AI-Assisted Development Adoption

Введение

Figure 1 Figure 2 Figure 3 Figure 4

Figure 5 Figure 6 Figure 7 Figure 8

Steve Yegge описал 8 уровней adoption AI-инструментов для разработчиков. Большинство инженеров сегодня находятся на уровнях 2-3: используют coding agent в IDE, иногда с YOLO mode. Цель этого краш-курса — повысить качество работы с AI-ассистентами / агентами, дать знания и практики для перехода на последующие уровни, где разработчик работает с CLI-агентами и управляет несколькими параллельными инстансами.

Уровень	Описание
1	Zero AI — только автокомплит, иногда Chat
2	IDE agent с разрешениями (Y/N на каждое действие)
3	IDE agent, YOLO mode — разрешения отключены
4	Wide agent в IDE — агент занимает весь экран
5	CLI, single agent, YOLO — diffs прокручиваются
6	CLI, multi-agent — 3-5 параллельных инстансов

7	10+ agents, hand-managed
8	Свой оркестратор

Материалы:

- Steve Yegge — Welcome to Gas Town - первоисточник (можно почитать вводную часть).
- Andrej Karpathy: Software Is Changing (Again) (старое, но расскажет о мышлении)

1. Промпт-инжиниринг

Промпт-инжиниринг — это навык формулировать запросы к LLM так, чтобы получать предсказуемые и качественные результаты. Для инженеров это означает переход от «спросил — получил что-то» к системному подходу, где вы контролируете поведение модели через структуру запроса, примеры и ограничения.

Ключевые техники: few-shot примеры, chain-of-thought reasoning, разбиение сложных задач, указание формата вывода. Понимание этих основ критично перед переходом к агентским инструментам — они все строятся на тех же принципах.

Материалы:

- https://www.promptingguide.ai/ru — комплексный гайд на русском

2. Структура промпта: роль, контекст, задача, критерии приемки

Эффективный промпт для coding agent состоит из четырёх блоков: **роль** (кем является агент и какие у него компетенции), **контекст** (релевантная информация о проекте, стеке, ограничениях), **задача** (что конкретно нужно сделать), **критерии приемки** (как понять, что задача выполнена правильно).

Этот фреймворк особенно важен для агентских инструментов, где промпт определяет не просто ответ, а целую цепочку действий. Чем точнее сформулированы критерии приёмки, тем меньше итераций понадобится для достижения результата.

Материалы:

- https://apidog.com/blog/deepseek-prompts-coding/ — PPFO Framework (Purpose, Planning, Format, Output)

- https://platform.openai.com/docs/guides/prompt-engineering — OpenAI guide с примерами структур
- https://www.youtube.com/watch?v=cMR2c3vQRAc

3. Инструменты: обзор и выбор

Рынок AI coding tools разделился на несколько категорий: CLI-агенты (работают в терминале, максимальная автоматизация), IDE-integrated (привычный workflow с AI-усилением), и chat-based (для исследования и быстрых вопросов). Выбор инструмента зависит от задачи: для рутинных правок подойдёт IDE, для масштабных рефакторингов — CLI-агент.

Комбинирование инструментов — норма: например, DeepSeek для дешёвых exploratory задач, Claude Code для сложной автоматизации, ChatGPT для объяснений и документации.

Инструмент	Тип	Лучше всего для	Модель/Provider
Claude Code	CLI	Большие codebase, автоматизация, multi-file changes	Claude (Anthropic)
OpenCode	CLI	Open-source альтернатива, multi-provider, terminal-first	Any (Claude, GPT, Gemini, local)
Cursor	IDE	Привычный VS Code workflow, команды в процессе перехода	Multi (GPT, Claude, Gemini)
Google Antigravity	IDE	Multi-agent orchestration, async tasks, бесплатно	Gemini 3, Claude, GPT-OSS
Codex CLI	CLI	OpenAI ecosystem, multimodal (screenshots), enterprise	GPT-5-Codex
Kilo Code	VS Code ext	Гибкие режимы, бюджетные проекты, no lock-in	Any (400+ моделей)
ChatGPT	Chat	Quick questions, learning, объяснение кода	GPT-5.x
DeepSeek	API/Chat	Cost-sensitive, open-source, self-hosting	DeepSeek V3/R1

Материалы:

- https://cursor.com/features — возможности Cursor

    - Cursor just changed forever
    - How I code with AI right now
- https://opencode.ai/docs/ — документация OpenCode
- https://antigravity.google/ — Antigravity
    - Google just dropped their Cursor killer (FREE Gemini 3 Pro???)
- https://developers.openai.com/codex/cli/ — Codex CLI docs
- https://www.arsturn.com/blog/kilocode-vs-cline-vs-claude-code-choosing-your-ai-coding-sidekick — сравнение Kilo/Cline/Claude Code

4. Режимы работы: планирование и исполнение

Эффективная работа с coding agents строится на разделении двух фаз: **планирование** (Plan) и **исполнение** (Act/Execute). В режиме планирования агент анализирует задачу, задаёт уточняющие вопросы, исследует codebase и формирует план действий — без внесения изменений. В режиме исполнения агент реализует план: редактирует файлы, запускает команды, итерирует до достижения результата.

Разделение режимов даёт несколько преимуществ: вы можете проверить план до начала изменений, использовать разные модели для разных фаз (например, Gemini для планирования, Claude для исполнения), и избежать ситуации, когда агент «уходит не туда» на сотни строк кода. Многие инструменты поддерживают это нативно (Kilo Code: Architect/Code modes, OpenCode: Plan/Build, Cursor: custom modes), в других можно настроить через промпты и sub-agents.

Материалы:

- https://carlrannaberg.medium.com/my-current-ai-coding-workflow-f6bdc449df7f — custom Planner/Executor modes в Cursor
- https://research.aimultiple.com/agentic-coding/ — task-based execution model, plan.md workflow
- https://ed-wentworth.medium.com/how-im-using-agentic-coding-with-claude-and-cursor-in-real-world-projects-b4b6694c132d — story → tasks workflow

5. Правила, настройки, кастомизация

Claude Code Superpowers

CLAUDE.md (или AGENTS.md для других инструментов) — это файл с инструкциями, который агент читает в начале каждой сессии. Здесь описываются команды проекта, code style, архитектура, запреты. Это «persistent memory» агента между сессиями.

Slash-команды (/commands) позволяют создавать переиспользуемые workflows: /fix-issue, /review, /test. Они хранятся в `.claude/commands/` и могут содержать сложные

многошаговые инструкции. Правильная настройка этих механизмов — ключ к масштабированию работы с агентом.

Материалы:

- https://www.humanlayer.dev/blog/writing-a-good-claude-md — best practices для CLAUDE.md
- https://agents.md/ - универсальное средство, но может потребоваться отдельный тюнинг для некоторых агентов.
- https://www.builder.io/blog/claude-md-guide — гайд по структуре
- https://code.claude.com/docs/en/best-practices — официальная документация Claude Code
- https://blog.sshh.io/p/how-i-use-every-claude-code-feature — sub-agents, commands, hooks
- https://www.producttalk.org/how-to-use-claude-code-features/ — slash commands, skills, plugins
- https://github.com/sammcj/agentic-coding/blob/main/Claude/CLAUDE.md - пример.
- https://github.com/maddevsio/shared_cursor_rules - пример.

6. Повышение качества через инструменты самопроверки

LLM генерируют код, который «выглядит правильно», но может содержать subtle bugs, security issues, или нарушать conventions проекта. Решение — интегрировать в workflow агента детерминистические инструменты: линтеры, type checkers, тесты, security scanners.

Практически это реализуется через hooks (автоматический запуск после изменений) или явные инструкции в CLAUDE.md: «После каждого изменения запусти `npm run lint && npm test`». TDD-подход особенно эффективен с агентами: сначала тесты, потом реализация, агент итерирует пока тесты не зелёные.

Материалы:

- https://www.humanlayer.dev/blog/writing-a-good-claude-md — «Never send an LLM to do a linter's job»
- https://factory.ai/news/using-linters-to-direct-agents — как автоматика может бить “агенту по рукам” и экономить ваше время.
- https://blog.sshh.io/p/how-i-use-every-claude-code-feature — управление context window
- https://agentic-coding.github.io/ (некоторые пункты обсуждаемы, но в целом очень показательный документ / манифест)
- https://habr.com/ru/articles/902422/

7. Организация контекста в проекте

Агент работает с тем контекстом, который вы ему даёте. Плохо организованный проект = агент тратит токены на поиск информации и делает неверные предположения. Хорошо организованный = агент быстро находит нужное и следует conventions.

Ключевые практики: README с описанием архитектуры, CLAUDE.md с командами и правилами, отдельные docs для сложных систем, чистая структура папок. Важно: контекстное окно ограничено (128K-200K токенов для большинства моделей), поэтому информация должна быть компактной и релевантной.

Материалы:

- https://dometrain.com/blog/creating-the-perfect-claudemd-for-claude-code/ — структура документации для AI
- https://www.builder.io/blog/claude-md-guide — иерархия memory files
- https://docs.cline.bot/prompting/cline-memory-bank — что такое memory bank, как концепция (не идеальная, но тем не менее)
- Spec-Driven Development:
    - https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/
    - https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

8. Продвинутые концепции: Context Engineering, Steering агентов, Ralph Loop

Context Engineering — управление тем, какая информация попадает в контекстное окно агента на каждом шаге. Это включает: что загружать в начале (instructions, memories), что добавлять по ходу (tool outputs, search results), что сжимать или удалять (summarization, trimming).

Steering — контроль отклонения поведения агента через инструкции и контексты, checkpoints, sub-agents. Вместо одного большого запроса — цепочка: research → plan → implement → verify. Каждый этап можно проверить и скорректировать до того, как агент зайдёт слишком далеко в неверном направлении.

Материалы:

- https://www.youtube.com/watch?v=rmvDxxNubIg — видео о context engineering
- https://www.youtube.com/watch?v=Yr9O6KFwbW4 — Ralph Loop
- https://rlancemartin.github.io/2025/06/23/context_engineering/ — обзор с примерами