и совместимых сред: дают ИИ-ассистенту доступ к инструментам (сборка, отладка, семантика C# и т.д.), а не только к тексту файлов.

Схема карточек (Purpose / Motivation / Vision и подсказки для агента): open-projec ts/README.md.

- **RoslynMcp** — MCP поверх Roslyn: диагностики, quick fixes, переход к определению, поиск вхождений, переименование, структура solution. **Зачем:** чтобы ассистент опирался на семантику C# и предлагал правки согласованно с компилятором.
  - **Purpose:** дать модели доступ к тому же слою анализа кода, что и IDE (Roslyn), а не только к тексту файлов.
  - **Motivation:** меньше правок, ломающих сборку, и меньше «угадывания» API без проверки компилятором.
  - **Vision:** основной канал работы с C# в Cursor там, где подключён MCP: диагностики → code actions → применение, навигация по символам.
  - **Scope:** решение/проект, открытые документы, операции Roslyn (в рамках возможностей сервера).
  - **Non-goals:** замена полноценной IDE, запуск приложений, отладка (это другие MCP).
  - **Для агента:** при ошибках/варнингах сначала `roslyn_get_diagnostics` и code actions, а не только `dotnet build`; позиции в тулах — 1-based; не смешивать с текстовым grep как с заменой семантики.
- **dotnet-debug-mcp** — отладка .NET через DAP (netcoredbg): брейкпоинты, запуск под отладчиком, стек, переменные, шаги, continue/stop. **Зачем:** воспроизводимая отладка из чата без ручного копирования стек-трейсов.
  - **Purpose:** одна отладочная сессия через DAP к целевому .NET-процессу или запуску под отладчиком.
  - **Motivation:** агент и человек видят одни и те же факты (стек, переменные), если интеграция с IDE выстроена; иначе — хотя бы предсказуемый контур из чата.

  - **Vision:** тот же «единый слой» состояния, что и UI IDE (брейкпоинты, останов); в экосистеме CascadeIDE см. `Financial/software/open/cascade-ide/docs/debug-human-agent-parity-v1.md`.
  - **Scope:** netcoredbg, брейкпоинты из JSON workspace, attach/launch по договорённости с хостом.
  - **Non-goals:** UI отладчика внутри другого приложения (если не связали по дизайну); замена системного отладчика во всех сценариях.
  - **Для агента:** перед пересборкой целевого DLL — завершить сессию (`debug_stop`), иначе PDB занят; attach vs launch; не убивать netcoredbg извне. Параметры и ограничения — README репозитория.

- **dotnet-build-test-mcp** — `dotnet build` / `dotnet test` с очередью и структурированным выводом ошибок и результатов тестов. **Зачем:** чтобы ИИ видел суть падений сборки и тестов без «простыни» лога.

  - **Purpose:** стандартизованный вызов сборки и тестов с очередью (single-flight) и компактным результатом для модели.
  - **Motivation:** агент получает список ошибок/упавших тестов без ручного парсинга консоли.
  - **Vision:** надёжная очередь на тяжёлых репозиториях; при необходимости — асинхронные job и чтение лога чанками.
  - **Scope:** `dotnet build`, `dotnet test`, путь к solution/каталогу.
  - **Non-goals:** семантика C# и рефакторинги (это Roslyn MCP); пошаговая отладка (это dotnet-debug-mcp).
  - **Для агента:** передавать `solution_path`; крупный `raw_output` только если явно нужен; не дублировать проверку «как компилятор» там, где уже есть Roslyn.

- **webcam-mcp** — захват веб-камеры и аудио, анализ и транскрипция (Whisper). **Зачем:** мультимодальные сценарии (голос/видео) в цепочке с ассистентом.

  - **Purpose:** локальный захват медиа и производные артефакты (кадры, wav, при необходимости mp4) в workspace.
  - **Motivation:** сценарии «показать экран/себя», голосовой ввод, последующий разбор без ручной загрузки файлов в чат.
  - **Vision:** предсказуемые пути сохранения, опционально связка с отдельным analysis-MCP для OCR/отчётов.
  - **Scope:** устройства по индексам, параметры длительности/FPS, форматы из README репозитория.
  - **Non-goals:** облачная обработка по умолчанию; замена системных настроек приватности ОС.
  - **Для агента:** почти всегда нужен `workspace_path`; модель Whisper — из env или параметра; проверять README на сплит capture/analysis, если репозиторий разделён.

Лицензии и детали — в README соответствующих репозиториев.

Дополнительно

Медицинская информация (прозрачно для работодателя): инвалидность **I группы** (ДЦП; передвижение с костылем под локоть). На выполнение трудовых обязанностей разработчика **не влияет** — пишу заранее, чтобы не было сюрпризов на этапе оффера.

Помимо разработки могу помочь с:

- автоматизацией создания документации;
- выстраиванием процессов «без боли» для участников;
- внедрением здравого Agile, разбором Agile-манифеста и кроссфункциональной командой.