Детальный план (v4): "ИИ в разработке: Компилятор для Идей"

Главная тема (Нарратив): Мы входим в эру "программирования по намерению" (Software 3.0). LLM — это "компилятор" для наших идей. Успех зависит не от мощности LLM, а от нашей способности дать ей точный "язык" (MCP, XML) и факты (RAG).

Аудитория: Смешанная (Менеджеры и Проггеры).
Цель: Вызвать аппетит, показать прикладные инструменты, дать дорожную карту для on-premise внедрения.
Время: 90 минут + 30 минут Q&A.

Часть 1: Введение и Контекст (15 минут)

Цель: Задать тон, вовлечь аудиторию через историю и показать, что это — неизбежность.

1.1. Хук: "Новый Компилятор" (5 мин)

- **Абстракция (Пример):** Эволюция уровня абстракции в программировании (Ассемблер $\to$ C $\to$ Python).
- **Тезис:** LLM — это следующий скачок абстракции. Переход от кода к **намерению**. LLM — это "компилятор", который переводит наше намерение (на естественном языке) в рабочий код.

1.2. Движение истории (5 мин)

- **Тема:** Краткая история и почему "взрыв" произошел сейчас (Transformer).
- **Тезис:** Это не просто "прогресс", это **фундаментальное архитектурное изменение**.

1.3. Нынешнее состояние (Статистика) (5 мин)

- **Тема:** Это уже не игрушки, это индустрия.
- [Слайд] Данные (GitHub Copilot, опросы): о росте производительности и внедрении LLM-инструментов.
- **Тезис:** Инструменты внедрены, и те, кто их игнорирует, уже начинают отставать.

Часть 2: Карта Мира LLM и Реалии (15 минут)

Цель: Синхронизировать терминологию и дать контекст ограничений (on-premise).

2.1. Модели: Типы, Размеры, Производители (5 мин)

- **Типы/Размеры:** LLM (OpenAI) vs. SLM (Llama 3, Falcon).

    - **Тезис:** SLM критически важны для on-premise решений и могут быть эффективнее в узких задачах (кодинг).
- **Производители:** Западные vs. Российские (Яндекс, Сбер).

2.2. Варианты Инференса (On-premise!) (5 мин)

- **Облако (API):** Плюсы/Минусы (Безопасность!).
- **On-premise (Локально):** Плюсы (Безопасность, Контроль, Ресурсы).
- **Тезис:** On-premise — это не "ограничение", а **бизнес-требование**.

2.3. Взаимозаменяемость (5 мин)

- **Тезис:** Для 80% формальных задач (рефакторинг, доки) модели взаимозаменяемы. Успех зависит не от модели, а от *данных* и *процессов*.

Часть 3: Практика для Разработчиков: "Как управлять LLM" (45-50 минут)

Цель: Дать инженерный взгляд: как LLM "думает" и как его "форматировать" для надежного результата.

3.1. Интерфейсы: Copilots и CLI (5 мин)

- **Copilots (IDE):** Плюсы (в потоке) vs. Минусы (ограниченный контекст).
- **CLI:** Гибкость, возможность интеграции в CI/CD (например, для авто-документации).

3.2. Ключевой Урок: LLM — не Редактор (10 мин)

- **Проблема:** Задачи, простые для человека, сложны для LLM (например, "замена текста").
- **Тейк:** LLM — это **генеративная** модель, которая предсказывает следующее слово, а не выполняет скрипт "поиск-замена".
- [Слайд] **Что происходит внутри:** Объяснить на примере: LLM генерирует *новый* текст, что ведет к "творческому" рефакторингу, пропускам и галлюцинациям.
- **Совет:** Для детерминированных задач — используйте Regex/AST. Для не-детерминированных — **жестко ограничьте LLM**.

3.3. Управление Галлюцинациями и Стилем (ICL/Few-shots) (10 мин)

- **Галлюцинации:** LLM не *знает*, он *предсказывает* наиболее вероятную последовательность, что иногда приводит к "правдоподобной, но неверной" информации.
- **In-Context Learning (ICL) / Few-Shots:**
    - **Объяснение:** "Обучение примерами" прямо в промпте.
    - **Вывод:** ICL — наш основной инструмент для **управления стилем, тоном и**

поведением LLM.

3.4. "Язык" для LLM: Структурированный промптинг (XML) (5 мин)

- **Тема:** Как "заключить" LLM в рамки.
- **Тейк: Принцип "Ограничение как Качество"**. Каждое ограничение (тег, формат) повышает надежность.
- **Пример:** Использование XML/JSON для передачи кода, ошибок и инструкций (см. пример в предыдущем плане).
- **Тезис:** Данные нужно готовить **под LLM**; он хорошо воспринимает структурированные форматы.

3.5. УГЛУБЛЕНО: MCP (Протокол Контекста) (10 мин)

- **Тема:** Как стандартизировать промптинг.
- **Тезис:** MCP — это **формат промпта**, который *гарантирует* аккуратный, машиночитаемый ответ. Это наш **ГОСТ**.
- **Что внутри:** (Контекст до/после курсора, путь файла, ожидаемый формат ответа).
- **Вывод:** MCP превращает LLM из "чат-бота" в **предсказуемый инструмент для автоматизации**.

3.6. "Мозг" для LLM: Агенты (5 мин)

- **Тема:** Заставляем LLM "думать" и "действовать".
- **Тезис:** Агент = LLM + Инструменты (Toolbox).
- **Схема ReAct:** Мысль $\to$ Действие (запуск кода, CLI, API) $\to$ Результат $\to$ Ответ.
- **Вывод:** Агенты — это то, как LLM решает сложные, многоэтапные инженерные задачи.

Часть 4: Low-code для Скорости и Прототипов (10 минут)

Цель: Ответить на запрос менеджеров про "MVP за 1-2 дня" и дать честную оценку.

4.1. Low-code / No-code системы с LLM (n8n и др.)

- **Low-code (Плюс):** Идеально для **MVP**, внутренних сервисов, быстрой связи API.
- **Low-code (Минус - "Боль"):**
  - **Замороченность с отладкой** (сложно понять, где сломалась "коробочка").
  - Перегруженность на сложных объектах.
- **Тейк:** Low-code (n8n) vs. Pro-code (Agents). Выбор инструмента под задачу.

Часть 5: Стратегия Внедрения и Первые Шаги (15 минут)

Цель: Дать реалистичный план внедрения и показать, куда двигаться.

5.1. RAG vs. Fine-tuning (5 мин)

- **RAG (Retrieval-Augmented Generation):**
  - **Объяснение:** "Open-book" тест. Мы даем LLM **факты** (наша кодовая база) и просим *суммировать*.
  - **Тезис:** Наш путь — это RAG. Он борется с галлюцинациями, использует *ваши* данные и не требует обучения.
- **Fine-tuning (Дообучение):**
  - **Проблема:** Бесполезно малыми мощностями. Дорого.
  - **Место:** Полезно для *дообучения стиля* или создания *локальных SLM-продуктов*.
- **Итог:** Внимание потратить на **подготовку данных и правил (RAG)**, а не на дообучение.

5.2. Управление Ожиданиями и "Low-Hanging Fruit" (10 мин)

- **Управление ожиданиями:** Сначала будет *падение* продуктивности (учеба), потом *рост*. Требуется время на шлифовку процессов.
- **Тейк: LLM как "Дешевый Аналитик Легаси"**. Самое выгодное применение LLM — разбор старого, нетестируемого кода и генерация покрытия (тесты, документация).
- **Топ-3 для старта (Наименее болезненно):**
  1. **Документация:** Генерация JSDoc, README по коду.
  2. **Генерация тестов:** Покрытие легаси-кода.
  3. **Рефакторинг:** Использование MCP для стандартизации.
- **Новое направление: LLM как Фабрика Синтетических Данных** (для тестовых сценариев и Fuzz-тестирования).

Часть 6: Заключение и Q&A (5 мин + 30 мин)

6.0. Финальный Тезис: Software 3.0 (Карпатый) (5 мин)

- Тезис: Мы находимся на ступени Software 3.0. Мы перешли от написания кода к кураторству данных и процессов.
  *
- **Большая Идея:** LLM в разработке — это **формирующийся рынок**. Его нельзя "изучить" со стороны, его можно только **понять, работая с ним**.
- **Призыв к действию:** "Стоит **сразу начать применять его**, пусть и небольшими дозами, чтобы в процессе сформировать понимание и платформу для развития, которое уже четко очерчено мировыми трендами".

6.1. Трейлер и Q&A

- **Трейлер (Анонс):** "В следующий раз: Deep Dive по RAG для нашей кодовой базы / Строим своего первого Агента".

- **Q&A** (30 мин)