Сертификация Claude Certified Architect -- Foundations

Учебное пособие (на основе официального руководства к экзамену)

Введение

Сертификация **Claude Certified Architect -- Foundations** подтверждает, что специалист способен принимать обоснованные решения о компромиссах при внедрении реальных решений на базе Claude. Экзамен проверяет базовые знания по Claude Code, Claude Agent SDK, Claude API и Model Context Protocol (MCP) -- основным технологиям для создания production-приложений с Claude.

Вопросы экзамена основаны на реалистичных сценариях из практики: построение агентных систем для поддержки клиентов, проектирование мультиагентных исследовательских пайплайнов, интеграция Claude Code в CI/CD, создание инструментов продуктивности для разработчиков и извлечение структурированных данных из неструктурированных документов.

Целевой кандидат

Идеальный кандидат -- **архитектор решений (solution architect)**, проектирующий и внедряющий production-приложения с Claude. Требуется опыт работы от 6 месяцев со следующими технологиями:

- **Claude Agent SDK** -- мультиагентная оркестрация, делегирование субагентам, интеграция инструментов, хуки жизненного цикла
- **Claude Code** -- CLAUDE.md, MCP-серверы, Agent Skills, режим планирования
- **Model Context Protocol (MCP)** -- инструменты, ресурсы для интеграции с бэкендом
- **Промпт-инженерия** -- JSON-схемы, few-shot примеры, шаблоны извлечения данных
- **Контекстные окна** -- работа с длинными документами, мультиагентные передачи
- **CI/CD пайплайны** -- автоматические ревью кода, генерация тестов
- **Эскалация и надёжность** -- обработка ошибок, human-in-the-loop

---

Формат экзамена

Параметр	Значение
Тип вопросов	Множественный выбор (1 правильный из 4)
Оценка	Шкала 100--1000, проходной балл **720**

Штраф за угадывание	Нет (отвечайте на все вопросы!)
Сценарии	4 из 8 возможных (выбираются случайно)

Содержание экзамена: 5 доменов

Домен	**Вес**
1. Агентная архитектура и оркестрация	**27%**
2. Проектирование инструментов и интеграция MCP	**18%**
3. Конфигурация и рабочие процессы Claude Code	**20%**
4. Промпт-инженерия и структурированный вывод	**20%**
5. Управление контекстом и надёжность	**15%**

Сценарии экзамена

Сценарий 1: Агент поддержки клиентов

Вы создаёте агента для обработки возвратов, споров по счетам и проблем с аккаунтами с помощью Claude Agent SDK. Агент использует MCP-инструменты (`get_customer`, `lookup_order`, `process_refund`, `escalate_to_human`). Цель -- 80%+ решение с первого контакта при правильной эскалации.

Сценарий 2: Генерация кода с Claude Code

Вы используете Claude Code для ускорения разработки: генерация кода, рефакторинг, отладка, документация. Нужно интегрировать его с пользовательскими slash-командами, конфигурациями CLAUDE.md и понимать, когда использовать режим планирования.

Сценарий 3: Мультиагентная исследовательская система

Координирующий агент делегирует задачи специализированным субагентам: поиск в интернете, анализ документов, синтез и генерация отчётов. Система должна создавать полные отчёты с цитатами.

Сценарий 4: Инструменты продуктивности для разработчиков

Агент помогает инженерам исследовать незнакомые кодовые базы, генерировать шаблонный код и автоматизировать рутинные задачи. Используются встроенные инструменты (Read, Write, Bash, Grep, Glob) и MCP-серверы.

Сценарий 5: Claude Code для непрерывной интеграции

Интеграция Claude Code в CI/CD пайплайн для автоматических ревью кода, генерации тестов и обратной связи по пулл-реквестам. Нужно проектировать промпты с минимумом ложных срабатываний.

Сценарий 6: Извлечение структурированных данных

Система извлекает информацию из неструктурированных документов, валидирует вывод с помощью JSON-схем и поддерживает высокую точность. Должна корректно обрабатывать граничные случаи.

Сценарий 7: Архитектурные паттерны разговорного ИИ

Вы проектируете многоходовые диалоговые системы, охватывающие управление контекстным окном, сохранение инструкций на протяжении диалога, стратегии памяти, проектирование инструментов для безопасного выполнения и обработку неоднозначных или противоречивых вводных данных пользователя.

Сценарий 8: Агентные инструменты ИИ *(материал отсутствует — помогите нам его добавить!)*

Этот сценарий был упомянут кандидатами, сдававшими экзамен, но пока не охвачен в этом руководстве. Если вы встречали вопросы из этого сценария на реальном экзамене, поделитесь ими в GitHub Issues, чтобы мы могли добавить полное покрытие. Ваш вклад поможет всем, кто готовится к экзамену.

Официальная документация

Ресурс	URL
Claude API -- Messages	https://platform.claude.com/docs/en/api/messages
Claude API -- Tool Use	https://platform.claude.com/docs/en/build-with-claude/tool-use
Claude API -- Message Batches	https://platform.claude.com/docs/en/build-with-claude/message-batches
Claude Agent SDK -- Обзор	https://platform.claude.com/docs/en/agent-sdk/overview
Claude Agent SDK -- Hooks	https://platform.claude.com/docs/en/agent-sdk/hooks
Claude Agent SDK -- Subagents	https://platform.claude.com/docs/en/agent-sdk/subagents
Claude Agent SDK -- Sessions	https://platform.claude.com/docs/en/agent-sdk/sessions
Model Context Protocol (MCP)	https://modelcontextprotocol.io/
MCP -- Tools	https://modelcontextprotocol.io/docs/concepts/tools
MCP -- Resources	https://modelcontextprotocol.io/docs/concepts/resources

MCP -- Servers	https://modelcontextprotocol.io/docs/concepts/servers
Claude Code -- Документация	https://code.claude.com/docs/en/overview
Claude Code -- CLAUDE.md и память	https://code.claude.com/docs/en/memory
Claude Code -- Skills (вкл. slash-команды)	https://code.claude.com/docs/en/skills
Claude Code -- Hooks	https://code.claude.com/docs/en/hooks
Claude Code -- Sub-agents	https://code.claude.com/docs/en/sub-agents
Claude Code -- MCP Integration	https://code.claude.com/docs/en/mcp
Claude Code -- GitHub Actions CI/CD	https://code.claude.com/docs/en/github-actions
Claude Code -- GitLab CI/CD	https://code.claude.com/docs/en/gitlab-ci-cd
Claude Code -- Headless (неинтерактивный режим)	https://code.claude.com/docs/en/headless
Prompt Engineering Guide	https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/overview
Extended Thinking	https://platform.claude.com/docs/en/build-with-claude/extended-thinking
Anthropic Cookbook (примеры кода)	https://github.com/anthropics/anthropic-cookbook

---

ЧАСТЬ I: ТЕОРЕТИЧЕСКАЯ БАЗА

В этой части подробно разобрана вся теория, необходимая для успешной сдачи экзамена. Материал организован по технологиям и концепциям, а не по доменам экзамена -- это позволяет глубже понять каждую тему.

Глава 1: Claude API -- основы взаимодействия с моделью

> Документация: Messages API | Prompt Engineering

1.1 Структура API-запроса

Claude API работает по принципу "запрос-ответ". Каждый запрос к Claude Messages API содержит:

```
{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "system": "Ты -- полезный ассистент.",
  "messages": [
    {"role": "user", "content": "Привет!"},
    {"role": "assistant", "content": "Здравствуйте!"},
    {"role": "user", "content": "Как дела?"}
  ],
  "tools": [...],
  "tool_choice": {"type": "auto"}
}
```

Ключевые поля:

- `model` -- выбор модели (claude-opus-4-6, claude-sonnet-4-6, claude-haiku-4-5)
- `max_tokens` -- максимальное количество токенов в ответе
- `system` -- системный промпт (задаёт поведение модели)
- `messages` -- история разговора (обязательно передавать полную историю для поддержания связности)
- `tools` -- определения доступных инструментов
- `tool_choice` -- стратегия выбора инструмента

1.2 Роли сообщений

В массиве `messages` используются две роли для диалога и одна инструктивная роль:

- `user` -- сообщения от пользователя, включая результаты вызова инструментов (отправляются как content block `tool_result` внутри сообщения с ролью `user`, а не как отдельная роль `tool`)
- `assistant` -- ответы модели (включаются при передаче истории), включая запросы на использование инструментов (content blocks `tool_use`)
- `system` -- можно задать через верхнеуровневое поле `system` (применяется с первого хода) или внутри `messages` как `{"role": "system", ...}` (применяется начиная с этого места, с учётом правил размещения — см. ниже)

Результаты вызова инструментов не отправляются как сообщение с ролью `"tool"`. Они отправляются как сообщение с ролью `user`, содержимое которого включает content block `tool_result`:

```
{
  "role": "user",
  "content": [
    {
      "type": "tool_result",
      "tool_use_id": "toolu_01...",
      "content": "..."
    }
  ]
}
```

`system` также может появляться как роль непосредственно в массиве `messages`, а не только через верхнеуровневый параметр `system`. Это предназначено для добавления инструкций в середине разговора без инвалидации закешированного префикса из верхнеуровневого поля `system`. Действуют определённые правила размещения:

- Должна идти сразу после хода `user` (в том числе содержащего блоки `tool_result`) или хода `assistant`, завершающегося использованием серверного инструмента.
- Должна предшествовать ходу `assistant` либо быть последним элементом массива.
- Не может находиться между блоком `tool_use` и его `tool_result` -- в этом случае возвращается ошибка 400.
- Более поздние сообщения `system` (в том числе добавленные в середине разговора) имеют приоритет над более ранними и над верхнеуровневым полем `system` для последующих ходов.

Критически важно: при каждом API-запросе необходимо передавать **полную историю разговора**. Модель не сохраняет состояние между запросами -- каждый вызов независим.

1.3 Поле stop_reason в ответе

Ответ Claude API содержит поле `stop_reason`, определяющее, почему модель прекратила генерацию:

Значение	Описание	Действие
`"end_turn"`	Модель завершила свой ответ	Показать результат пользователю
`"tool_use"`	Модель хочет вызвать инструмент	Выполнить инструмент, вернуть результат
`"max_tokens"`	Достигнут лимит токенов	Ответ обрезан, возможно нужно увеличить лимит
`"stop_sequence"`	Встречена стоп-последовательность	Обработать в зависимости от логики

Для агентных систем наиболее важны `"tool_use"` и `"end_turn"` -- они управляют агентным циклом.

1.4 Системный промпт (system prompt)

Системный промпт -- специальное сообщение, задающее контекст и правила поведения модели. Он:

- Не является частью массива `messages`, а передаётся отдельно в поле `system`
- Имеет приоритет перед пользовательскими сообщениями
- Загружается один раз и действует на протяжении всего разговора
- Используется для задания роли, ограничений, формата вывода

Важно для экзамена: формулировки в системном промпте могут создавать непреднамеренные ассоциации с инструментами. Например, инструкция "всегда проверяй клиента" может заставить модель слишком часто вызывать `get_customer`, даже когда это не нужно.

1.5 Контекстное окно

Контекстное окно -- суммарный объём текста (в токенах), который модель может обрабатывать одновременно. Включает:

- Системный промпт
- Всю историю сообщений
- Определения инструментов
- Результаты вызовов инструментов

Ключевые проблемы контекстного окна:

1. **Эффект "потеря в середине" (lost-in-the-middle):** модели надёжно обрабатывают информацию в начале и конце длинного ввода, но могут пропускать данные из середины. Решение -- размещать ключевую информацию в начале или конце.
2. **Накопление результатов инструментов:** каждый вызов инструмента добавляет результат в контекст. Если инструмент возвращает 40+ полей, а релевантны только 5 -- 87% контекста тратится впустую.
3. **Прогрессивная суммаризация:** при конденсации истории теряются числовые значения, проценты и даты, превращаясь в расплывчатые "примерно", "около", "несколько".

Глава 2: Инструменты (Tools) и tool_use

> Документация: Tool Use

2.1 Что такое tool_use

`tool_use` -- это механизм, позволяющий Claude вызывать внешние функции. Модель не выполняет код напрямую -- она генерирует структурированный запрос на вызов инструмента, а ваш код

выполняет его и возвращает результат.

2.2 Определение инструмента

Каждый инструмент определяется JSON-схемой:

```
{
  "name": "get_customer",
  "description": "Ищет клиента по email или ID. Возвращает профиль клиента, включая
имя, email, историю заказов и статус аккаунта. Используйте этот инструмент ПЕРЕД
lookup_order для верификации личности клиента. Принимает email (формат:
user@domain.com) или числовой customer_id.",
  "input_schema": {
    "type": "object",
    "properties": {
      "email": {"type": "string", "description": "Email клиента"},
      "customer_id": {"type": "integer", "description": "Числовой ID клиента"}
    },
    "required": []
  }
}
```

Критически важные аспекты описания инструмента:

1. **Описание -- основной механизм выбора.** LLM выбирает инструмент на основе его описания. Минимальные описания ("Retrieves customer information") ведут к ошибочному выбору между похожими инструментами.
2. **Включайте в описание:**
   - Что именно инструмент делает и возвращает
   - Форматы ввода и примеры значений
   - Граничные случаи и ограничения
   - Когда использовать этот инструмент vs похожие альтернативы
3. **Избегайте:** одинаковых или пересекающихся описаний для разных инструментов. Если `analyze_content` и `analyze_document` имеют почти идентичные описания -- модель будет путаться.
4. **Предпочтение встроенных инструментов над MCP:** Агенты могут предпочитать встроенные инструменты (Read, Grep) вместо MCP-инструментов с аналогичной функциональностью. Чтобы этого избежать, усильте описания MCP-инструментов — укажите конкретные преимущества, уникальные данные или контекст, который встроенные инструменты не предоставляют.

2.3 Параметр tool_choice

`tool_choice` управляет тем, как модель выбирает инструменты:

Значение	Поведение	Когда использовать

`{"type": "auto"}`	Модель решает сама: вызвать инструмент или ответить текстом	По умолчанию, для большинства случаев
`{"type": "any"}`	Модель **обязана** вызвать какой-либо инструмент	Когда нужен гарантированный структурированный вывод
`{"type": "tool", "name": "extract_metadata"}`	Модель **обязана** вызвать конкретный инструмент	Для принудительного порядка выполнения

Важные сценарии:

- `tool_choice: "any"` + несколько инструментов извлечения -- модель выберет подходящий, но гарантированно вернёт структурированные данные
- Принудительный выбор -- когда нужно обеспечить конкретный первый шаг (например, сначала `extract_metadata`, потом обогащение)

2.4 JSON-схемы для структурированного вывода

Использование `tool_use` с JSON-схемами -- **самый надёжный** способ получить структурированный вывод от Claude. Он:

- Гарантирует синтаксическую корректность JSON (нет незакрытых скобок, лишних запятых)
- Обеспечивает соответствие структуре (нужные поля присутствуют)
- **НЕ гарантирует** семантическую корректность (значения могут быть неверными)

Дизайн схемы -- ключевые принципы:

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["bug", "feature", "docs", "unclear", "other"]
    },
    "category_detail": {
      "type": ["string", "null"],
      "description": "Подробности, если category = 'other' или 'unclear'"
    },
    "severity": {
      "type": "string",
      "enum": ["critical", "high", "medium", "low"]
    },
    "confidence": {
      "type": "number",
      "minimum": 0,
      "maximum": 1
    },
    "optional_field": {
      "type": ["string", "null"],
      "description": "Null если информация не найдена в источнике"
    }
  },
  "required": ["category", "severity"]
}
```

Правила проектирования схем:

1. **Required vs Optional:** помечайте поля как required только если информация всегда доступна. Обязательные поля заставляют модель фабриковать значения, если их нет в источнике.
2. **Nullable поля:** используйте `"type": ["string", "null"]` для информации, которая может отсутствовать. Модель вернёт `null` вместо выдумывания.
3. **Enum с "other":** для категоризации добавляйте `"other"` + detail string, чтобы не терять данные вне предустановленных категорий.
4. **Enum "unclear":** для случаев, когда модель не может точно определить категорию -- лучше честный "unclear" чем ошибочная категория.

2.5 Различие синтаксических и семантических ошибок

Тип ошибки	Пример	Решение
Синтаксическая	Невалидный JSON, неправильный тип поля	tool_use с JSON-схемой (полностью устраняет)
Семантическая	Итоги строк не сходятся к общему, значение в неправильном поле, фабрикация	Валидационные проверки, повтор с обратной связью, self-correction

Глава 3: Claude Agent SDK -- построение агентных систем

> Документация: Agent SDK | Hooks | Subagents | Sessions

3.1 Что такое агентный цикл (Agentic Loop)

Агентный цикл -- основной паттерн для автономного выполнения задач. Модель не просто отвечает на вопрос, а выполняет последовательность действий:

```
1. Отправить запрос Claude с инструментами
2. Получить ответ
3. Проверить stop_reason:
   - "tool_use" -> выполнить инструмент, добавить результат в историю, перейти к шагу 1
   - "end_turn" -> задача завершена, показать результат пользователю
4. Повторять до завершения
```

Это модельно-управляемый (model-driven) подход: Claude сам решает, какой инструмент вызвать следующим, на основе контекста и результатов предыдущих действий. Это отличает его от предустановленных деревьев решений, где последовательность действий жёстко задана.

Антипаттерны (чего избегать):

- Парсинг текста ассистента для определения завершения ("Задача выполнена" в тексте ответа)
- Установка произвольного лимита итераций (max_iterations=5) как основного механизма остановки
- Проверка текстового содержимого ответа как индикатора завершения

Правильный подход: единственный надёжный сигнал завершения -- `stop_reason == "end_turn"`.

3.2 Конфигурация AgentDefinition

`AgentDefinition` -- это объект конфигурации агента в Claude Agent SDK:

```
agent = AgentDefinition(
    name="customer_support",
    description="Обрабатывает запросы клиентов по возвратам и проблемам с заказами",
    system_prompt="Ты -- агент поддержки клиентов...",
    allowed_tools=["get_customer", "lookup_order", "process_refund",
"escalate_to_human"],
    # Для координатора:
    # allowed_tools=["Task", "get_customer", ...]
)
```

Ключевые параметры:

- `name` / `description` -- идентификация и описание агента
- `system_prompt` -- системный промпт с инструкциями
- `allowed_tools` -- список разрешённых инструментов (принцип минимальных привилегий)

3.3 Hub-and-spoke: координатор и субагенты

Мультиагентная архитектура строится по принципу "hub-and-spoke" (звёздная топология):

```
        Координатор
       /     |      \
  Субагент1  Субагент2  Субагент3
  (поиск)   (анализ)    (синтез)
```

Координатор отвечает за:

- Декомпозицию задачи на подзадачи
- Решение, какие субагенты нужны (динамический выбор, а не всегда весь конвейер)
- Делегирование задач субагентам
- Агрегацию и валидацию результатов
- Обработку ошибок и повторные попытки
- Коммуникацию результатов пользователю

Критический принцип: субагенты имеют изолированный контекст.

- Субагенты **НЕ наследуют** историю разговора координатора автоматически
- Весь контекст должен быть **явно передан** в промпте субагента
- Субагенты не разделяют память между вызовами
- Вся коммуникация проходит через координатора (для наблюдаемости и контроля ошибок)

3.4 Инструмент Task для порождения субагентов

Субагенты порождаются через инструмент `Task`:

```
# allowedTools координатора должен включать "Task"
coordinator_agent = AgentDefinition(
    allowed_tools=["Task", "get_customer"]
)
```

Явная передача контекста -- обязательна:

```
# Плохо: субагент не знает контекст
Task: "Проанализируй документ"

# Хорошо: полный контекст в промпте
Task: "Проанализируй следующий документ.
Документ: [полный текст документа]
Результаты предыдущего поиска: [результаты веб-поиска]
Требования к формату вывода: [схема]"
```

Параллельное порождение: координатор может вызвать несколько `Task` в одном ответе -- субагенты запустятся параллельно:

```
# Один ответ координатора содержит:
Task 1: "Поиск статей по теме X"
Task 2: "Анализ документа Y"
Task 3: "Поиск статей по теме Z"
# Все три запустятся одновременно
```

3.5 Хуки (Hooks) в Agent SDK

Хуки -- механизм перехвата и трансформации на определённых точках жизненного цикла агента.

PostToolUse -- перехватывает результат инструмента перед передачей модели:

```python
# Пример: нормализация форматов дат из разных MCP-инструментов
@hook("PostToolUse")
def normalize_dates(tool_result):
    # Конвертировать Unix timestamp -> ISO 8601
    # Конвертировать "Mar 5, 2025" -> "2025-03-05"
    return normalized_result
```

Хук перехвата исходящих вызовов -- блокирует действия, нарушающие политику:

```python
# Пример: блокировка возвратов выше $500
@hook("PreToolUse")
def enforce_refund_limit(tool_call):
    if tool_call.name == "process_refund" and tool_call.args.amount > 500:
        return redirect_to_escalation(tool_call)
```

Ключевое различие: хуки vs промпт-инструкции

Характеристика	Хуки	Промпт-инструкции
Гарантия	**Детерминированная** (100%)	**Вероятностная** (>90%, но не 100%)
Когда использовать	Критические бизнес-правила, финансовые операции, комплаенс	Общие предпочтения, рекомендации, форматирование

Пример	Блокировка возвратов >$500	"Старайся решить проблему до эскалации"

Правило: когда ошибка имеет финансовые, юридические или безопасностные последствия -- используйте хуки, а не промпты.

Глава 4: Model Context Protocol (MCP)

> Документация: MCP | Tools | Resources | Servers

4.1 Что такое MCP

Model Context Protocol (MCP) -- открытый протокол для подключения внешних систем к Claude. MCP определяет три основных типа ресурсов:

1. **Tools (инструменты)** -- функции, которые агент может вызывать для выполнения действий (CRUD операции, запросы к API, выполнение команд)
2. **Resources (ресурсы)** -- данные, к которым агент может обращаться для получения контекста (документация, схемы БД, каталоги контента)
3. **Prompts (промпты)** -- предустановленные шаблоны промптов для типовых задач

4.2 MCP-серверы

MCP-сервер -- процесс, реализующий протокол MCP и предоставляющий инструменты/ресурсы. При подключении к MCP-серверу:

- Все инструменты обнаруживаются автоматически
- Инструменты со всех подключённых серверов доступны одновременно
- Описания инструментов определяют, как модель будет их использовать

4.3 Конфигурация MCP-серверов

Проектная конфигурация (`.mcp.json`) -- для командного использования:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "jira": {
      "command": "npx",
      "args": ["-y", "mcp-server-jira"],
      "env": {
        "JIRA_TOKEN": "${JIRA_TOKEN}"
      }
    }
  }
}
```

Ключевые аспекты:

- Файл `.mcp.json` хранится в корне проекта и управляется через VCS
- Переменные окружения (`${GITHUB_TOKEN}`) используются для секретов -- сами токены не коммитятся
- Доступен всем участникам проекта

Пользовательская конфигурация (`~/.claude.json`) -- для личных/экспериментальных серверов:

- Хранится в домашней директории пользователя
- Не передаётся через VCS
- Подходит для личных экспериментов и тестирования

Выбор серверов:

- Для стандартных интеграций (Jira, GitHub, Slack) -- предпочитайте существующие community MCP-серверы
- Собственные серверы -- только для уникальных team-specific рабочих процессов

4.4 Флаг isError в MCP

При ошибке в MCP-инструменте используется флаг `isError: true` в ответе. Это сигнализирует агенту, что вызов завершился неудачей.

Структурированная ошибка (правильно):

```
{
  "isError": true,
  "content": {
    "errorCategory": "transient",
    "isRetryable": true,
    "message": "Сервис временно недоступен. Таймаут при обращении к API заказов.",
    "attempted_query": "order_id=12345",
    "partial_results": null
  }
}
```

Общая ошибка (антипаттерн):

```
{
  "isError": true,
  "content": "Operation failed"
}
```

Общая ошибка не даёт агенту информации для принятия решения -- повторить попытку? изменить запрос? эскалировать?

4.5 MCP-ресурсы (Resources)

Ресурсы -- это данные, которые агент может запрашивать для получения контекста без выполнения действий:

- Каталоги контента (список всех задач в проекте, навигация по иерархии)
- Схемы баз данных (понимание структуры данных)
- Документация (API reference, внутренние гайды)
- Сводки проблем/задач

Преимущество ресурсов: агенту не нужно делать исследовательские вызовы инструментов для понимания доступных данных. Ресурс даёт "карту" сразу.

Глава 5: Claude Code -- конфигурация и рабочие процессы

> Документация: Claude Code | Memory / CLAUDE.md | Skills | MCP | Hooks | Sub-agents | GitHub Actions | Headless

5.1 Иерархия CLAUDE.md

CLAUDE.md -- файл(ы) с инструкциями для Claude Code. Существует иерархия с тремя уровнями:

```
1. Пользовательский: ~/.claude/CLAUDE.md
   - Применяется только к данному пользователю
   - НЕ передаётся через VCS
   - Личные предпочтения, стиль работы

2. Проектный: .claude/CLAUDE.md или корневой CLAUDE.md
   - Применяется ко всем участникам проекта
   - Управляется через VCS
   - Стандарты кодирования, тестирования, архитектурные решения

3. Директорный: CLAUDE.md в подкаталогах
   - Применяется при работе с файлами в этой директории
   - Специфичные конвенции для этой части кодовой базы
```

Типичная ошибка: новый участник команды не получает проектные инструкции, потому что они были помещены в `~/.claude/CLAUDE.md` (пользовательский уровень) вместо `.claude/CLAUDE.md` (проектный уровень).

5.2 Синтаксис @path (импорт файлов)

CLAUDE.md может ссылаться на внешние файлы с помощью синтаксиса `@path`, делая конфигурацию модульной:

```
# CLAUDE.md проекта

Стандарты кодирования описаны в @./standards/coding-style.md
Требования к тестам смотри в @./standards/testing-requirements.md
Обзор проекта в @README.md и зависимости в @package.json
```

Правила синтаксиса `@path`:

- Используйте `@` непосредственно перед путём к файлу (без пробела)
- Поддерживаются относительные и абсолютные пути
- Относительные пути разрешаются относительно содержащего файла
- Максимальная глубина вложенности импортов -- 5 уровней

Это избегает дублирования и позволяет каждому пакету включать только релевантные стандарты.

5.3 Директория .claude/rules/

`.claude/rules/` -- альтернатива монолитному CLAUDE.md для организации правил по темам:

```
.claude/rules/
  testing.md          -- конвенции тестирования
  api-conventions.md  -- конвенции API
  deployment.md       -- правила деплоя
  react-patterns.md   -- паттерны React
```

Ключевая фича: YAML frontmatter с полем `paths` для условной загрузки:

```
---
paths: ["src/api/**/*"]
---

Для API-файлов используй async/await с явной обработкой ошибок.
Каждый endpoint должен возвращать стандартную обёртку ответа.
```

```
---
paths: ["**/*.test.tsx", "**/*.test.ts"]
---

Тесты должны использовать describe/it блоки.
Используй фабрики данных вместо хардкода.
Не мокай базу данных -- используй тестовую БД.
```

Как это работает:

- Правило загружается **только** когда Claude Code редактирует файл, совпадающий с паттерном `paths`
- Это экономит контекст и токены -- нерелевантные правила не загружаются
- Glob-паттерны позволяют применять конвенции по типу файла **независимо от расположения** -- идеально для тестов, разбросанных по всей кодовой базе

Когда использовать `.claude/rules/` с paths vs directory-level CLAUDE.md:

- `.claude/rules/` с paths -- когда конвенции применяются к файлам, разбросанным по многим директориям (тесты, миграции)
- Directory-level CLAUDE.md -- когда конвенции привязаны к конкретной директории и не нужны за её пределами

5.4 Пользовательские slash-команды и Skills

> **Примечание:** в текущей версии Claude Code пользовательские команды (`.claude/commands/`) объединены с навыками (`.claude/skills/`). Оба формата создают вызываемые через `/название` команды. Экзаменационное руководство ссылается на `.claude/commands/` -- этот формат по-прежнему поддерживается.

Slash-команды -- это переиспользуемые промпт-шаблоны, вызываемые через `/название`:

Формат через `.claude/commands/` (legacy, поддерживается):

```
.claude/commands/
  review.md        -- /review -- стандартный код-ревью
  test-gen.md      -- /test-gen -- генерация тестов
```

Формат через `.claude/skills/` (актуальный):

```
.claude/skills/
  review/SKILL.md  -- /review -- с frontmatter конфигурацией
  test-gen/SKILL.md
```

Проектные команды (`.claude/commands/` или `.claude/skills/`):

- Хранятся в VCS, доступны всем при клонировании репозитория
- Обеспечивают единообразие рабочих процессов в команде

Пользовательские команды (`~/.claude/commands/` или `~/.claude/skills/`):

- Личные команды, не передаваемые через VCS
- Для индивидуальных рабочих процессов

5.5 Навыки (Skills) -- .claude/skills/

Навыки -- расширенные команды с конфигурацией через SKILL.md frontmatter:

```yaml
---
context: fork
allowed-tools: ["Read", "Grep", "Glob"]
argument-hint: "Путь к директории для анализа"
---

Проанализируй структуру кода в указанной директории.
Выведи отчёт о зависимостях и архитектурных паттернах.
```

Параметры frontmatter:

Параметр	Описание
`context: fork`	Запускает навык в изолированном субагенте. Многословный вывод не загрязняет основную сессию
`allowed-tools`	Ограничивает доступные инструменты (безопасность -- навык не может удалять файлы если не разрешено)
`argument-hint`	Подсказка, запрашивающая параметр при вызове без аргументов

Когда навык vs CLAUDE.md:

- **Навык** -- on-demand вызов для конкретной задачи (ревью, анализ, генерация)

- **CLAUDE.md** -- always-loaded универсальные стандарты и конвенции

Персональные навыки (`~/.claude/skills/`):

- Создавайте персональные варианты с другими именами, чтобы не затрагивать коллег

5.6 Режим планирования (Plan Mode) vs прямое выполнение

Режим планирования:

- Модель **только** исследует и планирует, не внося изменений
- Использует Read, Grep, Glob для исследования кодовой базы
- Результат -- план реализации, одобряемый пользователем
- Безопасная разведка без побочных эффектов

Когда использовать режим планирования:

- Масштабные изменения (десятки файлов)
- Множество допустимых подходов (микросервисы: как нарезать границы?)
- Архитектурные решения (какой фреймворк? какая структура?)
- Незнакомая кодовая база (нужно понять перед изменением)
- Миграции библиотек, затрагивающие 45+ файлов

Когда использовать прямое выполнение:

- Одно-файловые исправления с ясным stack trace
- Добавление одной валидационной проверки
- Хорошо понятные, однозначные изменения

Комбинированный подход:

1. Режим планирования для исследования и проектирования
2. Одобрение плана пользователем
3. Прямое выполнение для реализации одобренного плана

Explore subagent -- специализированный субагент для исследования кодовой базы:

- Изолирует многословный вывод от основного контекста
- Возвращает только сводку
- Предотвращает исчерпание контекстного окна при многофазных задачах

5.7 Команда /compact

`/compact` -- встроенная команда для сжатия контекста:

- Суммирует предыдущую историю, освобождая контекстное окно
- Используется в длинных сессиях исследования, когда контекст заполняется многословным выводом инструментов
- Риск: теряются точные числовые значения, даты и конкретные детали при суммаризации

5.8 Команда /memory

`/memory` -- встроенная команда для управления памятью между сессиями:

- Открывает файл `CLAUDE.md` для редактирования, позволяя сохранять заметки, предпочтения и контекст
- Информация сохраняется между сессиями и автоматически загружается при запуске
- Полезна для хранения: проектных конвенций, предпочтений пользователя, часто используемых команд, контекста текущей работы
- Альтернатива повторному объяснению одних и тех же инструкций в каждой сессии

5.9 Claude Code CLI для CI/CD

Флаг `-p` (или `--print`):

```
claude -p "Analyze this pull request for security issues"
```

- Неинтерактивный режим: обрабатывает промпт, выводит в stdout, завершается
- Без ожидания пользовательского ввода
- **Единственный правильный способ** запуска в CI/CD пайплайнах

Структурированный вывод для CI:

```
claude -p "Review this PR" --output-format json --json-schema
'{"type":"object",...}'
```

- `--output-format json` -- вывод в формате JSON
- `--json-schema` -- валидация вывода по схеме
- Результат можно парсить для автоматического постинга inline PR-комментариев

Изоляция контекста сессии: Одна и та же сессия Claude, которая **генерировала** код, менее эффективна для его **ревью** (модель сохраняет контекст рассуждений и менее склонна оспаривать свои решения). Используйте **независимый экземпляр** для ревью.

Предотвращение дублирования комментариев: При повторном ревью после новых коммитов -- включайте предыдущие результаты ревью в контекст и инструктируйте Claude сообщать только о **новых или неисправленных** проблемах.

5.10 fork_session и управление сессиями

`--resume <session-name>` -- возобновление именованной сессии:

```
claude --resume investigation-auth-bug
```

- Продолжает предыдущий разговор с сохранённым контекстом
- Полезно для длительных расследований в несколько сессий
- Риск: если файлы изменились с прошлой сессии, результаты инструментов могут быть устаревшими

`fork_session` -- создание независимой ветки от общей базы:

```
Исследование кодовой базы
         |
    fork_session
    /           \
Подход A:      Подход B:
Redux          Context API
```

- Оба форка наследуют контекст до точки ветвления
- Далее развиваются независимо
- Полезно для сравнения подходов или тестирования стратегий

Когда начать новую сессию вместо возобновления:

- Результаты инструментов устарели (файлы изменились)
- Прошло много времени и контекст деградировал
- Лучше начать с "Вот краткая сводка того, что мы обнаружили: ..." чем возобновлять со старыми данными

Глава 6: Промпт-инженерия -- продвинутые техники

> Документация: Prompt Engineering | Anthropic Cookbook

6.1 Few-shot промптинг

Few-shot промптинг -- включение 2-4 примеров ввода/вывода в промпт для демонстрации ожидаемого поведения.

Почему few-shot эффективнее текстовых описаний:

- Текстовое описание "будь точнее" интерпретируется по-разному
- Пример однозначно показывает ожидаемый формат и логику решений
- Модель обобщает паттерн на новые случаи (не просто повторяет примеры)

Виды few-shot примеров и их применение:

1. **Примеры для неоднозначных сценариев:**

```
Запрос: "Мой заказ сломан"
Действие: Вызвать get_customer -> lookup_order -> проверить статус.
Обоснование: "сломан" может означать повреждённый товар. Нужно проверить детали заказа.

Запрос: "Дайте мне менеджера"
Действие: Немедленно вызвать escalate_to_human.
Обоснование: Клиент явно запрашивает человека. Не пытаться решить самостоятельно.
```

2. **Примеры для формата вывода:**

```
Пример находки:
{
  "location": "src/auth/login.ts:42",
  "issue": "SQL injection в параметре username",
  "severity": "critical",
  "suggested_fix": "Использовать параметризованный запрос"
}
```

3. **Примеры для разграничения допустимого и проблемного кода:**

```
// Допустимо (не помечать):
const items = data.filter(x => x.active);

// Проблема (помечать):
const items = data.filter(x => x.active == true); // Используй строгое сравнение ===
```

4. **Примеры для извлечения из разных форматов документов:**

```
Документ с inline цитатами:
"Как показано в исследовании (Smith, 2023), уровень составляет 42%."
-> {"value": "42%", "source": "Smith, 2023", "type": "inline_citation"}

Документ с библиографией:
"Уровень составляет 42%. [1]"
-> {"value": "42%", "source": "reference_1", "type": "bibliography"}
```

5. **Примеры для неформальных измерений (informal measurements):**

```
Текст: "примерно две горсти риса"
-> {"amount": "~100г", "original_text": "две горсти", "precision": "approximate"}

Текст: "щепотка соли"
-> {"amount": "~1г", "original_text": "щепотка", "precision": "approximate"}
```

Few-shot особенно эффективен для извлечения неформальных и нестандартных единиц измерения, где текстовые правила не могут охватить всё разнообразие выражений.

Правила нормализации формата в промптах: При использовании строгих JSON-схем для структурированного вывода добавляйте в промпт правила нормализации формата:

```
Нормализация:
- Даты: всегда ISO 8601 (YYYY-MM-DD), "вчера" -> вычислить абсолютную дату
- Валюта: числовое значение + код валюты, "пять баксов" -> {"amount": 5, "currency":
"USD"}
- Проценты: десятичная дробь, "половина" -> 0.5
```

Это предотвращает семантические ошибки, когда JSON-схема синтаксически валидна, но значения несогласованны.

6.2 Явные критерии vs расплывчатые инструкции

Плохо (расплывчато):

```
Проверь комментарии в коде на точность.
Будь консервативен, сообщай только о высокоуверенных находках.
```

Хорошо (явные критерии):

```
Помечай комментарий как проблемный ТОЛЬКО если:
1. Комментарий описывает поведение, которое ПРОТИВОРЕЧИТ фактическому поведению кода
2. Комментарий ссылается на несуществующую функцию или переменную
3. TODO/FIXME комментарий для бага, который уже исправлен в коде

НЕ помечай:
- Комментарии, которые просто устарели стилистически
- Комментарии с мелкими неточностями в формулировках
- Отсутствие комментариев (это другая категория)
```

Определение критериев серьёзности с примерами:

```
CRITICAL: Сбой в рантайме для пользователей
  Пример: NullPointerException при обработке платежа

HIGH: Уязвимость безопасности
  Пример: SQL injection, XSS, отсутствие проверки прав

MEDIUM: Логическая ошибка без немедленного эффекта
  Пример: Неправильная сортировка, off-by-one ошибка

LOW: Качество кода
  Пример: Дублирование, неоптимальный алгоритм для малых данных
```

6.3 Prompt Chaining (цепочки промптов)

Prompt chaining -- разбиение сложной задачи на последовательные фокусированные шаги:

```
Шаг 1: Анализ файла auth.ts (только локальные проблемы)
       -> Результат: список проблем в auth.ts

Шаг 2: Анализ файла database.ts (только локальные проблемы)
       -> Результат: список проблем в database.ts

Шаг 3: Интеграционный проход (кросс-файловые зависимости)
       -> Результат: проблемы на стыках модулей
```

Зачем это нужно:

- Избегает **размытия внимания (attention dilution)** -- когда модель получает слишком много файлов одновременно, она может пропустить баги в одних файлах, давая при этом поверхностные комментарии по другим
- Обеспечивает **стабильное качество** анализа для каждого файла
- Позволяет **отдельно** анализировать кросс-файловые взаимодействия

Когда использовать prompt chaining vs динамическую декомпозицию:

- **Prompt chaining** -- для предсказуемых, повторяющихся задач (ревью кода, миграция файлов)
- **Динамическая декомпозиция** -- для открытых исследований, где подзадачи становятся ясными только в процессе

6.4 Паттерн "интервью"

Перед реализацией решения Claude задаёт уточняющие вопросы разработчику:

```
Claude: "Перед реализацией кэширования для API, несколько вопросов:
1. Какая стратегия инвалидации кэша предпочтительна -- TTL или event-based?
2. Допустима ли устаревшая дата (stale) при недоступности кэша?
3. Нужно ли кэшировать на уровне отдельных пользователей или глобально?
4. Какой ожидаемый объём данных для кэширования?"
```

Когда это полезно:

- Незнакомый домен (финтех, медицина, правовые системы)
- Задача с неочевидными implications (кэш-стратегии, режимы отказа)
- Несколько допустимых подходов, выбор между которыми зависит от контекста

6.5 Валидация и retry-with-feedback

Когда извлечённые данные не прошли валидацию:

```
Шаг 1: Извлечение данных из документа
Шаг 2: Валидация (Pydantic, JSON Schema, бизнес-правила)
Шаг 3: Если ошибка -- повторный запрос с контекстом:
  - Оригинальный документ
  - Предыдущее (ошибочное) извлечение
  - Конкретная ошибка: "Поле 'total' = 150, но сумма line_items = 145. Проверьте
значения."
```

Когда retry будет эффективен:

- Ошибки формата (дата в неправильном формате)
- Ошибки структуры (поле помещено не туда)
- Арифметические расхождения (модель может перепроверить)

Когда retry НЕ поможет:

- Информация отсутствует в источнике (нет в документе -- повтор не добавит её)
- Внешний контекст (данные в другом документе, не переданном модели)

Pydantic как инструмент валидации: Pydantic — Python-библиотека для валидации данных на основе схем. В контексте экзамена важны:

- **Валидация структуры:** Pydantic проверяет типы полей, обязательность, допустимые значения (enum) на уровне кода после получения JSON от Claude
- **Семантическая валидация:** Пользовательские валидаторы проверяют бизнес-логику (сумма позиций = итого, дата начала < дата окончания)
- **Циклы валидации/повтора:** При ошибке Pydantic-валидации — формируем сообщение об ошибке и отправляем Claude повторный запрос с контекстом ошибки
- **Генерация JSON-схем:** Pydantic-модели могут автоматически генерировать JSON Schema для параметра `tool_use`, обеспечивая единый источник правды для схемы

6.6 Self-correction (самокоррекция)

Паттерн для обнаружения внутренних противоречий:

```
{
  "stated_total": "$150.00",
  "calculated_total": "$145.00",
  "conflict_detected": true,
  "line_items": [
    {"name": "Widget A", "price": 75.00},
    {"name": "Widget B", "price": 70.00}
  ]
}
```

Модель извлекает **и** указанное значение, **и** расчётное -- если они расходятся, флаг `conflict_detected` позволяет обработать расхождение.

Глава 7: Message Batches API

> Документация: Message Batches

7.1 Обзор

Message Batches API позволяет отправлять пакеты запросов для асинхронной обработки:

Характеристика	Значение
Экономия	**50%** от стоимости синхронных вызовов
Окно обработки	До **24 часов** (нет гарантий SLA по задержке)
Multi-turn tool calling	**Не поддерживается** (один запрос = один ответ)
Корреляция	Поле `custom_id` для связывания запроса и ответа

7.2 Когда использовать Batch API vs синхронный API

Задача	API	Причина
Pre-merge проверка PR	**Синхронный**	Разработчик ждёт результат; 24 часа неприемлемо
Ночной отчёт о техдолге	**Batch**	Результат нужен к утру; 50% экономия
Еженедельный аудит безопасности	**Batch**	Нет спешки; 50% экономия
Интерактивный код-ревью	**Синхронный**	Нужен немедленный ответ
Обработка 10,000 документов	**Batch**	Массовая обработка; экономия существенна

7.3 Работа с custom_id

```
{
  "custom_id": "doc-invoice-2024-001",
  "params": {
    "model": "claude-sonnet-4-6",
    "max_tokens": 1024,
    "messages": [{"role": "user", "content": "Извлеки данные из: ..."}]
  }
}
```

`custom_id` позволяет:

- Связать результат с исходным документом

- При сбое -- повторно отправить **только** неудавшиеся документы
- Не обрабатывать успешные документы повторно

7.4 Обработка сбоев в пакетах

1. Отправка пакета из 100 документов
2. 95 обработаны успешно, 5 -- сбой (context limit exceeded)
3. Определение неудавшихся по `custom_id`
4. Модификация (разбиение длинных документов на чанки)
5. Повторная отправка только 5 неудавшихся

7.5 Расчёт SLA

Если нужен результат через 30 часов, а Batch API обрабатывает до 24 часов:

- Окно отправки: 30 - 24 = **6 часов**
- Пакеты должны быть отправлены не позже чем за 24 часа до дедлайна
- При частых отправках -- разбить на окна по 4 часа

Глава 8: Стратегии декомпозиции задач

8.1 Фиксированные конвейеры (Prompt Chaining)

Каждый шаг определён заранее:

```
Документ -> Извлечение метаданных -> Извлечение данных -> Валидация -> Обогащение -> Финальный вывод
```

Когда использовать:

- Предсказуемая структура задачи (ревью всегда по одному шаблону)
- Все шаги известны заранее
- Нужна стабильность и воспроизводимость

8.2 Динамическая адаптивная декомпозиция

Подзадачи генерируются на основе промежуточных результатов:

```
1. "Добавь тесты для legacy-кодовой базы"
2. -> Сначала: картирование структуры (Glob, Grep)
3. -> Обнаружено: 3 модуля без тестов, 2 с частичным покрытием
4. -> Приоритизация: начать с модуля платежей (высокий риск)
5. -> В процессе: обнаружена зависимость от внешнего API
6. -> Адаптация: добавить mock для внешнего API перед написанием тестов
```

Когда использовать:

- Открытые исследовательские задачи
- Когда полный объём работы неизвестен заранее
- Когда каждый шаг зависит от результатов предыдущего

8.3 Мульти-проходное ревью кода

Для пулл-реквестов с 10+ файлами:

```
Проход 1 (per-file): Анализ auth.ts -> список локальных проблем
Проход 1 (per-file): Анализ database.ts -> список локальных проблем
Проход 1 (per-file): Анализ routes.ts -> список локальных проблем
...
Проход 2 (интеграция): Анализ связей между файлами
  -> Кросс-файловые проблемы: несогласованные типы, циклические зависимости
```

Почему один проход для 14 файлов -- плохо:

- Размытие внимания: детальный анализ одних файлов, поверхностный -- других
- Противоречивые комментарии: паттерн помечается как проблемный в одном файле, но одобряется в другом
- Пропуск багов: очевидные ошибки пропущены из-за когнитивной перегрузки

Глава 9: Эскалация и Human-in-the-Loop

9.1 Когда эскалировать к человеку

Триггеры эскалации (чёткие правила):

Ситуация	Действие
Клиент явно просит "дайте мне менеджера"	Немедленная эскалация, без попытки решить
Политика не покрывает запрос клиента	Эскалация (например, сопоставление цен конкурентов, а политика об этом молчит)
Агент не может добиться прогресса	Эскалация после разумного числа попыток

Финансовая операция выше порога	Эскалация (лучше через хук, чем через промпт)
Множественные совпадения при поиске клиента	Запрос дополнительных идентификаторов, а не угадывание

Что НЕ является надёжным триггером:

Ненадёжный метод	Почему не работает
Анализ настроений	Настроение клиента не коррелирует со сложностью случая
Самооценка уверенности модели (1-10)	Модель уже некорректно уверена в неправильных решениях; плохо калибрована
Автоматический классификатор	Переусложнение; требует обучающих данных, которых может не быть

9.2 Паттерны эскалации

Немедленная эскалация:

```
Клиент: "Я хочу поговорить с менеджером"
Агент: [немедленно вызывает escalate_to_human]
НЕ: "Я могу помочь с вашим вопросом, позвольте мне..."
```

Эскалация с попыткой решения:

```
Клиент: "Мой холодильник сломался через 2 дня после покупки"
Агент: [проверяет заказ, предлагает замену по гарантии]
Если клиент недоволен решением -> эскалация
```

Нюансированная эскалация (acknowledge → resolve → escalate on reiteration):

```
Клиент: "Это безобразие, я очень недоволен качеством!"
Агент: [признаёт разочарование] "Я понимаю ваше разочарование."
       [предлагает решение] "Могу предложить замену или возврат."
Клиент: "Нет, я хочу поговорить с кем-то!"
Агент: [клиент повторно настаивает -> немедленная эскалация]
```

Ключевой принцип: сначала признать эмоцию клиента, затем предложить конкретное решение, и только при повторном настаивании — эскалировать. Не эскалировать при первом выражении недовольства (это не то же, что запрос менеджера).

Эскалация при пробеле в политике:

```
Клиент: "У конкурента X этот товар на 30% дешевле, сделайте скидку"
Политика: описывает только ценовые корректировки для собственного сайта
Агент: [эскалирует -- политика не покрывает сопоставление цен конкурентов]
```

9.3 Структурированные протоколы передачи (Handoff)

При эскалации агент должен передать человеку структурированную сводку:

```
{
  "customer_id": "CUST-12345",
  "customer_name": "Иван Петров",
  "issue_summary": "Запрос возврата за повреждённый товар",
  "order_id": "ORD-67890",
  "root_cause": "Товар прибыл повреждённым, фото прикреплены",
  "actions_taken": [
    "Верифицирован клиент через get_customer",
    "Подтверждён заказ через lookup_order",
    "Предложена стандартная замена -- клиент настаивает на возврате"
  ],
  "refund_amount": "$89.99",
  "recommended_action": "Одобрить полный возврат",
  "escalation_reason": "Клиент запросил разговор с менеджером"
}
```

Человек-оператор **не имеет доступа к транскрипту разговора** -- он видит только эту сводку. Поэтому она должна быть полной и самодостаточной.

9.4 Калибровка уверенности и человеческий контроль

Для систем извлечения данных:

1. **Field-level confidence scores:** модель выдаёт оценку уверенности для каждого извлечённого поля
2. **Калибровка:** использование размеченных валидационных наборов для настройки порогов
3. **Маршрутизация:**
   - Высокая уверенность + стабильная точность -> автоматическая обработка
   - Низкая уверенность или неоднозначный источник -> человеческое ревью

Стратифицированная случайная выборка:

- Даже для высокоуверенных извлечений регулярно проверяйте выборку
- Агрегатная точность 97% может маскировать 40% ошибок на определённом типе документов
- Анализируйте точность по типу документа и по полю, а не только в целом

Глава 10: Обработка ошибок в мультиагентных системах

10.1 Категории ошибок

Категория	Примеры	Повторяемость	Действие агента
Transient	Таймаут, 503, сетевой сбой	Да	Повторить с экспоненциальным backoff
Validation	Неверный формат ввода, отсутствует обязательное поле	Нет (нужно исправить ввод)	Изменить запрос и повторить
Business	Нарушение политики, лимит превышен	Нет	Объяснить пользователю, предложить альтернативу
Permission	Нет прав доступа	Нет	Эскалировать

10.2 Антипаттерны обработки ошибок

Антипаттерн	Проблема	Правильный подход
Общий статус "search unavailable"	Координатор не может принять решение о восстановлении	Возвращать тип ошибки, запрос, частичные результаты, альтернативы
Молчаливое подавление (пустой результат = успех)	Координатор думает, что совпадений нет, а на самом деле был сбой	Явно различать "нет результатов" от "сбой при поиске"
Завершение всего процесса при одном сбое	Потеря всех частичных результатов	Продолжить с частичными результатами, аннотировать пробелы
Бесконечные повторы внутри субагента	Задержка и расход ресурсов	Локальное восстановление (1-2 повтора), затем пропагация координатору

10.3 Структурированная ошибка субагента

```
{
  "status": "partial_failure",
  "failure_type": "timeout",
  "attempted_query": "AI impact on music industry 2024",
  "partial_results": [
    {"title": "AI Music Generation Report", "url": "...", "relevance": 0.8}
  ],
  "alternative_approaches": [
    "Попробовать более узкий запрос: 'AI music composition tools'",
    "Использовать альтернативный источник данных"
  ],
  "coverage_impact": "Не покрыто: влияние AI на музыкальное производство"
}
```

Координатор получает всю информацию для решения:

- Повторить с модифицированным запросом?
- Использовать частичные результаты?
- Привлечь другой субагент?
- Продолжить без этого раздела и аннотировать пробел?

10.4 Аннотации покрытия в финальном синтезе

```
## Отчёт: Влияние AI на творческие индустрии

### Визуальное искусство (ПОЛНОЕ ПОКРЫТИЕ)
[результаты исследования]

### Музыка (ЧАСТИЧНОЕ ПОКРЫТИЕ -- таймаут при поиске)
[частичные результаты]
⚠️ Примечание: покрытие этого раздела ограничено из-за таймаута поискового агента.

### Литература (ПОЛНОЕ ПОКРЫТИЕ)
[результаты исследования]
```

Глава 11: Управление контекстом в production-системах

11.1 Извлечение фактов в отдельный блок

Вместо того чтобы полагаться на историю разговора (которая деградирует при суммаризации), извлекайте ключевые факты в структурированный блок:

```
=== CASE FACTS (обновляется при каждом новом факте) ===
Customer ID: CUST-12345
Order ID: ORD-67890
Order Date: 2025-01-15
Order Amount: $89.99
Issue: Повреждённый товар при доставке
Customer Request: Полный возврат средств
Status: Ожидает одобрения менеджера
===
```

Этот блок включается в **каждый** промпт, вне зависимости от суммаризации истории.

11.2 Тримминг результатов инструментов

Если `lookup_order` возвращает 40+ полей, а для текущей задачи нужны только 5:

```
# PostToolUse хук: оставить только релевантные поля
@hook("PostToolUse", tool="lookup_order")
def trim_order_fields(result):
    return {
        "order_id": result["order_id"],
        "status": result["status"],
        "total": result["total"],
        "items": result["items"],
        "return_eligible": result["return_eligible"]
    }
```

Это экономит контекстное окно и снижает шум.

11.3 Position-aware ввод

Размещайте критическую информацию с учётом "lost-in-the-middle" эффекта:

```
[КЛЮЧЕВЫЕ ВЫВОДЫ -- в начале]
Обнаружено 3 критические уязвимости...

[ДЕТАЛЬНЫЕ РЕЗУЛЬТАТЫ -- середина]
=== Файл auth.ts ===
...
=== Файл database.ts ===
...

[ИНСТРУКЦИИ К ДЕЙСТВИЮ -- в конце]
Приоритет: исправить уязвимости auth.ts перед мержем.
```

11.4 Scratchpad-файлы

При длительном исследовании кодовой базы агент может записывать ключевые находки в scratchpad-файл:

```
# investigation-scratchpad.md
## Ключевые находки
- Класс PaymentProcessor в src/payments/processor.ts наследует от BaseProcessor
- Метод refund() вызывается из 3 мест: OrderController, AdminPanel, CronJob
- Внешний API PaymentGateway имеет rate limit 100 req/min
- Миграция #47 добавила поле refund_reason (NOT NULL) -- 2024-12-01
```

При деградации контекста (или в новой сессии) агент обращается к scratchpad вместо повторного исследования.

11.5 Делегирование субагентам для защиты контекста

```
Главный агент: "Исследуй зависимости модуля платежей"
  -> Субагент (Explore): читает 15 файлов, трассирует импорты
  -> Возвращает: "Модуль платежей зависит от AuthService, OrderModel, и внешнего
PaymentGateway API"

Главный агент: сохраняет 1 строку вместо 15 файлов в контексте
```

Отдельный контекстный слой (separate context layer): В мультиагентных системах каждый субагент работает с ограниченным контекстным бюджетом — он получает только ту информацию, которая нужна для его задачи. Координатор выступает как отдельный контекстный слой: агрегирует результаты субагентов, хранит глобальное состояние и распределяет контекст. Это предотвращает «утечку контекста», когда один агент потребляет окно информацией, нерелевантной для других.

Ограниченные контекстные бюджеты субагентов:

- Передавайте субагенту минимальный контекст: конкретную задачу + необходимые данные
- Инструктируйте субагента возвращать только структурированный результат, а не сырые данные
- Используйте `allowedTools` для ограничения набора инструментов субагента — меньше инструментов = меньше отвлечений и затрат контекста

11.6 Структурированное сохранение состояния (для crash recovery)

Каждый агент экспортирует своё состояние в известное место:

```
// agent-state/web-search-agent.json
{
  "status": "completed",
  "queries_executed": ["AI music 2024", "AI music composition"],
  "results_count": 12,
  "key_findings": [...],
  "coverage": ["music composition", "music production"],
  "gaps": ["music distribution", "music licensing"]
}
```

Координатор загружает манифест при возобновлении:

```
// agent-state/manifest.json
{
  "web-search": "completed",
  "doc-analysis": "in_progress",
  "synthesis": "not_started"
}
```

Глава 12: Сохранение происхождения информации (Provenance)

12.1 Проблема потери атрибуции

При суммаризации результатов нескольких источников теряется связь "утверждение -> источник":

```
❌ Плохо: "Рынок AI в музыке оценивается в $3.2 млрд."
   (Откуда это число? Из какого источника? За какой год?)

✅ Хорошо:
{
  "claim": "Рынок AI в музыке оценивается в $3.2 млрд.",
  "source_url": "https://example.com/report",
  "source_name": "Global AI Music Report 2024",
  "publication_date": "2024-06-15",
  "confidence": 0.9
}
```

12.2 Обработка конфликтующих данных

Когда два источника приводят разные цифры:

```
{
  "claim": "Доля AI-сгенерированной музыки на стриминговых платформах",
  "values": [
    {
      "value": "12%",
      "source": "Spotify Annual Report 2024",
      "date": "2024-03",
      "methodology": "Автоматическая классификация"
    },
    {
      "value": "8%",
      "source": "Music Industry Association Survey",
      "date": "2024-07",
      "methodology": "Опрос 500 лейблов"
    }
  ],
  "conflict_detected": true,
  "possible_explanation": "Различие в методологии и временном периоде"
}
```

НЕ выбирайте произвольно одно значение. Сохраняйте оба с атрибуцией и позвольте координатору решить.

12.3 Включение дат для корректной интерпретации

Без дат временные различия интерпретируются как противоречия:

```
❌ "Источник A говорит 10%, источник B говорит 15%. Противоречие."
✅ "Источник A (2023) говорит 10%, источник B (2024) говорит 15%. Вероятен рост на 5% за год."
```

12.4 Рендеринг по типу контента

Не конвертируйте всё в единый формат:

- **Финансовые данные** -> таблицы
- **Новости и анализ** -> проза
- **Технические находки** -> структурированные списки
- **Временные ряды** -> хронологический порядок

Глава 13: Встроенные инструменты Claude

Code

13.1 Справочник выбора инструментов

Задача	Инструмент	Пример
Найти файлы по имени/паттерну	**Glob**	`**/*.test.tsx`, `src/components/**/*.ts`
Найти содержимое в файлах	**Grep**	Имя функции, сообщение об ошибке, import
Прочитать файл целиком	**Read**	Загрузка файла для анализа
Записать новый файл	**Write**	Создание нового файла с нуля
Точечная правка существующего файла	**Edit**	Замена конкретного фрагмента по уникальному тексту
Выполнить shell-команду	**Bash**	git, npm, запуск тестов, сборка

13.2 Стратегия инкрементального исследования

Не читайте все файлы сразу. Стройте понимание инкрементально:

```
1. Grep: найти точки входа (определение функции, export)
2. Read: прочитать найденные файлы
3. Grep: найти использования (import, вызовы)
4. Read: прочитать файлы-потребители
5. Повторять пока не построено полное понимание
```

13.3 Fallback: Read + Write вместо Edit

Когда Edit падает из-за неуникального текстового соответствия:

1. **Read** -- загрузить полное содержимое файла
2. Модифицировать содержимое программно
3. **Write** -- записать обновлённую версию

ЧАСТЬ II: КОНСПЕКТ ПО ДОМЕНАМ ЭКЗАМЕНА

Домен 1: Агентная архитектура и оркестрация (27%)

1.1 Проектирование агентных циклов для автономного выполнения задач

Ключевые знания:

- **Жизненный цикл агентного цикла**: отправка запроса Claude, проверка `stop_reason` (`"tool_use"` vs `"end_turn"`), выполнение инструментов, возврат результатов для следующей итерации
- Результаты инструментов добавляются в историю разговора, чтобы модель могла рассуждать о следующем действии
- **Модельное принятие решений** (Claude сам решает, какой инструмент вызвать) vs предустановленные деревья решений

Ключевые навыки:

- Реализация управления потоком: цикл продолжается при `stop_reason = "tool_use"` и завершается при `"end_turn"`
- Добавление результатов инструментов в контекст между итерациями
- **Антипаттерны, которых надо избегать**: парсинг текста ассистента для определения завершения, установка произвольных лимитов итераций как основного механизма остановки

1.2 Оркестрация мультиагентных систем (координатор-субагент)

Ключевые знания:

- **Hub-and-spoke архитектура**: координатор управляет всей межагентной коммуникацией, обработкой ошибок и маршрутизацией информации
- Субагенты работают с **изолированным контекстом** -- они НЕ наследуют историю разговора координатора автоматически
- Роль координатора: декомпозиция задач, делегирование, агрегация результатов, динамический выбор субагентов
- Риск слишком узкой декомпозиции задач координатором

Ключевые навыки:

- Разделение исследовательского охвата между субагентами для минимизации дублирования
- Реализация итеративных циклов уточнения (координатор оценивает синтез, перенаправляет задачи)
- Маршрутизация всей коммуникации через координатора для наблюдаемости

1.3 Конфигурация вызова субагентов, передачи контекста и порождения

Ключевые знания:

- Инструмент `Task` для порождения субагентов; `allowedTools` должен включать `"Task"` для координатора
- Контекст субагента **должен быть явно передан** в промпте -- субагенты не наследуют родительский контекст
- Конфигурация `AgentDefinition`: описания, системные промпты, ограничения инструментов
- Управление сессиями через `fork_session` для исследования альтернативных подходов

Ключевые навыки:

- Включение полных результатов от предыдущих агентов в промпт субагента
- Использование структурированных форматов для разделения данных и метаданных при передаче контекста
- Порождение параллельных субагентов через несколько вызовов `Task` в одном ответе координатора
- Проектирование промптов координатора с целями и критериями качества, а не пошаговыми инструкциями

1.4 Реализация многошаговых рабочих процессов с паттернами применения и передачи

Ключевые знания:

- Разница между **программным применением** (хуки, предусловия) и **промпт-руководством** для упорядочивания рабочего процесса
- Когда нужна детерминированная гарантия (проверка личности перед финансовыми операциями) -- одних промптов недостаточно
- Структурированные протоколы передачи при эскалации (ID клиента, причина, рекомендуемое действие)

Ключевые навыки:

- Программные предусловия, блокирующие нижестоящие вызовы до завершения предыдущих шагов (блокировка `process_refund` пока `get_customer` не вернул ID)
- Декомпозиция многоаспектных запросов клиентов на отдельные элементы
- Формирование структурированных сводок при эскалации к человеку

1.5 Хуки Agent SDK для перехвата вызовов инструментов и нормализации данных

Ключевые знания:

- Паттерны хуков (например, `PostToolUse`) для перехвата результатов инструментов перед обработкой моделью
- Хуки для перехвата исходящих вызовов и применения правил соответствия (блокировка возвратов выше порога)
- Хуки дают **детерминированные гарантии** vs промпт-инструкции дают **вероятностное соответствие**

Ключевые навыки:

- `PostToolUse` хуки для нормализации форматов данных (Unix timestamps, ISO 8601, числовые коды статусов)
- Хуки перехвата для блокировки действий, нарушающих политику, с перенаправлением на эскалацию
- Выбор хуков вместо промптов, когда бизнес-правила требуют гарантированного соответствия

1.6 Стратегии декомпозиции задач для сложных рабочих процессов

Ключевые знания:

- **Фиксированные конвейеры** (prompt chaining) vs **динамическая адаптивная декомпозиция** на основе промежуточных результатов
- Prompt chaining: последовательные шаги (анализ каждого файла по отдельности, затем интеграционный проход)
- Адаптивные планы расследования, генерирующие подзадачи на основе обнаруженного

Ключевые навыки:

- Prompt chaining для предсказуемых мультиаспектных ревью, динамическая декомпозиция для открытых исследований
- Разбиение больших ревью кода на per-file анализ + отдельный кросс-файловый интеграционный проход
- Декомпозиция открытых задач: сначала картирование структуры, затем приоритизированный план

1.7 Управление состоянием сессии, возобновление и форкинг

Ключевые знания:

- `--resume <session-name>` для продолжения именованных сессий

- `fork_session` для создания независимых веток исследования из общей базы
- Важность информирования агента об изменениях в файлах при возобновлении сессий
- Новая сессия со структурированной сводкой надёжнее, чем возобновление с устаревшими результатами

Ключевые навыки:

- Использование `--resume` для продолжения именованных сессий расследования
- `fork_session` для параллельного сравнения подходов
- Выбор между возобновлением (контекст ещё актуален) и новой сессией (результаты устарели)

Домен 2: Проектирование инструментов и интеграция MCP (18%)

2.1 Проектирование интерфейсов инструментов с чёткими описаниями

Ключевые знания:

- Описания инструментов -- **основной механизм**, по которому LLM выбирает инструмент; минимальные описания ведут к ненадёжному выбору
- Важность включения форматов ввода, примеров запросов, граничных случаев и границ применимости
- Неоднозначные или пересекающиеся описания вызывают ошибочную маршрутизацию
- Формулировки в системном промпте могут создавать непреднамеренные ассоциации с инструментами

Ключевые навыки:

- Написание описаний, чётко разграничивающих каждый инструмент от похожих альтернатив
- Переименование инструментов для устранения функционального пересечения (например, `analyze_content` -> `extract_web_results`)
- Разделение общих инструментов на специализированные с определёнными контрактами ввода/вывода

2.2 Реализация структурированных ответов об ошибках для MCP-инструментов

Ключевые знания:

- Флаг `isError` в MCP для сообщения об ошибках агенту
- Различие между **временными ошибками** (таймауты), **ошибками валидации** (неверный ввод), **бизнес-ошибками** (нарушение политики) и **ошибками доступа**

- Общие ответы об ошибках ("Operation failed") не позволяют агенту принимать правильные решения о восстановлении
- Различие между повторяемыми и неповторяемыми ошибками

Ключевые навыки:

- Возврат структурированных метаданных: `errorCategory` (transient/validation/permission), `isRetryable`, описание
- `retriable: false` для нарушений бизнес-правил с понятными пользователю объяснениями
- Локальное восстановление в субагентах при временных сбоях, пропагация только неразрешимых ошибок
- Различение сбоев доступа (нужно повторить) от валидных пустых результатов (совпадений нет)

2.3 Распределение инструментов между агентами и настройка tool_choice

Ключевые знания:

- Слишком много инструментов у агента (18 вместо 4-5) **снижает надёжность** выбора
- Агенты с инструментами вне их специализации склонны их использовать не по назначению
- Scoped tool access: только инструменты для своей роли + ограниченные кросс-ролевые для частых нужд
- `tool_choice`: `"auto"`, `"any"`, принудительный выбор (`{"type": "tool", "name": "..."}`)

Ключевые навыки:

- Ограничение набора инструментов каждого субагента релевантными для его роли
- Замена общих инструментов на ограниченные альтернативы (например, `fetch_url` -> `load_document`)
- `tool_choice: "any"` для гарантии вызова инструмента вместо текстового ответа
- Принудительный выбор конкретного инструмента для обеспечения порядка выполнения

2.4 Интеграция MCP-серверов в Claude Code и агентные рабочие процессы

Ключевые знания:

- **Область видимости MCP-серверов**: проектная (`.mcp.json`) для команды vs пользовательская (`~/.claude.json`) для экспериментов
- Подстановка переменных окружения в `.mcp.json` (например, `${GITHUB_TOKEN}`) для управления секретами
- Инструменты со всех подключённых MCP-серверов обнаруживаются при подключении и доступны одновременно
- MCP-ресурсы как каталоги контента (сводки задач, схемы баз данных) для сокращения исследовательских вызовов

Ключевые навыки:

- Настройка общих MCP-серверов в проектном `.mcp.json` с переменными окружения для токенов
- Личные/экспериментальные серверы в `~/.claude.json`
- Предпочтение существующих community MCP-серверов вместо собственных для стандартных интеграций

2.5 Выбор и применение встроенных инструментов (Read, Write, Edit, Bash, Grep, Glob)

Ключевые знания:

- **Grep** -- поиск содержимого файлов (имена функций, сообщения об ошибках, импорты)
- **Glob** -- поиск файлов по паттернам имён и расширений
- **Read/Write** -- полные файловые операции; **Edit** -- точечные изменения с уникальным текстовым соответствием
- Когда Edit падает из-за неуникального совпадения -- Read + Write как запасной вариант

Ключевые навыки:

- Grep для поиска по содержимому кодовой базы, Glob для нахождения файлов по паттернам
- Инкрементальное построение понимания: Grep для поиска точек входа, затем Read для трассировки потоков
- Трассировка использования функций через модули-обёртки

Домен 3: Конфигурация и рабочие процессы Claude Code (20%)

3.1 Настройка CLAUDE.md с иерархией, областями видимости и модульной организацией

Ключевые знания:

- **Иерархия CLAUDE.md**: пользовательский (`~/.claude/CLAUDE.md`), проектный (`.claude/CLAUDE.md` или корневой `CLAUDE.md`), директорный (файлы `CLAUDE.md` в подкаталогах)
- Пользовательские настройки применяются только к данному пользователю и не передаются через VCS
- Синтаксис `@path` для ссылок на внешние файлы (например, `@./standards/coding-style.md`), модульность CLAUDE.md
- Директория `.claude/rules/` для тематических файлов правил вместо монолитного CLAUDE.md

Ключевые навыки:

- Диагностика проблем иерархии (новый участник не получает инструкции -- они в user-level вместо project-level)
- `@path` (например, `@./standards/testing.md`) для выборочного включения стандартов в CLAUDE.md каждого пакета
- Разбиение больших CLAUDE.md на файлы в `.claude/rules/` (testing.md, api-conventions.md, deployment.md)

3.2 Создание и настройка пользовательских slash-команд и навыков

Ключевые знания:

- **Проектные команды** в `.claude/commands/` (через VCS) vs **пользовательские** в `~/.claude/commands/`
- Навыки в `.claude/skills/` с файлами `SKILL.md` с frontmatter: `context: fork`, `allowed-tools`, `argument-hint`
- `context: fork` запускает навык в изолированном субагенте, не загрязняя основную сессию
- Персональные варианты навыков в `~/.claude/skills/` с другими именами

Ключевые навыки:

- Проектные slash-команды в `.claude/commands/` для доступности всей команде
- `context: fork` для изоляции навыков с многословным выводом
- `allowed-tools` для ограничения доступа к инструментам при выполнении навыка
- `argument-hint` для запроса параметров у разработчика

3.3 Применение path-specific правил для условной загрузки конвенций

Ключевые знания:

- Файлы `.claude/rules/` с YAML frontmatter полем `paths` для условной активации правил по glob-паттернам
- Path-scoped правила загружаются **только при редактировании совпадающих файлов**, экономя контекст и токены
- Преимущество glob-паттернов перед directory-level CLAUDE.md для конвенций, охватывающих несколько каталогов (например, тестовые файлы)

Ключевые навыки:

- Создание `.claude/rules/` файлов с `paths: ["terraform/**/*"]` для загрузки только при работе с matching-файлами
- Glob-паттерны (`**/*.test.tsx`) для применения конвенций по типу файла независимо от расположения

- Выбор path-specific правил вместо directory CLAUDE.md, когда конвенции распространяются на файлы по всей кодовой базе

3.4 Определение, когда использовать режим планирования vs прямое выполнение

Ключевые знания:

- **Режим планирования** -- для сложных задач с масштабными изменениями, множеством допустимых подходов, архитектурными решениями
- **Прямое выполнение** -- для простых, хорошо понятных изменений (добавление одной валидации)
- Режим планирования позволяет безопасно исследовать кодовую базу до фиксации изменений
- Explore subagent для изоляции многословного вывода обнаружения

Ключевые навыки:

- Режим планирования для задач с архитектурными последствиями (микросервисы, миграции 45+ файлов)
- Прямое выполнение для исправлений с ясным stack trace и одним файлом
- Explore subagent для предотвращения исчерпания контекстного окна при многофазных задачах
- Комбинация: планирование для исследования + прямое выполнение для реализации

3.5 Итеративное уточнение для прогрессивного улучшения

Ключевые знания:

- Конкретные примеры ввода/вывода -- самый эффективный способ коммуникации ожиданий
- **Test-driven итерация**: сначала тесты, затем итерация по результатам ошибок
- **Паттерн "интервью"**: Claude задаёт вопросы, чтобы выявить неочевидные соображения перед реализацией
- Когда предоставлять все проблемы одним сообщением (взаимосвязанные) vs последовательно (независимые)

Ключевые навыки:

- 2-3 конкретных примера ввода/вывода для уточнения требований трансформации
- Тестовые наборы с ожидаемым поведением, граничными случаями и производительностью ДО реализации
- Паттерн интервью для выявления аспектов дизайна (стратегии инвалидации кэша, режимы отказа)
- Конкретные тест-кейсы с примерами ввода и ожидаемого вывода для граничных случаев

3.6 Интеграция Claude Code в CI/CD пайплайны

Ключевые знания:

- Флаг `-p` (или `--print`) для неинтерактивного режима в автоматизированных пайплайнах
- `--output-format json` и `--json-schema` для структурированного вывода в CI
- CLAUDE.md как механизм предоставления проектного контекста (стандарты тестирования, критерии ревью) для CI-вызванного Claude Code
- **Изоляция контекста сессии**: та же сессия, что генерировала код, менее эффективна для ревью по сравнению с независимым экземпляром

Ключевые навыки:

- Запуск Claude Code в CI с флагом `-p` для предотвращения зависания из-за интерактивного ввода
- `--output-format json` с `--json-schema` для структурированных результатов (inline PR-комментарии)
- Включение предыдущих результатов ревью при перезапуске после новых коммитов (для отчёта только о новых проблемах)
- Документирование стандартов тестирования и доступных фикстур в CLAUDE.md для повышения качества генерируемых тестов
- Включение существующих тестовых файлов в контекст при генерации новых тестов — предотвращает дублирование и обеспечивает консистентный стиль

Домен 4: Промпт-инженерия и структурированный вывод (20%)

4.1 Проектирование промптов с явными критериями для повышения точности

Ключевые знания:

- Явные критерии важнее расплывчатых инструкций (например, "помечать комментарии только при расхождении с кодом" vs "проверять точность комментариев")
- Общие инструкции типа "будь консервативнее" работают хуже конкретных категориальных критериев
- Влияние ложных срабатываний на доверие разработчиков: высокий уровень ложных срабатываний в одних категориях подрывает доверие к точным категориям

Ключевые навыки:

- Конкретные критерии ревью: какие проблемы сообщать (баги, безопасность) vs пропускать (мелкий стиль)
- Временное отключение категорий с высоким уровнем ложных срабатываний

- Определение явных критериев серьёзности с примерами кода для каждого уровня

4.2 Применение few-shot промптинга для улучшения консистентности вывода

Ключевые знания:

- Few-shot примеры -- **самый эффективный** метод для стабильно форматированного, действенного вывода
- Роль few-shot в демонстрации обработки неоднозначных случаев (выбор инструмента, пробелы в тестовом покрытии)
- Few-shot позволяют модели обобщать на новые паттерны, а не только повторять предустановленные
- Эффективность few-shot для снижения галлюцинаций в задачах извлечения

Ключевые навыки:

- 2-4 целевых примера для неоднозначных сценариев с обоснованием выбора
- Few-shot, демонстрирующие формат вывода (местоположение, проблема, серьёзность, предложенное исправление)
- Примеры, отличающие допустимые паттерны кода от реальных проблем
- Примеры корректного извлечения из документов с разной структурой

4.3 Обеспечение структурированного вывода через tool_use и JSON-схемы

Ключевые знания:

- `tool_use` с JSON-схемами -- **самый надёжный** подход для гарантированного вывода, соответствующего схеме; устраняет синтаксические ошибки JSON
- `tool_choice: "auto"` -- модель может вернуть текст; `"any"` -- обязана вызвать инструмент; принудительный выбор -- конкретный инструмент
- Строгие JSON-схемы устраняют синтаксические ошибки, но **не предотвращают семантические** (итоги не сходятся, значения в неправильных полях)
- Дизайн схемы: required vs optional поля, enum с "other" + detail string для расширяемых категорий

Ключевые навыки:

- Определение инструментов извлечения с JSON-схемами и извлечение данных из ответа `tool_use`
- `tool_choice: "any"` для гарантии структурированного вывода при множественных схемах
- Принудительный вызов конкретного инструмента: `tool_choice: {"type": "tool", "name": "extract_metadata"}`
- Дизайн полей схемы как optional (nullable), когда источник может не содержать информацию -- предотвращение фабрикации значений

- Значения enum `"unclear"` и `"other"` + detail поля для расширяемой категоризации

4.4 Реализация валидации, повторных попыток и обратной связи для качества извлечения

Ключевые знания:

- **Retry-with-error-feedback**: добавление конкретных ошибок валидации к промпту при повторе для направления модели к исправлению
- Ограничения повторов: неэффективны, когда информация просто отсутствует в источнике
- Дизайн петли обратной связи: отслеживание паттерна кода, вызвавшего находку (`detected_pattern`)
- Различие между **семантическими ошибками** (значения не сходятся) и **синтаксическими** (устраняются tool_use)

Ключевые навыки:

- Follow-up запросы с оригинальным документом, ошибочным извлечением и конкретными ошибками валидации
- Определение, когда повтор будет неэффективен (информация только во внешнем документе)
- Поля `detected_pattern` в находках для анализа ложных срабатываний
- Дизайн самокоррекции: извлечение "calculated_total" и "stated_total" для выявления расхождений

4.5 Проектирование стратегий эффективной пакетной обработки

Ключевые знания:

- **Message Batches API**: экономия 50%, окно обработки до 24 часов, без гарантий SLA по задержке
- Пакетная обработка подходит для **неблокирующих** задач (ночные отчёты, аудиты) и НЕ подходит для блокирующих (pre-merge проверки)
- Batch API **не поддерживает** multi-turn tool calling в рамках одного запроса
- Поля `custom_id` для корреляции запрос/ответ в пакетах

Ключевые навыки:

- Синхронный API для блокирующих проверок, Batch API для ночных/еженедельных задач
- Расчёт частоты отправки пакетов с учётом SLA (окна по 4 часа для 30-часовой гарантии с 24-часовой пакетной обработкой)
- Обработка сбоев: повторная отправка только неудавшихся документов (идентификация по `custom_id`)
- Уточнение промпта на выборке перед массовой обработкой

4.6 Проектирование архитектур мульти-инстансного и мульти-проходного ревью

Ключевые знания:

- **Ограничения саморевью**: модель сохраняет контекст рассуждений, менее склонна оспаривать собственные решения
- Независимые инстансы ревью (без предшествующего контекста) эффективнее для нахождения тонких проблем
- **Мульти-проходное ревью**: per-file локальный анализ + кросс-файловый интеграционный проход для избежания размытия внимания

Ключевые навыки:

- Использование второго независимого инстанса Claude для ревью без контекста генерации
- Разбиение мульти-файловых ревью на фокусированные per-file проходы + интеграционные проходы для кросс-файлового анализа потоков данных
- Проходы верификации с самооценкой уверенности для калиброванной маршрутизации ревью

Домен 5: Управление контекстом и надёжность (15%)

5.1 Управление контекстом разговора для сохранения критической информации

Ключевые знания:

- **Риски прогрессивной суммаризации**: конденсация числовых значений, процентов, дат в расплывчатые резюме
- Эффект **"потеря в середине"**: модели надёжно обрабатывают начало и конец длинных вводов, но могут пропускать находки из середины
- Результаты инструментов накапливаются в контексте непропорционально их релевантности (40+ полей при 5 нужных)
- Важность передачи полной истории разговора в последующих API-запросах

Ключевые навыки:

- Извлечение транзакционных фактов в постоянный блок "case facts" за пределами суммаризированной истории
- Тримминг многословных выводов инструментов до релевантных полей
- Размещение ключевых выводов в начале агрегированных данных с явными заголовками секций
- Требование к субагентам включать метаданные (даты, источники) в структурированные выводы

5.2 Проектирование эффективных паттернов эскалации и разрешения неоднозначностей

Ключевые знания:

- Подходящие триггеры эскалации: запрос клиента на человека, пробелы/исключения в политике, невозможность прогресса
- Различие между немедленной эскалацией (клиент явно требует) и попыткой решения (проблема в компетенции агента)
- Анализ настроений и самооценка уверенности -- **ненадёжные** прокси для сложности случая
- Множественные совпадения клиентов требуют запроса дополнительных идентификаторов, а не эвристического выбора

Ключевые навыки:

- Явные критерии эскалации с few-shot примерами в системном промпте
- Немедленное выполнение явных запросов клиента на человека без предварительного расследования
- Эскалация при неоднозначности или молчании политики относительно конкретного запроса
- Запрос дополнительных идентификаторов при множественных совпадениях в результатах

5.3 Реализация стратегий распространения ошибок в мультиагентных системах

Ключевые знания:

- Структурированный контекст ошибки (тип сбоя, запрос, частичные результаты, альтернативы) для умного восстановления координатором
- Различие между сбоями доступа (таймауты -- нужно решение о повторе) и валидными пустыми результатами (нет совпадений)
- Общие статусы ошибок ("search unavailable") скрывают ценный контекст от координатора
- Молчаливое подавление ошибок или завершение всего рабочего процесса при единичном сбое -- оба являются антипаттернами

Ключевые навыки:

- Возврат структурированного контекста ошибки: тип сбоя, что было предпринято, частичные результаты, возможные альтернативы
- Различение сбоев доступа от валидных пустых результатов
- Локальное восстановление субагентов при временных сбоях, пропагация только неразрешимых с частичными результатами
- Аннотации покрытия в синтезе: какие находки подтверждены vs где есть пробелы

5.4 Эффективное управление контекстом при исследовании больших кодовых баз

Ключевые знания:

- **Деградация контекста** в длинных сессиях: модель начинает давать нестабильные ответы, ссылаясь на "типичные паттерны" вместо конкретных классов
- Роль scratchpad-файлов для сохранения ключевых находок через границы контекста
- Делегирование субагентам для изоляции многословного исследовательского вывода
- Структурированное сохранение состояния для восстановления после сбоев

Ключевые навыки:

- Порождение субагентов для конкретных вопросов при сохранении высокоуровневой координации в основном агенте
- Scratchpad-файлы с ключевыми находками, ссылки на них при последующих вопросах
- Суммирование ключевых находок перед порождением субагентов следующей фазы
- Использование `/compact` для снижения использования контекста при длинных сессиях исследования

5.5 Проектирование рабочих процессов с человеческим контролем и калибровкой уверенности

Ключевые знания:

- Риск того, что агрегатные метрики (97% общей точности) маскируют низкую производительность на определённых типах документов или полях
- **Стратифицированная случайная выборка** для измерения процента ошибок в высокоуверенных извлечениях
- Калибровка оценок уверенности на уровне полей с использованием размеченных валидационных наборов
- Валидация точности по типу документа и сегменту поля перед автоматизацией

Ключевые навыки:

- Реализация стратифицированной случайной выборки для обнаружения новых паттернов ошибок
- Анализ точности по типу документа и полю для проверки стабильной производительности
- Вывод оценок уверенности на уровне полей, калибровка порогов ревью на размеченных данных
- Маршрутизация извлечений с низкой уверенностью или неоднозначными источниками на человеческое ревью

5.6 Сохранение происхождения информации и обработка неопределённости в мультиисточниковом синтезе

Ключевые знания:

- Атрибуция источников теряется при суммаризации без сохранения маппингов "утверждение-источник"
- Важность структурированных маппингов, которые агент синтеза должен сохранять при объединении
- Обработка конфликтующих статистик: аннотирование конфликтов с атрибуцией источников вместо произвольного выбора
- Включение дат публикации/сбора данных для предотвращения ошибочной интерпретации временных различий как противоречий

Ключевые навыки:

- Требование к субагентам выводить маппинги "утверждение-источник" (URL, название документа, цитаты)
- Структурирование отчётов: разделение на устоявшиеся и спорные находки
- Сохранение конфликтующих значений с аннотациями и передача координатору для согласования
- Включение дат публикации для корректной временной интерпретации
- Рендеринг контента по типу: финансовые данные -- таблицы, новости -- проза, технические находки -- структурированные списки

Примеры экзаменационных вопросов с разбором

Вопрос 1 (Сценарий: Агент поддержки клиентов)

Ситуация: Данные показывают, что в 12% случаев агент пропускает `get_customer` и вызывает `lookup_order` только по имени клиента, что приводит к ошибочным возвратам.

Какое изменение наиболее эффективно?

- A) Добавить программное предусловие, блокирующее `lookup_order` и `process_refund` до получения ID от `get_customer` **[ВЕРНО]**
- B) Улучшить системный промпт
- C) Добавить few-shot примеры
- D) Реализовать классификатор маршрутизации

Почему A: Когда критическая бизнес-логика требует определённой последовательности инструментов, программное обеспечение даёт **детерминированные гарантии**, которые промпт-подходы (B, C) обеспечить не могут. D решает проблему доступности, а не порядка инструментов.

Вопрос 2 (Сценарий: Агент поддержки клиентов)

Ситуация: Агент часто вызывает `get_customer` вместо `lookup_order` при вопросах о заказах. Описания инструментов минимальны и похожи.

Какой первый шаг?

- A) Few-shot примеры
- B) Расширить описания каждого инструмента с форматами ввода, примерами и границами **[ВЕРНО]**
- C) Слой маршрутизации
- D) Объединить инструменты

Почему B: Описания инструментов -- основной механизм выбора для LLM. Это наименьшее по усилиям, наиболее эффективное решение. A добавляет токены без исправления корня проблемы. C -- переусложнение. D требует больше усилий, чем оправдано.

Вопрос 3 (Сценарий: Агент поддержки клиентов)

Ситуация: Агент решает только 55% вопросов при цели 80%. Эскалирует простые случаи и пытается сам решать сложные исключения из политики.

Как улучшить калибровку?

- A) Добавить явные критерии эскалации с few-shot примерами **[ВЕРНО]**
- B) Самооценка уверенности (1-10) с автоматической эскалацией
- C) Отдельный классификатор на исторических данных
- D) Анализ настроений

Почему A: Прямо адресует корневую причину -- нечёткие границы решений. B ненадёжен (модель уже некорректно уверена). C -- переусложнение. D решает другую проблему (настроение != сложность).

Вопрос 4 (Сценарий: Генерация кода с Claude Code)

Ситуация: Нужна пользовательская команда `/review` для стандартного ревью кода, доступная всей команде при клонировании репозитория.

Где создать файл команды?

- A) `.claude/commands/` в репозитории проекта **[ВЕРНО]**
- B) `~/.claude/commands/`
- C) Корневой `CLAUDE.md`
- D) `.claude/config.json`

Почему A: Проектные команды в `.claude/commands/` управляются через VCS и автоматически доступны всем. B -- для личных команд. C -- для инструкций, не определений команд. D не существует.

Вопрос 5 (Сценарий: Генерация кода с Claude Code)

Ситуация: Нужно реструктурировать монолит в микросервисы (десятки файлов, решения о границах сервисов).

Какой подход?

- A) Режим планирования: исследовать кодовую базу, понять зависимости, спроектировать подход **[ВЕРНО]**
- B) Прямое выполнение инкрементально
- C) Прямое выполнение с подробными инструкциями заранее
- D) Прямое выполнение, переключение на планирование при сложностях

Почему A: Именно для этого предназначен режим планирования: масштабные изменения, множество подходов, архитектурные решения. B рискует дорогой переработкой. C предполагает знание структуры заранее. D -- реактивный подход.

Вопрос 6 (Сценарий: Генерация кода с Claude Code)

Ситуация: Кодовая база с разными конвенциями по областям (React, API, базы данных). Тесты рядом с кодом. Нужно автоматическое применение конвенций.

Какой подход?

- A) Файлы правил в `.claude/rules/` с YAML frontmatter и glob-паттернами **[ВЕРНО]**
- B) Всё в корневом CLAUDE.md
- C) Навыки в `.claude/skills/`
- D) CLAUDE.md в каждой директории

Почему A: `.claude/rules/` с glob-паттернами (например, `**/*.test.tsx`) позволяет автоматически применять конвенции по путям файлов -- идеально для тестов, разбросанных по кодовой базе. B полагается на вывод модели. C требует ручного вызова. D не работает для файлов в разных директориях.

Вопрос 7 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Система исследует "влияние ИИ на творческие индустрии", но отчёты покрывают только визуальное искусство. Координатор разбил тему на: "ИИ в цифровом арте", "ИИ в графическом дизайне", "ИИ в фотографии".

Причина?

- A) Агент синтеза не обнаруживает пробелы
- B) Координатор слишком узко декомпозировал задачу **[ВЕРНО]**
- C) Поисковый агент недостаточно полно ищет
- D) Агент анализа документов фильтрует невизуальные источники

Почему B: Логи прямо показывают: координатор разбил "творческие индустрии" только на визуальные подтемы, полностью пропустив музыку, литературу и кино. Субагенты корректно выполнили свои задания -- проблема в том, **что** им поручили.

Вопрос 8 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Субагент веб-поиска уходит в таймаут при исследовании сложной темы. Нужно спроектировать передачу информации об ошибке координатору.

Какой подход к распространению ошибок лучше всего обеспечит умное восстановление?

- A) Вернуть координатору структурированный контекст ошибки: тип сбоя, запрос, частичные результаты и возможные альтернативы **[ВЕРНО]**
- B) Реализовать автоматический retry с exponential backoff внутри субагента, возвращая общий статус "search unavailable" после исчерпания попыток
- C) Перехватить таймаут внутри субагента и вернуть пустой набор результатов, помеченный как успешный
- D) Пробросить исключение таймаута наверх к обработчику, завершающему весь рабочий процесс

Почему A: Структурированный контекст ошибки даёт координатору информацию для принятия решения: повторить с модифицированным запросом, попробовать альтернативный подход или продолжить с частичными результатами. B скрывает контекст за общим статусом. C маскирует сбой как успех. D завершает весь процесс без необходимости.

Вопрос 9 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Агент синтеза часто нуждается в проверке конкретных утверждений при объединении результатов. Сейчас он возвращает управление координатору, тот вызывает поисковый агент, затем снова запускает синтез. Это добавляет 2-3 round-trip и увеличивает задержку на 40%. 85% проверок -- простые факт-чеки (даты, имена, статистика), 15% требуют глубокого расследования.

Как снизить накладные расходы, сохраняя надёжность?

- A) Дать агенту синтеза ограниченный инструмент `verify_fact` для простых проверок, а сложные продолжать направлять через координатора **[ВЕРНО]**

- B) Накопить все запросы на проверку в пакет и отправить координатору разом
- C) Дать агенту синтеза полный доступ ко всем инструментам веб-поиска
- D) Заранее кэшировать расширенный контекст вокруг каждого источника

Почему A: Применяет принцип наименьших привилегий -- агент синтеза получает ровно то, что нужно для 85% типичных случаев (простой факт-чек), сохраняя координационный паттерн для сложных. B создаёт блокирующие зависимости (шаги синтеза могут зависеть от ранее проверенных фактов). C нарушает разделение ответственности. D полагается на спекулятивное кэширование, которое не может надёжно предсказать потребности.

Вопрос 10 (Сценарий: Claude Code для CI)

Ситуация: Пайплайн запускает `claude "Analyze this pull request for security issues"`, но зависает, ожидая интерактивный ввод.

Правильный подход?

- A) Флаг `-p`: `claude -p "Analyze this pull request for security issues"` **[ВЕРНО]**
- B) Переменная окружения `CLAUDE_HEADLESS=true`
- C) Перенаправление stdin из `/dev/null`
- D) Флаг `--batch`

Почему A: `-p` (или `--print`) -- документированный способ запуска в неинтерактивном режиме. Обрабатывает промпт, выводит в stdout и завершается. Остальные варианты -- несуществующие функции или обходные решения Unix.

Вопрос 11 (Сценарий: Claude Code для CI)

Ситуация: Команда хочет сократить затраты на API для автоматизированного анализа. Сейчас Claude в реальном времени обслуживает два рабочих процесса: (1) блокирующую проверку перед мержем, которая должна завершиться до того, как разработчики смогут смержить PR, и (2) отчёт о техническом долге, генерируемый ночью для утреннего ревью. Менеджер предлагает перевести оба на Message Batches API ради экономии 50%.

Как оценить это предложение?

- A) Использовать пакетную обработку только для отчётов о техдолге; оставить реалтайм-вызовы для pre-merge проверок **[ВЕРНО]**
- B) Перевести оба процесса на пакетную обработку с поллингом статуса для проверки завершения
- C) Оставить реалтайм-вызовы для обоих, чтобы избежать проблем с упорядочиванием результатов
- D) Перевести оба на пакетную обработку с фолбэком на реалтайм, если пакет обрабатывается слишком долго

Почему A: Message Batches API экономит 50%, но время обработки — до 24 часов без гарантированного SLA по задержке. Это делает его непригодным для блокирующих pre-merge проверок, где разработчики ждут результата, но идеальным для ночных пакетных задач вроде отчётов о техдолге. Вариант B неверен, потому что "часто быстрее" не гарантировано для блокирующих процессов. Вариант C — упущенная экономия. Вариант D добавляет ненужную сложность, когда простое решение — использовать каждый API для его целевого сценария.

Вопрос 12 (Сценарий: Мультифайловое ревью кода)

Ситуация: Pull request изменяет 14 файлов в модуле отслеживания запасов. Однопроходное ревью всех файлов даёт несогласованные результаты: детальные комментарии для одних файлов, но поверхностные для других, пропущенные баги и противоречивая обратная связь — паттерн отмечается как проблемный в одном файле, но одобряется в идентичном коде в другом файле PR.

Как реструктурировать ревью?

- A) Разделить на фокусированные проходы: анализ каждого файла по отдельности на локальные проблемы, затем отдельный интеграционный проход для межфайловых потоков данных **[ВЕРНО]**
- B) Требовать от разработчиков разбивать крупные PR на подачи по 3-4 файла
- C) Переключиться на модель более высокого уровня с большим контекстным окном для полноценного внимания ко всем 14 файлам за один проход
- D) Запустить три независимых прохода по полному PR и отмечать только те проблемы, которые найдены минимум в двух из трёх прогонов

Почему A: Разделение на фокусированные проходы напрямую устраняет корневую причину — размывание внимания при обработке многих файлов одновременно. Пофайловый анализ обеспечивает стабильную глубину, а отдельный интеграционный проход ловит межфайловые проблемы. Вариант B перекладывает бремя на разработчиков без улучшения системы. Вариант C — заблуждение: увеличение контекстного окна не решает проблем с качеством внимания. Вариант D фактически подавляет обнаружение реальных багов, требуя консенсуса по проблемам, которые могут обнаруживаться непоследовательно.

Пробный экзамен (Practice Test)

> 60 вопросов по 4 сценариям. Формат и сложность соответствуют реальному экзамену.
>
> Как вариант, вы можете пройти эти вопросы в формате интерактивного экзамена: Практический тест (RU)

Сценарий: Мультиагентная исследовательская система

Вопрос 1 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Агент анализа документов обнаруживает, что два достоверных источника содержат прямо противоречащую статистику по ключевой метрике: правительственный отчёт указывает рост 40%, а отраслевой анализ — 12%. Оба источника выглядят достоверными, и расхождение может существенно повлиять на выводы исследования. Как агенту анализа документов наиболее эффективно обработать эту ситуацию?

Какой подход наиболее эффективен?

- A) Применить эвристики достоверности источников для выбора наиболее вероятно точной цифры, завершить анализ с использованием этого значения и добавить сноску с упоминанием расхождения.
- B) Включить обе цифры в результат анализа, не отмечая их как противоречащие, позволив агенту синтеза определить, какую использовать на основе более широкого контекста исследования.
- C) Остановить анализ и немедленно эскалировать координатору, попросив его определить, какой источник авторитетнее, прежде чем агент продолжит обработку оставшихся документов.
- D) Завершить анализ документов с обеими цифрами, явно аннотировать конфликт с атрибуцией источников и позволить координатору решить, как согласовать данные, перед передачей на синтез. **[ВЕРНО]**

Почему D: Это наиболее эффективный подход, поскольку он соблюдает разделение ответственности: агент анализа документов завершает свою основную задачу без блокировки, сохраняет оба противоречивых значения с явной атрибуцией источников и корректно передаёт решение о согласовании координатору, у которого есть более широкий контекст для разрешения конфликта.

Вопрос 2 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Агенты веб-поиска и анализа документов завершили свои задачи и вернули результаты координатору. Каков следующий шаг для создания интегрированного исследовательского отчёта?

Какой следующий шаг наиболее уместен?

- A) Каждый агент отправляет свои результаты напрямую агенту генерации отчётов, минуя координатора.
- B) Агент анализа документов запрашивает результаты веб-поиска и объединяет их внутренне.
- C) Координатор передаёт оба набора результатов агенту синтеза для единой интеграции. **[ВЕРНО]**
- D) Координатор объединяет необработанные выходные данные обоих агентов и возвращает их как итоговый результат.

Почему C: Координатор передаёт оба набора результатов агенту синтеза для единой интеграции — это корректный шаг в архитектуре «координатор — субагенты», обеспечивающий

централизованный контроль и качественное объединение данных.

Вопрос 3 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Субагент анализа документов часто сталкивается со сбоями при обработке PDF-файлов — некоторые содержат повреждённые секции, вызывающие исключения парсинга, другие защищены паролем, а иногда библиотека парсинга зависает на больших файлах. В настоящее время любое исключение немедленно завершает работу субагента и возвращает ошибку координатору, который должен решить — повторить попытку, пропустить документ или признать всю задачу проваленной. Это вызывает чрезмерную вовлечённость координатора в рутинную обработку ошибок. Какое архитектурное улучшение наиболее эффективно?

Какое улучшение наиболее эффективно?

- A) Создать выделенного агента обработки ошибок, который отслеживает все сбои субагентов через общую очередь и принимает решения о восстановлении самостоятельно, отправляя команды повторного запуска непосредственно субагентам.
- B) Настроить субагента всегда возвращать частичные результаты со статусом успеха, встраивая детали ошибок в метаданные. Координатор обрабатывает все ответы как успешные и фильтрует проблемные элементы при синтезе.
- C) Поручить координатору валидировать все документы перед отправкой субагенту, отклоняя документы, которые могут вызвать сбои, чтобы субагент получал только обрабатываемые файлы.
- D) Реализовать в субагенте локальное восстановление для транзитных сбоев и передавать координатору только те ошибки, которые он не может разрешить самостоятельно, включая информацию о предпринятых попытках и полученных частичных результатах. **[ВЕРНО]**

Почему D: Реализация локального восстановления для транзитных сбоев внутри субагента следует принципу обработки ошибок на самом низком уровне, способном их разрешить. Это снижает чрезмерную вовлечённость координатора, при этом действительно неразрешимые проблемы эскалируются с полным контекстом, включая информацию о предпринятых попытках восстановления и полученных частичных результатах.

Вопрос 4 (Сценарий: Мультиагентная исследовательская система)

Ситуация: После запуска системы по теме «влияние ИИ на креативные индустрии» вы наблюдаете, что каждый субагент завершает работу успешно: агент веб-поиска находит релевантные статьи, агент анализа документов корректно резюмирует статьи, а агент синтеза создаёт связный текст. Однако итоговые отчёты охватывают только визуальное искусство, полностью упуская музыку, литературу и кинопроизводство. При изучении логов координатора вы видите, что он декомпозировал тему на три подзадачи: «ИИ в создании цифрового искусства», «ИИ в графическом дизайне» и «ИИ в фотографии». Какова наиболее вероятная первопричина?

Какова наиболее вероятная первопричина?

- A) Агент синтеза не содержит инструкций для выявления пробелов в покрытии полученных данных.
- B) Агент анализа документов отфильтровывает источники, связанные с невизуальными креативными индустриями, из-за чрезмерно строгих критериев релевантности.
- C) Декомпозиция задач координатором слишком узкая, что приводит к назначениям субагентам, не охватывающим все релевантные области темы. **[ВЕРНО]**
- D) Запросы агента веб-поиска недостаточно полны и должны быть расширены для охвата большего количества секторов креативных индустрий.

Почему C: Логи координатора прямо показывают, что он декомпозировал широкую тему только на три подзадачи в области визуального искусства (цифровое искусство, графический дизайн, фотография), полностью упустив музыку, литературу и кино. Поскольку все субагенты корректно выполнили свои назначенные задачи, узкая декомпозиция координатора является очевидной первопричиной недостаточного покрытия.

Вопрос 5 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Субагент веб-поиска возвращает результаты только для 3 из 5 запрошенных категорий источников (сайты конкурентов и отраслевые отчёты — успешно, но архивы новостей и ленты социальных сетей завершились по тайм-ауту). Субагент анализа документов успешно обработал все предоставленные документы. Субагент синтеза должен создать резюме результатов из входных данных смешанного качества. Какая стратегия распространения ошибок наиболее эффективна?

Какая стратегия распространения ошибок наиболее эффективна?

- A) Продолжить синтез, используя только успешные источники, генерируя результат без указания, какие данные были недоступны.
- B) Субагент синтеза возвращает ошибку координатору, указывая на неполные данные от вышестоящих агентов, что запускает полный повтор или сбой задачи.
- C) Субагент синтеза запрашивает координатора повторить запрос к источникам с тайм-аутом с увеличенным временем ожидания, обеспечивая полный охват данных перед началом синтеза.
- D) Структурировать результат синтеза с аннотациями покрытия, указывающими, какие выводы хорошо подкреплены, а в каких тематических областях есть пробелы из-за недоступных источников. **[ВЕРНО]**

Почему D: Структурирование результатов синтеза с аннотациями покрытия воплощает принцип плавной деградации с прозрачностью, позволяя нижестоящим потребителям и конечным пользователям понимать, какие выводы хорошо подкреплены, а в каких тематических областях есть пробелы. Этот подход сохраняет ценность выполненной работы, одновременно распространяя информацию о неопределённости для принятия обоснованных решений об уровне достоверности.

Вопрос 6 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Субагент анализа документов обнаруживает повреждённый PDF-файл, который не может распарсить. При проектировании обработки ошибок системы — как наиболее эффективно обработать этот сбой?

Какой подход наиболее эффективен?

- A) Вернуть ошибку с контекстом агенту-координатору, позволив ему решить, как действовать дальше. **[ВЕРНО]**
- B) Молча пропустить повреждённый документ и продолжить обработку остальных файлов, чтобы не прерывать рабочий процесс.
- C) Автоматически повторить парсинг документа три раза с экспоненциальной задержкой перед сообщением о сбое.
- D) Выбросить исключение, завершающее весь исследовательский рабочий процесс.

Почему A: Возврат ошибки с контекстом координатору является наиболее эффективным подходом, поскольку позволяет координатору принять обоснованное решение — пропустить файл, попробовать альтернативный метод парсинга или уведомить пользователя — сохраняя при этом видимость сбоя.

Вопрос 7 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Логи продакшена показывают устойчивый паттерн: запросы типа «проанализировать загруженный квартальный отчёт» маршрутизируются к агенту веб-поиска в 45% случаев вместо агента анализа документов. При изучении определений инструментов вы обнаруживаете, что агент веб-поиска имеет инструмент `analyze_content` с описанием «анализирует контент и извлекает ключевую информацию», а агент анализа документов имеет инструмент `analyze_document` с описанием «анализирует документы и извлекает ключевую информацию». Как следует решить проблему неправильной маршрутизации?

Как решить проблему неправильной маршрутизации?

- A) Добавить классификатор предварительной маршрутизации, определяющий, ссылается ли пользователь на загруженные файлы или веб-контент, до принятия решений координатором о делегировании.
- B) Переименовать инструмент веб-поиска в `extract_web_results` и обновить его описание на «обрабатывает и возвращает информацию, полученную из веб-поиска и URL-адресов». **[ВЕРНО]**
- C) Добавить few-shot примеры в промпт координатора, показывающие корректную маршрутизацию: «Пользователь загружает квартальный отчёт → агент анализа документов» и «Пользователь спрашивает о веб-странице → агент веб-поиска».
- D) Расширить описание инструмента анализа документов, включив примеры использования типа «Используйте для загруженных PDF, Word-документов и таблиц», оставив инструмент веб-поиска без изменений.

Почему B: Переименование инструмента веб-поиска в `extract_web_results` и обновление его описания с явным указанием на веб-поиск и URL-адреса напрямую устраняет первопричину, убирая семантическое пересечение между именами и описаниями двух инструментов. Это делает назначение каждого инструмента однозначным, позволяя координатору корректно различать анализ документов и веб-поиск.

Вопрос 8 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Коллега предлагает, чтобы агент анализа документов отправлял свои результаты напрямую агенту синтеза, минуя координатора. В чём главное преимущество сохранения координатора как центрального узла для всей коммуникации между субагентами?

В чём главное преимущество сохранения координатора как центрального узла?

- A) Координатор может наблюдать за всеми взаимодействиями, единообразно обрабатывать ошибки и решать, какую информацию должен получить каждый субагент. **[ВЕРНО]**
- B) Координатор объединяет множественные запросы к субагентам в пакеты, сокращая общее количество API-вызовов и общую задержку.
- C) Маршрутизация через координатора позволяет реализовать автоматическую логику повторных попыток, которую прямые межагентные вызовы не могут поддерживать.
- D) Субагенты работают с изолированной памятью, и прямая коммуникация потребует сложной сериализации, которую может выполнить только координатор.

Почему A: Паттерн координатора обеспечивает централизованную видимость всех взаимодействий, единообразную обработку ошибок по всей системе и детальный контроль над тем, какую информацию получает каждый субагент — это основные преимущества коммуникации по схеме «звезда».

Вопрос 9 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Субагент веб-поиска завершается по тайм-ауту при исследовании сложной темы. Вам нужно спроектировать, как информация об этом сбое возвращается координатору. Какой подход к распространению ошибок лучше всего обеспечивает интеллектуальное восстановление?

Какой подход к распространению ошибок лучше всего обеспечивает интеллектуальное восстановление?

- A) Вернуть координатору структурированный контекст ошибки, включающий тип сбоя, выполненный запрос, любые частичные результаты и потенциальные альтернативные подходы. **[ВЕРНО]**
- B) Перехватить тайм-аут внутри субагента и вернуть пустой набор результатов, помеченный как успешный.

- C) Реализовать автоматическую логику повторных попыток с экспоненциальной задержкой внутри субагента, возвращая общий статус «поиск недоступен» только после исчерпания всех попыток.
- D) Распространить исключение тайм-аута напрямую обработчику верхнего уровня, который завершает весь исследовательский рабочий процесс.

Почему A: Возврат структурированного контекста ошибки — включая тип сбоя, выполненный запрос, частичные результаты и альтернативные подходы — предоставляет координатору всю информацию, необходимую для принятия интеллектуальных решений о восстановлении, например, повторить запрос с изменённым запросом или продолжить с частичными результатами. Это лучший подход, поскольку сохраняет максимальный контекст для обоснованного принятия решений на уровне координации.

Вопрос 10 (Сценарий: Мультиагентная исследовательская система)

Ситуация: При проектировании системы вы предоставили агенту анализа документов доступ к универсальному инструменту `fetch_url`, чтобы он мог загружать документы по URL. Логи продакшена показывают, что этот агент теперь часто загружает страницы результатов поисковых систем для выполнения спонтанного веб-поиска — поведение, которое должно маршрутизироваться через агент веб-поиска. Это вызывает несогласованные результаты. Какое исправление наиболее эффективно?

Какое исправление наиболее эффективно?

- A) Заменить `fetch_url` инструментом `load_document`, который валидирует, что URL-адреса указывают на форматы документов. **[ВЕРНО]**
- B) Удалить `fetch_url` у агента анализа документов и маршрутизировать всю загрузку URL через координатора к агенту веб-поиска.
- C) Реализовать фильтрацию, блокирующую вызовы `fetch_url` к известным доменам поисковых систем, разрешая при этом другие URL.
- D) Добавить инструкции в промпт агента анализа документов, уточняющие, что `fetch_url` следует использовать только для загрузки URL документов, а не для поиска.

Почему A: Замена универсального инструмента на документ-специфический инструмент, валидирующий URL-адреса на соответствие форматам документов, устраняет первопричину, ограничивая возможности на уровне интерфейса. Это следует принципу наименьших привилегий, делая нежелательное поведение поиска невозможным, а не просто нежелательным.

Вопрос 11 (Сценарий: Мультиагентная исследовательская система)

Ситуация: При исследовании широкой темы вы наблюдаете, что агент веб-поиска и агент анализа документов оба исследуют одни и те же подтемы, что приводит к значительному дублированию в их

результатах. Расход токенов почти удвоился без пропорционального увеличения широты или глубины исследовательского покрытия. Как наиболее эффективно решить эту проблему?

Как наиболее эффективно решить эту проблему?

- A) Позволить обоим агентам завершить параллельную работу, затем координатор дедуплицирует пересекающиеся результаты перед передачей агенту синтеза.
- B) Координатор явно разделяет исследовательское пространство перед делегированием, назначая каждому агенту отдельные подтемы или типы источников. **[ВЕРНО]**
- C) Реализовать механизм общего состояния, где агенты логируют свою текущую область фокуса, позволяя другим агентам динамически избегать дублирования работы в процессе выполнения.
- D) Перейти к последовательному выполнению, где анализ документов запускается только после завершения веб-поиска, используя результаты веб-поиска как контекст для избежания дублирования.

Почему B: Явное разделение координатором исследовательского пространства перед делегированием является наиболее эффективным подходом, поскольку устраняет первопричину — нечёткие границы задач — до начала какой-либо работы. Это сохраняет преимущества параллельного выполнения, предотвращая дублирование усилий и напрасный расход токенов.

Вопрос 12 (Сценарий: Мультиагентная исследовательская система)

Ситуация: В ходе исследования материалов субагент веб-поиска запрашивает три категории источников с разными результатами: академические базы данных вернули 15 релевантных статей, отраслевые отчёты вернули «0 результатов», а базы патентов вернули «Тайм-аут соединения». При проектировании распространения ошибок к координатору — какой подход обеспечивает лучшие решения по восстановлению?

Какой подход обеспечивает лучшие решения по восстановлению?

- A) Агрегировать результаты в единую метрику процента успешности (например, «67% покрытие источников») с подробными логами, доступными по запросу.
- B) Сообщить и о тайм-ауте, и о «0 результатов» как о сбоях, требующих вмешательства координатора.
- C) Субагент повторяет транзитные сбои внутренне и сообщает только о постоянных ошибках.
- D) Различать сбои доступа (тайм-аут), требующие решения о повторе, и валидные пустые результаты («0 результатов»), представляющие успешные запросы. **[ВЕРНО]**

Почему D: Это правильный ответ, поскольку тайм-аут (сбой доступа) и «0 результатов» (валидный пустой результат) — семантически различные результаты, требующие разных реакций. Их различение позволяет координатору повторить запрос к базе патентов с тайм-аутом, принимая при этом пустые результаты отраслевых отчётов как валидную и информативную находку.

Вопрос 13 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Мониторинг продакшена выявляет непостоянное качество синтеза. Когда агрегированные результаты составляют ~75K токенов, агент синтеза надёжно цитирует информацию из первых 15K токенов (заголовки и сниппеты веб-поиска) и последних 10K токенов (выводы анализа документов), но часто пропускает критические находки из средних 50K токенов — даже когда эти находки напрямую отвечают на исследовательский вопрос. Как следует реструктурировать агрегированный ввод?

Как следует реструктурировать агрегированный ввод?

- A) Резюмировать все выходные данные субагентов до менее чем 20K токенов перед агрегацией, обеспечивая нахождение контента в пределах надёжного диапазона обработки модели.
- B) Транслировать результаты субагентов агенту синтеза инкрементально, обрабатывая сначала результаты веб-поиска до завершения, а затем вводя результаты анализа документов.
- C) Разместить резюме ключевых находок в начале агрегированного ввода и организовать детальные результаты с явными заголовками секций для облегчения навигации. **[ВЕРНО]**
- D) Реализовать ротацию, чередующую, результаты какого субагента размещаются первыми в разных исследовательских задачах, обеспечивая обоим источникам равное позиционирование в начале с течением времени.

Почему C: Размещение резюме ключевых находок в начале использует эффект первичности, гарантируя, что критическая информация занимает наиболее надёжно обрабатываемую позицию. Добавление явных заголовков секций по всему агрегированному вводу помогает модели навигировать и обращать внимание на контент в средней части, напрямую смягчая феномен «потеря в середине».

Вопрос 14 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Во время тестирования объединённые выходные данные агента веб-поиска (85K токенов, включая контент страниц) и агента анализа документов (70K токенов, включая цепочки рассуждений) составляют 155K токенов, но агент синтеза работает оптимально при входных данных менее 50K токенов. Какое решение наиболее эффективно?

Какое решение наиболее эффективно?

- A) Модифицировать вышестоящих агентов для возврата структурированных данных (ключевые факты, цитаты, оценки релевантности) вместо объёмного контента и рассуждений. **[ВЕРНО]**
- B) Добавить промежуточного агента суммаризации, который конденсирует находки перед передачей на синтез.
- C) Обрабатывать находки агентом синтеза последовательными пакетами, поддерживая состояние между вызовами.
- D) Хранить находки в векторной базе данных и предоставить агенту синтеза инструменты поиска для запросов в процессе работы.

Почему A: Модификация вышестоящих агентов для возврата структурированных данных (ключевые факты, цитаты, оценки релевантности) устраняет первопричину, сокращая объём токенов в источнике при сохранении существенной информации. Это исключает объёмный контент страниц и цепочки рассуждений, которые раздувают количество токенов без добавления ценности для этапа синтеза.

Вопрос 15 (Сценарий: Мультиагентная исследовательская система)

Ситуация: Во время тестирования вы наблюдаете, что агенту синтеза часто требуется верифицировать конкретные утверждения при объединении результатов. В настоящее время, когда нужна верификация, агент синтеза возвращает управление координатору, который вызывает агент веб-поиска, а затем повторно вызывает синтез с результатами. Это добавляет 2–3 дополнительных цикла на задачу и увеличивает задержку на 40%. Ваша оценка показывает, что 85% этих верификаций — простые проверки фактов (даты, имена, статистика), а 15% требуют более глубокого исследования. Какой подход наиболее эффективно снижает накладные расходы при сохранении надёжности системы?

Какой подход наиболее эффективен?

- A) Предоставить агенту синтеза доступ ко всем инструментам веб-поиска, чтобы он мог обрабатывать любую потребность в верификации напрямую без циклов через координатора.
- B) Поручить агенту синтеза накапливать все потребности в верификации и возвращать их пакетом координатору в конце своего прохода, который затем отправляет их все агенту веб-поиска сразу.
- C) Поручить агенту веб-поиска проактивно кэшировать дополнительный контекст вокруг каждого источника во время первоначального исследования, предвосхищая, что агенту синтеза может потребоваться верифицировать.
- D) Предоставить агенту синтеза инструмент `verify_fact` с ограниченной областью для простых проверок, а сложные верификации продолжать делегировать агенту веб-поиска через координатора. **[ВЕРНО]**

Почему D: Предоставление инструмента верификации фактов с ограниченной областью позволяет обработать 85% простых проверок напрямую, устраняя большинство циклов, при этом сохраняя путь делегирования через координатора для 15% сложных верификаций. Это применяет принцип наименьших привилегий, удерживая агент синтеза сфокусированным на своей основной задаче, при этом значительно снижая задержку.

Сценарий: Claude Code для непрерывной интеграции

Вопрос 16 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваш CI-пайплайн запускает CLI Claude Code (в режиме `--print`) с использованием CLAUDE.md для предоставления контекста проекта при ревью кода, и разработчики в целом находят ревью содержательными. Однако они сообщают, что интеграция находок в рабочий процесс затруднена — Claude выдаёт повествовательные абзацы, которые приходится вручную копировать в комментарии к PR. Команда хочет автоматически публиковать каждую находку как отдельный inline-комментарий к PR в соответствующем месте кода, что требует структурированных данных с путём к файлу, номером строки, уровнем серьёзности и предложенным исправлением. Какой подход наиболее эффективен?

Какой подход наиболее эффективен?

- A) Добавить раздел «Формат вывода ревью» в CLAUDE.md с примерами структурированных находок, чтобы Claude выучил ожидаемый формат из контекста проекта.
- B) Использовать флаги CLI `--output-format json` и `--json-schema` для принудительного структурирования находок, затем парсить вывод для публикации inline-комментариев через GitHub API. **[ВЕРНО]**
- C) Включить явные инструкции по форматированию в промпт ревью, требуя, чтобы каждая находка следовала разбираемому шаблону типа `[FILE:путь] [LINE:n] [SEVERITY:уровень] ...`.
- D) Сохранить повествовательный формат ревью, но добавить этап суммаризации, использующий Claude для генерации структурированного JSON-резюме находок.

Почему B: Использование `--output-format json` с `--json-schema` обеспечивает структурированный вывод на уровне CLI, гарантируя корректно сформированный JSON с необходимыми полями (путь к файлу, номер строки, серьёзность, предложенное исправление), который можно надёжно распарсить и опубликовать как inline-комментарии к PR через GitHub API. Это наиболее эффективный подход, поскольку использует встроенные возможности CLI, специально предназначенные для принудительного структурирования вывода.

Вопрос 17 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваша команда использует Claude Code для генерации предложений по коду, но вы замечаете паттерн: неочевидные проблемы — оптимизации производительности, ломающие граничные случаи, очистки, неожиданно меняющие поведение — обнаруживаются только когда другой член команды ревьюит PR. Рассуждения Claude во время генерации показывают, что он рассматривал эти случаи, но пришёл к выводу, что его подход корректен. Какой подход напрямую устраняет первопричину этого ограничения самоконтроля?

Какой подход напрямую устраняет первопричину?

- A) Запустить второй независимый экземпляр Claude Code для ревью изменений без доступа к рассуждениям генератора. **[ВЕРНО]**

- B) Включить режим расширенного обдумывания для этапа генерации, позволяя более тщательное обсуждение перед выдачей предложений.
- C) Добавить явные инструкции по самопроверке в промпт генерации, прося Claude критиковать собственные предложения перед финализацией вывода.
- D) Включить полные тестовые файлы и документацию в контекст промпта, чтобы Claude лучше понимал ожидаемое поведение во время генерации.

Почему A: Использование второго независимого экземпляра Claude Code без доступа к рассуждениям генератора напрямую устраняет первопричину, исключая предвзятость подтверждения. Эта свежая перспектива отражает преимущество человеческого peer review, где другой член команды замечает проблемы, которые автор рационализировал.

Вопрос 18 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Компонент ревью кода работает итеративно: Claude анализирует изменённый файл, затем может запросить связанные файлы (импорты, базовые классы, тесты) через вызов инструментов для понимания контекста перед предоставлением финальной обратной связи. Ваше приложение определяет инструмент, позволяющий Claude запрашивать содержимое файлов; Claude вызывает этот инструмент, получает результаты и продолжает анализ. Вы оцениваете пакетную обработку для снижения затрат на API. Каково основное техническое ограничение при рассмотрении пакетной обработки для этого рабочего процесса?

Каково основное техническое ограничение?

- A) Пакетная обработка не имеет идентификаторов корреляции запросов для сопоставления выходных данных с входными запросами.
- B) Асинхронная модель не позволяет выполнять инструменты в ходе запроса и возвращать результаты для продолжения анализа Claude. **[ВЕРНО]**
- C) Batch API не поддерживает определения инструментов в параметрах запроса.
- D) Задержка пакетной обработки до 24 часов слишком велика для обратной связи по пулл-реквестам, хотя в остальном рабочий процесс мог бы функционировать.

Почему B: Асинхронная модель Batch API типа «отправил и забыл» означает, что нет механизма для перехвата вызова инструмента в ходе запроса, выполнения инструмента и возврата результатов для продолжения анализа Claude. Это фундаментально несовместимо с итеративными рабочими процессами вызова инструментов, требующими нескольких раундов вызова и ответа инструментов в рамках одного логического взаимодействия.

Вопрос 19 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваша система CI/CD выполняет три типа анализа на базе Claude: (1) быстрые проверки стиля на каждый PR, которые блокируют слияние до завершения, (2) комплексные аудиты

безопасности всей кодовой базы, запускаемые еженедельно, и (3) генерация тестовых кейсов, запускаемая еженочно для недавно изменённых модулей. Message Batches API предлагает экономию 50%, но обработка может занять до 24 часов. Вы хотите оптимизировать затраты на API при сохранении приемлемого опыта разработчиков. Какая комбинация корректно сопоставляет каждую задачу с подходом к API?

Какая комбинация корректна?

- A) Использовать Message Batches API для всех трёх задач для максимальной экономии 50%, настроив пайплайн на опрос завершения пакета.
- B) Использовать синхронные вызовы для проверок стиля PR; использовать Message Batches API для еженедельных аудитов безопасности и ночной генерации тестов. **[ВЕРНО]**
- C) Использовать синхронные вызовы для всех трёх задач ради стабильного времени отклика, полагаясь на кэширование промптов для снижения затрат по всем рабочим нагрузкам.
- D) Использовать синхронные вызовы для проверок стиля PR и ночной генерации тестов; использовать Message Batches API только для еженедельных аудитов безопасности.

Почему B: Проверки стиля PR блокируют разработчиков и требуют немедленных ответов через синхронные вызовы, тогда как еженедельные аудиты безопасности и ночная генерация тестов — это плановые задачи с гибкими сроками, которые легко допускают окно пакетной обработки до 24 часов, обеспечивая экономию 50% на обеих задачах.

Вопрос 20 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваши автоматизированные ревью выявляют реальные проблемы, но разработчики сообщают, что обратная связь не является действенной. Находки содержат формулировки типа «сложная логика распределения тикетов» или «потенциальный null pointer» без указания, что именно нужно изменить. Когда вы добавляете детальные инструкции типа «всегда включать конкретные предложения по исправлению», модель всё равно выдаёт непоследовательный результат — иногда детальный, иногда расплывчатый. Какая техника промптинга наиболее надёжно обеспечит стабильно действенную обратную связь?

Какая техника промптинга наиболее надёжна?

- A) Дополнительно уточнить инструкции с более явными требованиями к каждой части формата обратной связи (расположение, проблема, серьёзность, предложенное исправление).
- B) Расширить контекстное окно для включения большего объёма окружающей кодовой базы, чтобы модель имела достаточно информации для предложения конкретных исправлений.
- C) Реализовать двухпроходный подход, где один промпт выявляет проблемы, а второй генерирует исправления, позволяя специализацию.
- D) Добавить 3–4 few-shot примера, показывающих точный требуемый формат: выявленная проблема, расположение в коде, конкретное предложение по исправлению. **[ВЕРНО]**

Почему D: Few-shot примеры — наиболее эффективная техника для достижения стабильного формата вывода, когда инструкции сами по себе дают переменные результаты. Предоставление 3–4 примеров, показывающих точный желаемый формат (проблема, расположение, конкретное

исправление), даёт модели конкретный паттерн для следования, что надёжнее абстрактных инструкций.

Вопрос 21 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваш CI-пайплайн включает два режима ревью кода на базе Claude: хук pre-merge-commit, блокирующий слияние PR до завершения, и «глубокий анализ», запускаемый на ночь, с опросом завершения пакета и публикацией детальных предложений в PR. Вы хотите снизить затраты на API с помощью Message Batches API, предлагающего экономию 50%, но требующего опроса и до 24 часов на обработку. Какой режим должен использовать пакетную обработку?

Какой режим должен использовать пакетную обработку?

- A) Только хук pre-merge-commit.
- B) Только глубокий анализ. **[ВЕРНО]**
- C) Оба режима.
- D) Ни один из режимов.

Почему B: Глубокий анализ — идеальный кандидат для пакетной обработки, поскольку он уже запускается на ночь, допускает задержку и использует модель опроса для проверки завершения перед публикацией результатов — это идеально соответствует асинхронной, основанной на опросе архитектуре Message Batches API при получении экономии 50%.

Вопрос 22 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваше автоматизированное ревью анализирует комментарии и docstring. Текущий промпт инструктирует Claude «проверять, что комментарии точны и актуальны». Находки часто помечают допустимые паттерны (маркеры TODO, простые описания), при этом пропуская комментарии, описывающие поведение, которое код больше не реализует. Какое изменение устраняет первопричину этого непоследовательного анализа?

Какое изменение устраняет первопричину?

- A) Включить данные `git blame`, чтобы Claude мог выявлять комментарии, предшествующие недавним изменениям кода.
- B) Добавить few-shot примеры вводящих в заблуждение комментариев, чтобы помочь модели распознавать аналогичные паттерны в кодовой базе.
- C) Отфильтровать паттерны комментариев TODO, FIXME и описательных комментариев перед анализом для снижения шума.
- D) Указать явные критерии: отмечать комментарии только тогда, когда заявленное в них поведение противоречит фактическому поведению кода. **[ВЕРНО]**

Почему D: Указание явных критериев — отмечать комментарии только при противоречии заявленного поведения фактическому поведению кода — напрямую устраняет первопричину, заменяя расплывчатую инструкцию точным определением того, что считается проблемой. Это устраняет как ложные срабатывания на допустимых паттернах, так и пропуски действительно вводящих в заблуждение комментариев.

Вопрос 23 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваша система автоматического ревью кода показывает непоследовательные оценки серьёзности — аналогичные проблемы, такие как риски null pointer, получают уровень «critical» в одних PR, но только «medium» в других. Доверие разработчиков снижается, так как команды не могут предсказать, какие находки требуют немедленного внимания. Как наиболее эффективно улучшить согласованность оценок серьёзности?

Как наиболее эффективно улучшить согласованность?

- A) Включить явные критерии серьёзности в промпт с конкретными примерами кода для каждого уровня серьёзности. **[ВЕРНО]**
- B) Модифицировать промпт, прося Claude оценивать серьёзность относительно других проблем в том же PR, чтобы наиболее серьёзная проблема всегда отмечалась как critical, а остальные оценивались пропорционально.
- C) Добавить файл CLAUDE.md, перечисляющий типы проблем и их серьёзность по умолчанию, инструктируя Claude обращаться к этому сопоставлению при назначении оценок.
- D) Запросить у Claude обоснование каждой оценки серьёзности, затем использовать эти обоснования для ручной калибровки и корректировки оценок при ревью.

Почему A: Включение явных критериев серьёзности с конкретными примерами кода напрямую устраняет первопричину непоследовательности, убирая неоднозначность в определении каждого уровня серьёзности. Это проверенная техника промпт-инженерии, дающая модели чёткие ориентиры для классификации, что ведёт к более надёжным и предсказуемым оценкам серьёзности.

Вопрос 24 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваше автоматизированное ревью генерирует предложения по тестовым кейсам для каждого PR. При ревью PR, добавляющего отслеживание завершения курсов, Claude предлагает 10 тестовых кейсов, но обратная связь разработчиков показывает, что 6 дублируют сценарии, уже покрытые существующим набором тестов. Какое изменение наиболее эффективно снизит количество дублирующих предложений?

Какое изменение наиболее эффективно?

- A) Включить существующий тестовый файл в контекст, чтобы Claude мог определить, какие сценарии уже покрыты. **[ВЕРНО]**
- B) Снизить количество запрашиваемых предложений с 10 до 5, предполагая, что Claude приоритизирует наиболее ценные кейсы первыми.
- C) Добавить инструкции, направляющие Claude фокусироваться исключительно на граничных случаях и условиях ошибок, а не на успешных сценариях.
- D) Реализовать постобработку, фильтрующую предложения, описания которых совпадают по ключевым словам с именами существующих тестов.

Почему A: Включение существующего тестового файла в контекст напрямую устраняет первопричину дублирования: Claude может избежать предложения уже покрытых сценариев только если знает, какие тесты уже существуют. Это предоставляет Claude информацию, необходимую для определения, какие предложения будут действительно новыми и ценными.

Вопрос 25 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: После начального автоматического ревью, выявившего 12 находок, разработчик отправляет новые коммиты для устранения проблем. При повторном запуске ревью выдаёт 8 находок, но разработчики сообщают, что 5 из них дублируют предыдущие комментарии к коду, который уже был исправлен в новых коммитах. Как наиболее эффективно устранить эту избыточную обратную связь, сохраняя тщательность анализа?

Как наиболее эффективно устранить избыточную обратную связь?

- A) Запускать ревью только при создании PR и в финальном pre-merge состоянии, пропуская промежуточные коммиты.
- B) Добавить постобработочный фильтр, удаляющий находки, совпадающие с предыдущими по путям к файлам и описаниям проблем, перед публикацией комментариев.
- C) Ограничить область ревью только файлами, изменёнными в последнем push, исключая файлы из более ранних коммитов.
- D) Включить предыдущие находки ревью в контекст, инструктируя Claude сообщать только о новых или всё ещё неустранённых проблемах. **[ВЕРНО]**

Почему D: Включение предыдущих находок ревью в контекст позволяет Claude интеллектуально различать новые проблемы и те, которые уже были устранены недавними коммитами. Этот подход сохраняет тщательность анализа, используя способность Claude к рассуждению для избежания избыточной обратной связи по исправленному коду.

Вопрос 26 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваш скрипт пайплайна выполняет `claude "Analyze this pull request for security issues"`, но задание зависает бесконечно. Логи показывают, что Claude Code ожидает

интерактивного ввода. Каков правильный подход для запуска Claude Code в автоматизированном пайплайне?

Каков правильный подход?

- A) Добавить флаг `--batch`: `claude --batch "Analyze this pull request for security issues"`.
- B) Добавить флаг `-p`: `claude -p "Analyze this pull request for security issues"`. **[ВЕРНО]**
- C) Перенаправить stdin из `/dev/null`: `claude "Analyze this pull request for security issues" < /dev/null`.
- D) Установить переменную окружения `CLAUDE_HEADLESS=true` перед запуском команды.

Почему B: Флаг `-p` (или `--print`) — это документированный способ запуска Claude Code в неинтерактивном режиме. Он обрабатывает заданный промпт, выводит результат в stdout и завершается без ожидания пользовательского ввода, что идеально подходит для CI/CD-пайплайнов.

Вопрос 27 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Пулл-реквест изменяет 14 файлов в модуле отслеживания запасов. Ваше однопроходное ревью, анализирующее все файлы вместе, выдаёт непоследовательные результаты: детальная обратная связь по одним файлам, но поверхностные комментарии по другим, пропущенные очевидные баги и противоречивая обратная связь — паттерн помечается как проблемный в одном файле, но одобряется идентичный код в другом файле того же PR. Как следует реструктурировать ревью?

Как следует реструктурировать ревью?

- A) Запустить три независимых прохода ревью по полному PR и отмечать только проблемы, которые появляются минимум в двух из трёх запусков.
- B) Разделить на фокусированные проходы: анализировать каждый файл индивидуально на локальные проблемы, затем запустить отдельный интеграционно-ориентированный проход для изучения межфайловых потоков данных. **[ВЕРНО]**
- C) Требовать от разработчиков разделять крупные PR на меньшие отправки по 3–4 файла перед запуском автоматического ревью.
- D) Перейти на модель более высокого уровня с большим контекстным окном, чтобы уделить всем 14 файлам достаточное внимание за один проход.

Почему B: Разделение ревью на фокусированные проходы по каждому файлу напрямую устраняет первопричину рассеивания внимания, обеспечивая стабильную глубину и надёжное выявление локальных проблем. Отдельный интеграционно-ориентированный проход затем обрабатывает межфайловые аспекты, такие как зависимости потоков данных, покрывая оба измерения качества ревью.

Вопрос 28 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваше автоматическое ревью кода в среднем выдаёт 15 находок на пулл-реквест, при этом разработчики сообщают о 40% ложных срабатываний. Узким местом является время исследования: разработчикам приходится переходить к каждой находке, чтобы прочитать обоснование Claude, прежде чем решить — устранять или отклонять. Ваш CLAUDE.md уже содержит исчерпывающие правила для допустимых паттернов, и заинтересованные стороны отклонили любой подход, фильтрующий находки до просмотра разработчиками. Какое изменение лучше всего решит проблему времени на исследование?

Какое изменение лучше всего решит проблему?

- A) Требовать от Claude включать своё обоснование и оценку уверенности непосредственно в каждую находку. **[ВЕРНО]**
- B) Добавить постпроцессор, анализирующий паттерны находок и автоматически подавляющий те, что соответствуют историческим сигнатурам ложных срабатываний.
- C) Категоризировать находки как «блокирующие проблемы» vs «предложения» с различными требованиями к ревью по уровням.
- D) Настроить Claude отображать только находки с высокой уверенностью, отфильтровывая неуверенные флаги до того, как разработчики их увидят.

Почему A: Включение обоснования и оценок уверенности непосредственно в каждую находку напрямую решает проблему времени на исследование, позволяя разработчикам быстро оценивать находки без необходимости переходить к каждой отдельно. Этот подход соблюдает ограничение на отсутствие фильтрации, поскольку все находки остаются видимыми, при этом триаж значительно ускоряется.

Вопрос 29 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Анализ вашего автоматического ревью кода показывает значительные различия в частоте ложных срабатываний по категориям находок. Находки по безопасности и корректности имеют 8% ложных срабатываний, находки по производительности — 18%, находки по стилю и именованию — 52%, а по документации — 48%. Опросы разработчиков показывают растущее недоверие — многие начали отклонять находки без просмотра, потому что «половина ошибочна». Категории с высоким процентом ложных срабатываний подрывают доверие к точным категориям. Какой подход лучше всего восстановит доверие разработчиков при одновременном улучшении системы?

Какой подход лучше всего восстановит доверие?

- A) Временно отключить категории с высоким процентом ложных срабатываний (стиль, именование, документация) и оставить только высокоточные категории, работая над улучшением промптов. **[ВЕРНО]**

- B) Сохранить все категории, но отображать оценку уверенности с каждой находкой, позволяя разработчикам решать, какие исследовать.
- C) Сохранить все категории включёнными, добавляя few-shot примеры для улучшения точности каждой категории в течение ближайших недель.
- D) Применить единообразное снижение строгости по всем категориям, чтобы привести общий уровень ложных срабатываний к приемлемому.

Почему A: Временное отключение категорий с высоким процентом ложных срабатываний (стиль, именование, документация) немедленно останавливает эрозию доверия, убирая шум, из-за которого разработчики отклоняют все находки, при этом сохраняя ценность высокоточных категорий, таких как безопасность и корректность. Этот подход даёт время для улучшения промптов проблемных категорий перед их повторным включением, восстанавливая доверие через продемонстрированную точность.

Вопрос 30 (Сценарий: Claude Code для непрерывной интеграции)

Ситуация: Ваша команда хочет снизить затраты на API для автоматизированного анализа. В настоящее время синхронные вызовы Claude обеспечивают два рабочих процесса: (1) блокирующая pre-merge проверка, которая должна завершиться до того, как разработчики смогут выполнить слияние, и (2) отчёт о техническом долге, генерируемый за ночь для просмотра на следующее утро. Ваш менеджер предлагает перевести оба на Message Batches API для экономии 50%. Как следует оценить это предложение?

Как следует оценить это предложение?

- A) Перевести оба на пакетную обработку с откатом на синхронные вызовы, если пакеты обрабатываются слишком долго.
- B) Перевести оба рабочих процесса на пакетную обработку с опросом статуса для проверки завершения.
- C) Использовать пакетную обработку только для отчётов о техническом долге; сохранить синхронные вызовы для pre-merge проверок. **[ВЕРНО]**
- D) Сохранить синхронные вызовы для обоих рабочих процессов, чтобы избежать проблем с порядком пакетных результатов.

Почему C: Это правильный подход, поскольку время обработки Message Batches API до 24 часов без гарантированного SLA по задержке делает его идеальным для ночных отчётов о техническом долге, но неприемлемым для блокирующих pre-merge проверок, где разработчики ожидают. Это сопоставляет каждый рабочий процесс с соответствующим API на основе требований к задержке.

Сценарий: Генерация кода с Claude Code

Вопрос 31 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы попросили Claude Code реализовать функцию, которая преобразует ответы API во внутренний нормализованный формат. После двух итераций структура вывода всё ещё не соответствует ожиданиям — некоторые поля вложены по-другому, а временные метки отформатированы неправильно. Вы описывали требования в прозаической форме, но Claude каждый раз интерпретирует их по-разному.

Какой подход наиболее эффективен для следующей итерации?

- A) Написать JSON-схему, описывающую ожидаемую структуру вывода, и после каждой итерации валидировать результат Claude по этой схеме.
- B) Предоставить 2–3 конкретных примера «вход-выход», демонстрирующих ожидаемое преобразование для репрезентативных ответов API. **[ВЕРНО]**
- C) Переписать требования с большей технической точностью, указав конкретные соответствия полей, правила вложенности и строки формата временных меток.
- D) Попросить Claude объяснить своё текущее понимание требований, чтобы выявить, где расходятся интерпретации.

Почему B: Предоставление конкретных примеров «вход-выход» — наиболее эффективный подход, поскольку он устраняет неоднозначность, присущую текстовым описаниям, показывая Claude точный ожидаемый результат преобразования. Это напрямую устраняет корневую причину — неверную интерпретацию текстовых требований — давая однозначные, конкретные образцы для вложенности полей и форматирования временных меток.

Вопрос 32 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вам нужно добавить Slack как новый канал уведомлений. В существующей кодовой базе есть ясные, устоявшиеся паттерны для email, SMS и push-каналов. Однако API Slack предлагает принципиально разные подходы к интеграции — входящие вебхуки (простые, только односторонние), токены ботов (позволяют подтверждать доставку и управлять программно) или Slack Apps (двунаправленные события, требуют одобрения воркспейса). В вашей задаче указано «добавить поддержку Slack» без уточнения метода интеграции и необходимости продвинутых функций вроде отслеживания доставки.

Как следует подойти к этой задаче?

- A) Начать прямое выполнение, используя входящие вебхуки, чтобы соответствовать существующему паттерну односторонних уведомлений.
- B) Перейти в режим планирования, чтобы изучить варианты интеграции и их архитектурные последствия, а затем представить рекомендацию перед реализацией. **[ВЕРНО]**
- C) Начать прямое выполнение, создав каркас класса Slack-канала по существующим паттернам, отложив выбор метода интеграции на потом.
- D) Начать прямое выполнение, используя подход с токеном бота для обеспечения возможности подтверждения доставки.

Почему B: Это правильный ответ, поскольку интеграция со Slack предполагает несколько допустимых подходов с существенно различающимися архитектурными последствиями, а требования неоднозначны. Использование режима планирования для изучения компромиссов между вебхуками, токенами ботов и Slack Apps позволяет принять обоснованное решение и согласовать подход с командой до начала реализации.

Вопрос 33 (Сценарий: Генерация кода с Claude Code)

Ситуация: Ваш файл CLAUDE.md вырос до более чем 400 строк, содержащих стандарты кодирования, соглашения по тестированию, подробный чек-лист для ревью PR, инструкции по деплою и процедуры миграции базы данных. Вы хотите, чтобы Claude всегда следовал стандартам кодирования и соглашениям по тестированию, но применял рекомендации по ревью PR, деплою и миграциям только при выполнении соответствующих задач.

Какой подход к реструктуризации наиболее эффективен?

- A) Перенести все рекомендации в отдельные файлы Skills, организованные по типу рабочего процесса, оставив в CLAUDE.md лишь краткое описание проекта.
- B) Оставить весь контент в CLAUDE.md, но использовать синтаксис @import для организации в отдельно поддерживаемые файлы по категориям.
- C) Разделить CLAUDE.md на файлы в `.claude/rules/` с glob-паттернами, привязанными к путям, чтобы каждое правило загружалось только для соответствующих типов файлов.
- D) Оставить универсальные стандарты в CLAUDE.md и создать Skills для специфичных рабочих процессов (ревью PR, деплой, миграции) с ключевыми словами-триггерами. **[ВЕРНО]**

Почему D: Это наиболее эффективный подход, поскольку содержимое CLAUDE.md загружается в каждой сессии, гарантируя постоянное применение стандартов кодирования и соглашений по тестированию, тогда как Skills вызываются по требованию, когда Claude обнаруживает соответствующие ключевые слова-триггеры, что идеально подходит для специфичных рабочих процессов вроде ревью PR, деплоя и миграций.

Вопрос 34 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вам поручено реструктуризировать монолитное приложение команды в микросервисы. Это затронет изменения в десятках файлов и потребует принятия решений о границах сервисов и зависимостях модулей.

Какой подход следует выбрать?

- A) Перейти в режим планирования, чтобы исследовать кодовую базу, понять зависимости и спроектировать подход к реализации перед внесением изменений. **[ВЕРНО]**
- B) Начать в режиме прямого выполнения и переключиться в режим планирования только при обнаружении неожиданной сложности в процессе реализации.
- C) Начать с прямого выполнения и вносить изменения инкрементально, позволяя реализации выявить естественные границы сервисов.

- D) Использовать прямое выполнение с подробными предварительными инструкциями, детально описывающими структуру каждого сервиса.

Почему A: Использование режима планирования для исследования кодовой базы, понимания зависимостей и проектирования подхода перед внесением изменений — правильная стратегия для сложной архитектурной реструктуризации, такой как разделение монолита. Это позволяет безопасно исследовать и принимать обоснованные решения о границах сервисов до коммита потенциально дорогостоящих изменений в десятках файлов.

Вопрос 35 (Сценарий: Генерация кода с Claude Code)

Ситуация: Ваша команда создала скилл `/analyze-codebase`, выполняющий комплексный анализ кода — сканирование зависимостей, подсчёт покрытия тестами и метрики качества кода. После запуска этой команды участники команды сообщают, что Claude становится менее отзывчивым в сессии и теряет контекст исходной задачи.

Как наиболее эффективно решить эту проблему, сохранив полные возможности анализа?

- A) Добавить `context: fork` в frontmatter скилла для запуска анализа в изолированном контексте субагента. **[ВЕРНО]**
- B) Добавить `model: haiku` в frontmatter для использования более быстрой и эффективной модели при анализе.
- C) Разделить скилл на три меньших скилла, каждый из которых генерирует меньше вывода.
- D) Добавить инструкции в скилл для сжатия всех результатов в краткую сводку перед отображением.

Почему A: Использование `context: fork` в frontmatter скилла запускает анализ в изолированном контексте субагента, что предотвращает засорение контекстного окна основной сессии объёмным выводом и потерю Claude контекста исходной задачи. Это сохраняет полные возможности анализа, поддерживая отзывчивость основной сессии.

Вопрос 36 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы создали скилл `/commit` в `.claude/skills/commit/SKILL.md`, который использует ваша команда. Один из разработчиков хочет настроить его под свой персональный рабочий процесс (другой формат сообщений коммитов, дополнительные проверки), не затрагивая коллег.

Что вы порекомендуете?

- A) Создать персональную версию в `~/.claude/skills/` с другим именем, например `/my-commit`. **[ВЕРНО]**
- B) Добавить условную логику на основе имени пользователя в frontmatter проектного скилла.
- C) Создать персональную версию по пути `~/.claude/skills/commit/SKILL.md` с тем же именем.
- D) Установить `override: true` в frontmatter персонального скилла для приоритета над проектной версией.

Почему A: Персональные скиллы имеют приоритет над проектными скиллами с тем же именем, поэтому повторное использование имени `commit` незаметно скроет скилл команды именно для этого разработчика — он перестанет получать обновления всякий раз, когда команда улучшает `/commit`, и должен будет помнить, что запускает другой скилл под тем же именем команды. Назвав персональный вариант `/my-commit`, этот конфликт полностью устраняется: разработчик продолжает использовать поддерживаемый командой `/commit` и получает отдельный, ясно названный скилл для личного рабочего процесса, без риска перепутать их или пропустить обновления команды.

Вопрос 37 (Сценарий: Генерация кода с Claude Code)

Ситуация: Ваша команда использует Claude Code уже несколько месяцев. Недавно три разработчика сообщили, что Claude корректно следует рекомендации «всегда включать комплексную обработку ошибок», но четвёртый разработчик, только что присоединившийся к команде, сообщает, что Claude не следует этой рекомендации. Все четыре разработчика работают в одном репозитории и имеют актуальный код.

Какова наиболее вероятная причина и как это исправить?

- A) Рекомендация находится в файлах `~/.claude/CLAUDE.md` (на уровне пользователя) у изначальных разработчиков, а не в проектном `.claude/CLAUDE.md`. Следует перенести инструкцию в файл проектного уровня, чтобы все участники команды её получали. **[ВЕРНО]**
- B) Файл `~/.claude/CLAUDE.md` нового разработчика содержит конфликтующие инструкции, которые переопределяют проектные настройки. Ему следует удалить конфликтующую секцию из своей пользовательской конфигурации.
- C) Claude Code со временем формирует модели предпочтений для каждого пользователя через повторяющиеся взаимодействия. Новому разработчику нужно многократно указывать требование по обработке ошибок, пока Claude не запомнит его предпочтения.
- D) Claude Code кэширует содержимое CLAUDE.md после первого чтения. У изначальных разработчиков кэшированные версии, а новый разработчик загрузил файл после его изменения. Всем разработчикам следует очистить кэш Claude Code.

Почему A: Это наиболее вероятная причина: если рекомендация по обработке ошибок была добавлена в пользовательский `~/.claude/CLAUDE.md` каждого из изначальных разработчиков, а не в проектный `.claude/CLAUDE.md`, новые участники команды не будут её получать. Перенос инструкции в файл конфигурации проектного уровня гарантирует, что все текущие и будущие участники команды автоматически получат эту рекомендацию.

Вопрос 38 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы обнаружили, что включение 2–3 полных примеров реализации эндпоинтов в качестве контекста значительно улучшает консистентность при генерации новых эндпоинтов API. Однако этот контекст полезен только при создании новых эндпоинтов — не при исправлении ошибок, ревью кода или другой работе в директории API.

Какой подход к конфигурации наиболее эффективен?

- A) Добавить примеры кода эндпоинтов с документацией паттернов в проектный файл CLAUDE.md, чтобы они были автоматически доступны.
- B) Вручную ссылаться на примеры эндпоинтов в каждом запросе на генерацию, копируя соответствующий код в промпт.
- C) Настроить правила для конкретных путей в `.claude/rules/api/`, включающие примеры кода и активирующиеся при работе в директории API.
- D) Создать скилл, ссылающийся на примеры эндпоинтов и содержащий инструкции по следованию паттернам, вызываемый по требованию через слеш-команду. **[ВЕРНО]**

Почему D: Создание скилла с примерами эндпоинтов и инструкциями по следованию паттернам позволяет вызывать его по требованию через слеш-команду, гарантируя загрузку контекста только при генерации новых эндпоинтов, а не во время несвязанных задач вроде исправления ошибок или ревью кода.

Вопрос 39 (Сценарий: Генерация кода с Claude Code)

Ситуация: Ваша команда создала скилл `/migration`, генерирующий файлы миграции базы данных. Скилл принимает имя миграции через `$ARGUMENTS`. В продакшене наблюдаются три проблемы: (1) разработчики часто вызывают скилл без аргументов, что приводит к плохо именованным файлам, (2) скилл иногда использует детали схемы базы данных из несвязанных предыдущих разговоров, и (3) разработчик случайно запустил деструктивную очистку тестов, когда скилл имел широкий доступ к инструментам.

Какой подход к конфигурации решает все три проблемы?

- A) Использовать позиционные параметры `$1` и `$2` вместо `$ARGUMENTS` для принудительного указания конкретных входных данных, включить явные ссылки на файл схемы через синтаксис @ для контроля контекста и добавить frontmatter description с предупреждением о деструктивных операциях.
- B) Добавить `argument-hint` в frontmatter для запроса обязательных параметров, использовать `context: fork` для изоляции выполнения и ограничить `allowed-tools` операциями записи файлов. **[ВЕРНО]**
- C) Разделить на отдельные скиллы `/migration-create` и `/migration-apply`, добавить инструкции в каждый SKILL.md для запроса имени миграции, если оно не указано, и использовать разные области `allowed-tools` для каждого скилла.
- D) Включить инструкции валидации в SKILL.md скилла, направляющие Claude проверять, что `$ARGUMENTS` содержит допустимое имя, добавить промпты для игнорирования предыдущего контекста разговора и перечислить запрещённые операции, которых Claude должен избегать.

Почему B: Этот подход корректно использует три различные функции конфигурации скиллов для решения каждой проблемы: `argument-hint` в frontmatter показывает ожидаемые параметры при автодополнении (решает проблему отсутствующих аргументов), `context: fork` изолирует выполнение в контексте субагента, отдельном от истории разговора (предотвращает утечку контекста из предыдущих разговоров), а `allowed-tools` ограничивает доступ к инструментам только операциями записи файлов (предотвращает деструктивные действия).

Вопрос 40 (Сценарий: Генерация кода с Claude Code)

Ситуация: В вашей кодовой базе есть различные области с разными соглашениями по кодированию: React-компоненты используют функциональный стиль с хуками, обработчики API используют async/await с определённой обработкой ошибок, а модели базы данных следуют паттерну репозитория. Тестовые файлы распределены по всей кодовой базе рядом с тестируемым кодом (например, Button.test.tsx рядом с Button.tsx), и вы хотите, чтобы все тесты следовали одним и тем же соглашениям независимо от расположения.

Какой наиболее поддерживаемый способ обеспечить автоматическое применение Claude правильных соглашений при генерации кода?

- A) Объединить все соглашения в корневом файле CLAUDE.md под заголовками для каждой области, полагаясь на способность Claude определить, какая секция применима.
- B) Создать скиллы в `.claude/skills/` для каждого типа кода, включающие соответствующие соглашения в файлах SKILL.md.
- C) Разместить отдельный файл CLAUDE.md в каждом подкаталоге, содержащий соглашения для данной области.
- D) Создать файлы правил в `.claude/rules/` с YAML frontmatter, указывающим glob-паттерны для условного применения соглашений на основе путей к файлам. **[ВЕРНО]**

Почему D: Использование файлов правил в `.claude/rules/` с YAML frontmatter и glob-паттернами (например, `**/*.test.tsx`, `src/api/**/*.ts`) позволяет автоматически и детерминированно применять соглашения на основе путей к файлам, независимо от их расположения в структуре каталогов. Это наиболее поддерживаемый подход, поскольку он обрабатывает сквозные задачи, такие как тестовые файлы, распределённые по всей кодовой базе, без дублирования или ручного вмешательства.

Вопрос 41 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы хотите создать пользовательскую слеш-команду `/review`, запускающую стандартный чек-лист ревью кода вашей команды. Эта команда должна быть доступна каждому разработчику при клонировании или получении обновлений репозитория.

Где следует создать файл этой команды?

- A) В `~/.claude/commands/` в домашнем каталоге каждого разработчика.
- B) В каталоге `.claude/commands/` в репозитории проекта. **[ВЕРНО]**
- C) В файле `.claude/config.json` с массивом commands.
- D) В файле CLAUDE.md в корне проекта.

Почему B: Размещение пользовательских слеш-команд в каталоге `.claude/commands/` внутри репозитория проекта является правильным, поскольку эти файлы находятся под контролем версий и автоматически доступны каждому разработчику, который клонирует или получает обновления

репозитория. Это предназначенное место для пользовательских команд проектного уровня в Claude Code.

Вопрос 42 (Сценарий: Генерация кода с Claude Code)

Ситуация: Файл CLAUDE.md вашей команды вырос до более чем 500 строк, смешивая соглашения TypeScript, рекомендации по тестированию, паттерны API и процедуры деплоя. Разработчикам трудно находить и обновлять нужные секции.

Какой подход поддерживает Claude Code для организации инструкций проектного уровня в сфокусированные тематические модули?

- A) Определить файл `.claude/config.yaml`, сопоставляющий паттерны файлов с конкретными секциями внутри CLAUDE.md.
- B) Создать отдельные markdown-файлы в `.claude/rules/`, каждый из которых охватывает одну тему (например, `testing.md`, `api-conventions.md`). **[ВЕРНО]**
- C) Разделить инструкции на файлы README.md в соответствующих подкаталогах, которые Claude автоматически загружает как инструкции.
- D) Создать несколько файлов с именем CLAUDE.md на разных уровнях дерева каталогов, каждый из которых переопределяет инструкции родительского.

Почему B: Claude Code поддерживает каталог `.claude/rules/`, в котором можно создавать отдельные markdown-файлы для тематических рекомендаций (например, `testing.md`, `api-conventions.md`), позволяя командам организовывать большие наборы инструкций в сфокусированные, легко поддерживаемые модули.

Вопрос 43 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы создаёте пользовательский скилл `/explore-alternatives`, который ваша команда использует для мозгового штурма и оценки различных подходов к реализации перед выбором одного из них. Однако разработчики сообщают, что после запуска этого скилла последующие ответы Claude находятся под влиянием обсуждения вариантов — иногда Claude ссылается на отвергнутые подходы или сохраняет исследовательский контекст, который мешает реальной работе по реализации.

Как наиболее эффективно сконфигурировать этот скилл?

- A) Использовать префикс `!` в скилле для выполнения логики исследования как подпроцесса bash.
- B) Добавить `context: fork` в frontmatter скилла. **[ВЕРНО]**
- C) Разделить скилл на два отдельных — `/explore-start` и `/explore-end` — для обозначения границ, когда исследовательский контекст должен быть отброшен.
- D) Создать скилл в `~/.claude/skills/` вместо `.claude/skills/`.

Почему B: Опция `context: fork` в frontmatter запускает скилл в изолированном контексте субагента, поэтому обсуждение вариантов не засоряет историю основного разговора. Это предотвращает влияние отвергнутых подходов и исследовательского контекста на последующую работу по реализации.

Вопрос 44 (Сценарий: Генерация кода с Claude Code)

Ситуация: Ваша команда хочет добавить MCP-сервер GitHub для поиска PR и проверки статуса CI через Claude Code. Каждый из шести разработчиков имеет собственный персональный токен доступа GitHub. Вы хотите обеспечить единообразие инструментария в команде без коммита учётных данных в систему контроля версий.

Какой подход к конфигурации наиболее эффективен?

- A) Каждому разработчику настроить сервер в пользовательской области с помощью `claude mcp add --scope user`.
- B) Создать обёртку MCP-сервера, которая читает токены из файла `.env` и проксирует запросы к API GitHub, затем добавить эту обёртку в проектный `.mcp.json`.
- C) Добавить сервер в проектный `.mcp.json` с подстановкой переменных окружения (`${GITHUB_TOKEN}`) для аутентификации и задокументировать требуемую переменную окружения в README проекта. **[ВЕРНО]**
- D) Настроить сервер в проектной области с токеном-заглушкой, а затем поручить разработчикам переопределить его в своей локальной конфигурации.

Почему C: Использование проектного `.mcp.json` с подстановкой переменных окружения (`${GITHUB_TOKEN}`) — это идиоматический подход: он обеспечивает единый, версионируемый источник истины для конфигурации MCP команды, позволяя каждому разработчику предоставлять свои собственные учётные данные через переменные окружения. Документирование требуемой переменной в README обеспечивает простоту онбординга без коммита секретов.

Вопрос 45 (Сценарий: Генерация кода с Claude Code)

Ситуация: Вы добавляете обёртки обработки ошибок для внешних вызовов API в кодовой базе из 120 файлов. Задача состоит из трёх фаз: (1) обнаружение всех мест и паттернов вызовов API, (2) совместное проектирование подхода к обработке ошибок и (3) консистентная реализация обёрток. Во время Фазы 1 Claude генерирует объёмный вывод, перечисляя сотни мест вызовов с контекстом. Ваше контекстное окно быстро заполняется ещё до завершения обнаружения.

Какой подход наиболее эффективен для завершения задачи с сохранением консистентности реализации?

- A) Использовать субагент Explore для Фазы 1, чтобы изолировать объёмный вывод и вернуть сводку, затем продолжить Фазы 2–3 в основном разговоре. **[ВЕРНО]**
- B) Продолжить все фазы в основном разговоре, периодически используя `/compact` для уменьшения использования контекста по мере продвижения по файлам.

- C) Переключиться в headless-режим с `--continue`, передавая явные сводки контекста между пакетными вызовами для поддержания непрерывности.
- D) Определить паттерн обработки ошибок в CLAUDE.md, затем обрабатывать файлы пакетами в нескольких сессиях, полагаясь на общий файл памяти для консистентности.

Почему A: Использование субагента Explore для Фазы 1 идеально, поскольку он изолирует объёмный вывод обнаружения в отдельном контексте, возвращая в основной разговор только краткую сводку. Это сохраняет контекстное окно основного разговора для фаз совместного проектирования и консистентной реализации, где сохранённый контекст наиболее ценен.

Сценарий: Агент поддержки клиентов

Вопрос 46 (Сценарий: Агент поддержки клиентов)

Ситуация: При тестировании вы замечаете, что агент часто вызывает `get_customer`, когда пользователи спрашивают о статусе заказа, хотя `lookup_order` был бы более уместен. Что следует проверить в первую очередь для решения этой проблемы?

Что следует проверить в первую очередь?

- A) Реализовать классификатор предварительной обработки, определяющий запросы о заказах и направляющий их напрямую к `lookup_order`.
- B) Сократить количество доступных агенту инструментов для упрощения выбора.
- C) Добавить few-shot примеры, покрывающие все возможные паттерны запросов о заказах, в системный промпт.
- D) Проверить описания инструментов, чтобы убедиться, что они чётко разграничивают назначение каждого инструмента. **[ВЕРНО]**

Почему D: Описания инструментов — это основной вход, который модель использует для принятия решения о вызове инструмента. Когда агент постоянно выбирает неправильный инструмент, первый диагностический шаг — проверить, чётко ли описания инструментов разграничивают назначение каждого инструмента и указывают, когда каждый из них следует использовать.

Вопрос 47 (Сценарий: Агент поддержки клиентов)

Ситуация: Ваш агент обрабатывает запросы с одной проблемой с точностью 94% (например, «Мне нужен возврат за заказ #1234»). Однако когда клиенты включают несколько проблем в одно сообщение (например, «Мне нужен возврат за заказ #1234 и также хочу обновить адрес доставки для заказа #5678»), точность выбора инструментов падает до 58%. Агент обычно решает только одну проблему или путает параметры между запросами. Какой подход наиболее эффективно повысит надёжность обработки запросов с несколькими проблемами?

Какой подход наиболее эффективен?

- A) Реализовать слой предварительной обработки, использующий отдельный вызов модели для декомпозиции сообщений с несколькими проблемами на отдельные запросы, обработки каждого независимо и объединения результатов.
- B) Объединить связанные инструменты в меньшее количество универсальных инструментов.
- C) Добавить few-shot примеры в промпт, демонстрирующие корректное рассуждение и последовательность инструментов для запросов с несколькими проблемами. **[ВЕРНО]**
- D) Реализовать валидацию ответов, обнаруживающую неполные ответы и автоматически повторно запрашивающую агента для решения пропущенных проблем.

Почему C: Добавление few-shot примеров, демонстрирующих корректное рассуждение и последовательность вызова инструментов для запросов с несколькими проблемами, является наиболее эффективным подходом, поскольку агент уже хорошо справляется с отдельными проблемами при точности 94% — ему просто нужно руководство по паттерну обработки нескольких проблем в одном сообщении. Это низкозатратная, проверенная техника, напрямую устраняющая первопричину неспособности агента декомпозировать и корректно маршрутизировать параметры между несколькими запросами.

Вопрос 48 (Сценарий: Агент поддержки клиентов)

Ситуация: Логи продакшена показывают, что для простых запросов типа «возврат за заказ #1234» ваш агент решает проблему за 3–4 вызова инструментов с 91% успешностью. Однако для сложных запросов типа «мне выставили счёт дважды, моя скидка не применилась, и я хочу отменить» агент в среднем выполняет 12+ вызовов инструментов с только 54% успешностью — часто исследуя проблемы последовательно и собирая избыточные данные о клиенте для каждой из них. Какое изменение наиболее эффективно улучшит обработку сложных запросов?

Какое изменение наиболее эффективно?

- A) Добавить явные контрольные точки верификации между этапами, требуя от агента фиксировать прогресс после решения каждой проблемы перед переходом к следующей.
- B) Сократить количество доступных инструментов, объединив `get_customer`, `lookup_order` и инструменты, связанные с биллингом, в единый инструмент `investigate_issue`.
- C) Декомпозировать запрос на отдельные проблемы, затем исследовать каждую параллельно с использованием общего контекста клиента перед синтезом решения. **[ВЕРНО]**
- D) Добавить few-shot примеры, демонстрирующие идеальные последовательности вызовов инструментов для различных сценариев многоаспектных биллинговых ситуаций, в системный промпт.

Почему C: Декомпозиция запроса на отдельные проблемы и их параллельное исследование с общим контекстом клиента напрямую устраняет обе ключевые проблемы: исключает избыточное получение данных за счёт повторного использования контекста между проблемами и сокращает общее количество вызовов инструментов за счёт параллелизации исследований перед синтезом единого решения.

Вопрос 49 (Сценарий: Агент поддержки клиентов)

Ситуация: Ваш агент достигает 55% решения с первого контакта, что значительно ниже целевых 80%. Логи показывают, что он эскалирует простые случаи (стандартные замены повреждённых товаров с фотоподтверждением), при этом пытаясь автономно обрабатывать сложные ситуации, требующие исключений из политики. Как наиболее эффективно улучшить калибровку эскалации?

Как наиболее эффективно улучшить калибровку?

- A) Заставить агента самостоятельно оценивать уверенность по шкале 1–10 перед каждым ответом и автоматически маршрутизировать запросы людям, когда уверенность падает ниже порога.
- B) Развернуть отдельную модель-классификатор, обученную на исторических тикетах, для прогнозирования, какие запросы нуждаются в эскалации, до начала обработки основным агентом.
- C) Добавить явные критерии эскалации в системный промпт с few-shot примерами, демонстрирующими, когда эскалировать, а когда решать автономно. **[ВЕРНО]**
- D) Реализовать анализ тональности для определения уровня фрустрации клиента и автоматической эскалации при превышении порога негативной тональности.

Почему C: Добавление явных критериев эскалации с few-shot примерами напрямую устраняет первопричину — нечёткие границы принятия решений между простыми и сложными случаями. Это наиболее пропорциональное и эффективное первое вмешательство, обучающее агента точно определять, когда эскалировать, а когда решать автономно, без необходимости дополнительной инфраструктуры.

Вопрос 50 (Сценарий: Агент поддержки клиентов)

Ситуация: После вызова `get_customer` и `lookup_order` агент получил все доступные системные данные, но сталкивается с неопределённостью. Какая ситуация представляет наиболее обоснованный триггер для вызова `escalate_to_human`?

Какая ситуация наиболее обоснованна для эскалации?

- A) Клиент хочет отменить заказ, отправленный вчера, с доставкой завтра. Агент должен эскалировать, потому что клиент может передумать после получения посылки.
- B) Клиент утверждает, что не получил заказ, но отслеживание показывает, что он был доставлен и подписан по его адресу три дня назад. Агент должен эскалировать, потому что предоставление противоречащих доказательств может навредить отношениям с клиентом.
- C) Клиент запрашивает сопоставление цен с конкурентом. Ваши политики допускают корректировки при снижении цен на вашем сайте в течение 14 дней, но ничего не говорят о ценах конкурентов. Агент должен эскалировать для интерпретации политики. **[ВЕРНО]**
- D) Сообщение клиента содержит и вопрос по биллингу, и возврат товара. Агент должен эскалировать, чтобы человек мог координировать обработку обеих проблем в одном взаимодействии.

Почему C: Это представляет реальный пробел в политике, где правила компании покрывают снижение цен на собственном сайте, но ничего не говорят о сопоставлении цен конкурентов, что означает, что агент не может выдумать политику и должен эскалировать для человеческого суждения о том, как интерпретировать или расширить существующие правила.

Вопрос 51 (Сценарий: Агент поддержки клиентов)

Ситуация: Данные продакшена показывают, что в 12% случаев ваш агент пропускает `get_customer` и сразу вызывает `lookup_order`, используя только указанное клиентом имя, что иногда приводит к ошибочной идентификации аккаунтов и некорректным возвратам. Какое изменение наиболее эффективно устранит эту проблему надёжности?

Какое изменение наиболее эффективно?

- A) Добавить few-shot примеры, показывающие, что агент всегда вызывает `get_customer` первым, даже когда клиенты предоставляют детали заказа добровольно.
- B) Реализовать классификатор маршрутизации, анализирующий каждый запрос и включающий только подмножество инструментов, подходящих для данного типа запроса.
- C) Добавить программное предусловие, блокирующее вызовы `lookup_order` и `process_refund` до тех пор, пока `get_customer` не вернёт верифицированный идентификатор клиента. **[ВЕРНО]**
- D) Усилить системный промпт, указав, что верификация клиента через `get_customer` обязательна перед любыми операциями с заказами.

Почему C: Добавление программного предусловия, блокирующего нижестоящие инструменты до возврата верифицированного идентификатора клиента от `get_customer`, обеспечивает детерминистическую гарантию соблюдения требуемой последовательности. Это наиболее эффективный подход, поскольку исключает саму возможность пропуска верификации агентом, независимо от поведения LLM.

Вопрос 52 (Сценарий: Агент поддержки клиентов)

Ситуация: Метрики продакшена показывают, что при решении сложных случаев с биллинговыми спорами или возвратами нескольких заказов оценки удовлетворённости клиентов на 15% ниже, чем для простых случаев — даже когда решение технически корректно. Анализ первопричин выявляет, что агент предоставляет точные решения, но непоследовательно объясняет обоснование: иногда упуская релевантные детали политики, иногда пропуская информацию о сроках или следующих шагах. Конкретные пробелы в контексте варьируются от случая к случаю. Вы хотите улучшить качество решений без добавления человеческого контроля. Какой подход наиболее эффективен?

Какой подход наиболее эффективен?

- A) Добавить этап самокритики, на котором агент оценивает черновик ответа на полноту — убеждаясь, что он решает проблему клиента, включает релевантный контекст и предвосхищает дополнительные вопросы. **[ВЕРНО]**

- B) Добавить этап подтверждения, на котором агент спрашивает «Это полностью решает вашу проблему?» перед закрытием, позволяя клиентам запросить дополнительную информацию при необходимости.
- C) Повысить уровень модели с Haiku до Sonnet для сложных случаев, маршрутизируя на основе определённой сложности случая.
- D) Реализовать few-shot примеры в системном промпте, показывающие полные объяснения решений для пяти распространённых типов сложных случаев, демонстрирующие, как включать контекст политики, сроки и следующие шаги.

Почему A: Этап самокритики (паттерн оценщик-оптимизатор) напрямую устраняет первопричину непоследовательной полноты объяснений, заставляя агента оценивать собственный черновик по конкретным критериям — таким как контекст политики, сроки и следующие шаги — перед его представлением. Это выявляет пробелы, специфичные для каждого случая, которые варьируются в разных сложных сценариях, без необходимости человеческого контроля.

Вопрос 53 (Сценарий: Агент поддержки клиентов)

Ситуация: Метрики продакшена показывают, что ваш агент в среднем выполняет 4+ API-цикла на решение. Анализ выявляет, что Claude часто запрашивает `get_customer` и `lookup_order` в отдельных последовательных ходах, даже когда оба нужны изначально. Как наиболее эффективно сократить количество циклов?

Как наиболее эффективно сократить количество циклов?

- A) Реализовать спекулятивное выполнение, автоматически вызывающее вероятно необходимые инструменты параллельно с любым запрошенным инструментом, возвращая все результаты независимо от того, что было запрошено.
- B) Увеличить `max_tokens`, чтобы дать Claude больше пространства для планирования и естественного объединения запросов инструментов.
- C) Создать составные инструменты типа `get_customer_with_orders`, объединяющие распространённые комбинации поиска в единые вызовы.
- D) Инструктировать Claude в промпте объединять запросы инструментов за один ход и возвращать все результаты вместе до следующего API-вызова. **[ВЕРНО]**

Почему D: Инструктирование Claude объединять связанные запросы инструментов за один ход и возврат всех результатов вместе до следующего API-вызова использует нативную способность Claude запрашивать несколько инструментов одновременно. Это наиболее эффективный подход, поскольку напрямую устраняет паттерн последовательных вызовов с минимальными архитектурными изменениями.

Вопрос 54 (Сценарий: Агент поддержки клиентов)

Ситуация: Ваш агент поддержки использует прогрессивную суммаризацию — при достижении контекстом 70% ёмкости более старые ходы суммируются, а недавние остаются дословно. Логи продакшена выявляют паттерн: клиенты ссылаются на конкретные суммы («скидка 15%, о которой я

говорил»), но агент отвечает с некорректными значениями. Исследование показывает, что эти детали были упомянуты 20+ ходов назад и были сконденсированы в расплывчатые резюме типа «обсуждалось акционное ценообразование». Какое исправление наиболее эффективно?

Какое исправление наиболее эффективно?

- A) Увеличить порог суммаризации с 70% до 85% ёмкости, чтобы у разговоров было больше места до срабатывания суммаризации.
- B) Хранить полную историю разговора во внешнем хранилище и реализовать поиск по ней, когда агент обнаруживает ссылочные фразы типа «как я упоминал».
- C) Извлекать транзакционные факты (суммы, даты, номера заказов) в постоянный блок «факты по делу», включаемый в каждый промпт вне суммаризированной истории. **[ВЕРНО]**
- D) Пересмотреть промпт суммаризации, чтобы явно сохранять все числовые значения, проценты, даты и ожидания, озвученные клиентом, дословно.

Почему C: Извлечение транзакционных фактов (суммы, даты, номера заказов) в постоянный блок «факты по делу» устраняет первопричину: суммаризация по своей природе теряет точные детали. Сохранение критической информации в структурированном блоке вне суммаризированной истории гарантирует, что эти факты надёжно доступны в каждом промпте, независимо от того, сколько ходов было суммаризировано.

Вопрос 55 (Сценарий: Агент поддержки клиентов)

Ситуация: Ваш инструмент `get_customer` возвращает все совпадения при поиске по имени. В настоящее время Claude выбирает клиента с самым недавним заказом при множественных результатах, но данные продакшена показывают, что это приводит к работе с неправильным аккаунтом клиента в 15% случаев множественных совпадений. Как решить эту проблему?

Как решить эту проблему?

- A) Реализовать систему оценки уверенности, которая действует автоматически при уверенности выше 85% и запрашивает уточнение при уверенности ниже этого порога.
- B) Инструктировать Claude запрашивать дополнительный идентификатор (email, телефон или номер заказа) при множественных результатах `get_customer`, прежде чем предпринимать какие-либо действия, специфичные для клиента. **[ВЕРНО]**
- C) Модифицировать `get_customer` для возврата только одного наиболее вероятного совпадения на основе алгоритма ранжирования, упрощая решение Claude за счёт устранения неоднозначных результатов.
- D) Добавить few-shot примеры, показывающие Claude, как использовать контекст разговора (упомянутые продукты, даты) для определения правильного клиента без запроса уточнений.

Почему B: Запрос у пользователя дополнительного идентификатора (email, телефон или номер заказа) — наиболее надёжный способ устранить неоднозначность множественных совпадений, поскольку пользователь обладает определённым знанием о собственной личности. Один дополнительный ход разговора — небольшая цена за устранение 15% ошибок, вызванных неправильным выбором клиента.

Вопрос 56 (Сценарий: Агент поддержки клиентов)

Ситуация: Логи продакшена выявляют устойчивый паттерн: когда клиенты включают слово «аккаунт» в сообщения (например, «Я хочу проверить мой аккаунт по заказу, который я сделал вчера»), агент вызывает `get_customer` первым в 78% случаев. Когда клиенты формулируют аналогичные запросы без «аккаунта» (например, «Я хочу проверить заказ, который я сделал вчера»), он вызывает `lookup_order` первым в 93% случаев. Описания инструментов написаны хорошо и однозначны. Какова наиболее вероятная первопричина этого расхождения?

Какова наиболее вероятная первопричина?

- A) Системный промпт содержит инструкции, чувствительные к ключевым словам, которые направляют поведение на основе терминов вроде «аккаунт», создавая непреднамеренные паттерны выбора инструментов. **[ВЕРНО]**
- B) Базовое обучение модели создаёт ассоциации между терминологией «аккаунт» и операциями, связанными с клиентом, которые переопределяют описания инструментов.
- C) Модели требуется больше обучающих данных по мультиконцептным сообщениям, и её следует дообучить на примерах, содержащих и терминологию аккаунта, и заказа.
- D) Описания инструментов нуждаются в дополнительных негативных примерах, указывающих, когда НЕ использовать каждый инструмент, для предотвращения этой путаницы, вызванной ключевыми словами.

Почему A: Это наиболее вероятная первопричина, поскольку систематический, вызванный ключевым словом паттерн (78% vs 93%) убедительно указывает на явную логику маршрутизации в системном промпте, реагирующую на слово «аккаунт» и направляющую агента к инструментам, связанным с клиентом. Поскольку описания инструментов уже хорошо написаны и однозначны, расхождение указывает на инструкции уровня промпта, создающие непреднамеренное поведенческое управление.

Вопрос 57 (Сценарий: Агент поддержки клиентов)

Ситуация: Логи продакшена показывают, что агент часто вызывает `get_customer`, когда пользователи спрашивают о заказах (например, «проверить мой заказ #12345»), вместо вызова `lookup_order`. Оба инструмента имеют минимальные описания («Получает информацию о клиенте» / «Получает детали заказа») и принимают похожие форматы идентификаторов. Каков наиболее эффективный первый шаг для улучшения надёжности выбора инструментов?

Каков наиболее эффективный первый шаг?

- A) Реализовать слой маршрутизации, анализирующий пользовательский ввод перед каждым ходом и предварительно выбирающий подходящий инструмент на основе обнаруженных ключевых слов и паттернов идентификаторов.
- B) Объединить оба инструмента в единый `lookup_entity`, принимающий любой идентификатор и внутренне определяющий, к какому бэкенду обращаться.
- C) Добавить few-shot примеры в системный промпт, демонстрирующие корректные паттерны выбора инструментов, с 5–8 примерами маршрутизации запросов о заказах к `lookup_order`.

- D) Расширить описание каждого инструмента, включив форматы входных данных, примеры запросов, граничные случаи и границы, объясняющие, когда использовать его, а когда — аналогичные инструменты. **[ВЕРНО]**

Почему D: Расширение описаний инструментов с включением форматов входных данных, примеров запросов, граничных случаев и границ напрямую устраняет первопричину — минимальные описания, не позволяющие LLM различать похожие инструменты. Это низкозатратный, высокоэффективный первый шаг, улучшающий основной механизм, который LLM использует для выбора инструментов.

Вопрос 58 (Сценарий: Агент поддержки клиентов)

Ситуация: Вы реализуете агентный цикл для вашего агента поддержки. После каждого API-вызова к Claude вам нужно определить — продолжить цикл (выполнить запрошенные инструменты и вызвать Claude снова) или остановиться (представить финальный ответ клиенту). Что определяет это решение?

Что определяет это решение?

- A) Проверить поле `stop_reason` в ответе Claude — продолжать, если оно равно `tool_use`, и останавливаться, если равно `end_turn`. **[ВЕРНО]**
- B) Парсить текст ответа Claude на наличие фраз типа «Я завершил» или «Могу ли я ещё чем-то помочь?» — эти сигналы естественного языка указывают на завершение задачи.
- C) Установить максимальное количество итераций (например, 10 вызовов) и останавливаться по достижении, независимо от того, указывает ли Claude на необходимость дальнейшей работы.
- D) Проверить, содержит ли ответ текстовый контент от ассистента — если Claude сгенерировал пояснительный текст, цикл должен завершиться.

Почему A: Поле `stop_reason` — это явный структурированный сигнал Claude для управления циклом: `tool_use` указывает, что Claude хочет выполнить инструмент и получить результаты обратно, а `end_turn` указывает, что Claude завершил свой ответ и цикл должен быть завершён.

Вопрос 59 (Сценарий: Агент поддержки клиентов)

Ситуация: Логи продакшена показывают, что агент неправильно интерпретирует данные от ваших MCP-инструментов: Unix-временные метки от `get_customer`, даты ISO 8601 от `lookup_order` и числовые коды статусов (1=в ожидании, 2=отправлен). Некоторые инструменты — это сторонние MCP-серверы, которые вы не можете модифицировать. Какой подход к нормализации форматов данных наиболее удобен в обслуживании?

Какой подход наиболее удобен в обслуживании?

- A) Использовать хук PostToolUse для перехвата результатов инструментов и применения преобразований форматирования перед обработкой агентом. **[ВЕРНО]**
- B) Модифицировать контролируемые инструменты для возврата человеко-читаемых форматов; создать обёртки для сторонних инструментов.

- C) Создать инструмент `normalize_data`, который агент вызывает после каждого получения данных для преобразования значений.
- D) Добавить подробную документацию форматов в системный промпт, объясняющую соглашения о данных каждого инструмента.

Почему A: Использование хука PostToolUse обеспечивает централизованную, детерминистическую точку для перехвата и нормализации всех выходных данных инструментов — включая данные от сторонних MCP-серверов — до их обработки агентом. Это наиболее удобный в обслуживании подход, поскольку применяет преобразования единообразно через код, а не полагаясь на интерпретацию LLM или поведение агента.

Вопрос 60 (Сценарий: Агент поддержки клиентов)

Ситуация: Логи продакшена показывают, что агент иногда выбирает `get_customer`, когда `lookup_order` был бы более уместен, особенно для неоднозначных запросов типа «Мне нужна помощь с моей недавней покупкой». Вы решаете добавить few-shot примеры в системный промпт для улучшения выбора инструментов. Какой подход наиболее эффективно решит эту проблему?

Какой подход наиболее эффективно решит эту проблему?

- A) Добавить явные рекомендации «используйте, когда» и «не используйте, когда» в описание каждого инструмента, покрывающие неоднозначные случаи.
- B) Добавить примеры, сгруппированные по инструменту — все сценарии `get_customer` вместе, затем все сценарии `lookup_order`.
- C) Добавить 4–6 примеров, нацеленных на неоднозначные сценарии, каждый с обоснованием, почему один инструмент был выбран вместо правдоподобных альтернатив. **[ВЕРНО]**
- D) Добавить 10–15 примеров ясных, однозначных запросов, демонстрирующих корректный выбор инструмента для типичных сценариев каждого инструмента.

Почему C: Нацеливание few-shot примеров на конкретные неоднозначные сценарии, где происходят ошибки, с явным обоснованием того, почему один инструмент предпочтительнее другого, напрямую обучает модель процессу сравнительного принятия решений, необходимому для граничных случаев. Этот подход наиболее эффективен, поскольку разобранные примеры с обоснованием работают лучше декларативных правил для нюансированного выбора инструментов.

Вопрос 61 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Ваш инструмент `remove_team_member` использует параметр `dry_run: boolean` для предварительного просмотра последствий перед выполнением. Мониторинг продакшена показывает, что агент обходит шаг предпросмотра и вызывает инструмент напрямую с `dry_run=false`. Вам нужно гарантировать, что каждому удалению предшествует предпросмотр, подтверждённый пользователем.

Какой подход наиболее надёжен?

- A) Добавить серверную валидацию, разрешающую `dry_run=false` только если вызов с `dry_run=true` с идентичными параметрами был выполнен в течение последних 60 секунд.
- B) Пометить инструмент как требующий подтверждения и настроить слой оркестрации для запроса одобрения пользователя перед перенаправлением вызовов к помеченным инструментам.
- C) Добавить подробные инструкции и few-shot примеры в описание инструмента, требующие от агента всегда вызывать сначала с `dry_run=true` и ждать подтверждения пользователя.
- D) Заменить двумя инструментами: `preview_remove_member` возвращает детали воздействия и одноразовый токен подтверждения; `execute_remove_member` требует этот токен, связывая выполнение с предпросмотром. **[ВЕРНО]**

Почему D: Подход со связыванием токеном делает архитектурно невозможным выполнение без предварительного предпросмотра — инструмент выполнения буквально требует токен, который может сгенерировать только инструмент предпросмотра. Это единственный подход, применяющий ограничение на уровне кода, а не полагающийся на соблюдение инструкций LLM (C), временные эвристики (A) или инфраструктуру оркестрации (B).

Вопрос 62 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Мониторинг продакшена показывает, что ваш инструмент `search_catalog` даёт сбой в 12% случаев: 8% — тайм-ауты сети, которые успешно завершаются при повторной попытке, и 4% — синтаксические ошибки запроса, которые никогда не удаются. В настоящее время оба типа ошибок возвращаются одинаково, вызывая бесполезные повторные попытки.

Как вам следует изменить обработку ошибок инструмента?

- A) Добавить few-shot примеры в системный промпт, демонстрирующие, как различать сетевые ошибки и синтаксические.
- B) Применить логику повторных попыток с экспоненциальной задержкой ко всем ошибкам равномерно.
- C) Реализовать автоматические повторные попытки с задержкой для тайм-аутов сети внутри инструмента; немедленно возвращать синтаксические ошибки с деталями валидации параметров. **[ВЕРНО]**
- D) Возвращать все ошибки с булевым флагом `retryable` и деталями типа ошибки.

Почему C: Обработка повторных попыток на уровне инструмента для временных ошибок — правильный уровень абстракции. Инструмент обладает исчерпывающими знаниями о типе ошибки и может реализовать детерминированную логику повторных попыток, не полагаясь на агента для интерпретации флага (D) или следования инструкциям промпта (A). Равномерная задержка (B) тратит время на синтаксические ошибки, которые никогда не удадутся.

Вопрос 63 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: В ходе нескольких ходов обсуждения инвестиционной стратегии пользователь заявил «У меня очень низкая толерантность к риску», а затем «Я хочу максимизировать свою доходность». Теперь он спрашивает: «Во что мне следует инвестировать?»

Какой подход лучше всего обеспечивает соответствие рекомендации реальному приоритету пользователя?

- A) Обозначить противоречие и попросить пользователя уточнить, что важнее. **[ВЕРНО]**
- B) Предоставить отдельные рекомендации для обоих сценариев.
- C) Действовать исходя из наиболее недавно заявленного предпочтения.
- D) Рекомендовать сбалансированный портфель, не касаясь конфликта.

Почему A: Когда предпочтения пользователя прямо противоречат друг другу, обозначение конфликта и запрос на уточнение — единственный способ гарантировать соответствие рекомендации истинному намерению пользователя. Максимизация доходности и низкая толерантность к риску — принципиально несовместимые цели, требующие человеческого решения.

Вопрос 64 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи уточняют предпочтения плейлиста на протяжении нескольких ходов. Через два сообщения после того, как пользователь сказал «Я люблю джаз», Claude спрашивает «Какие жанры вам нравятся?»

Какова наиболее вероятная причина?

- A) Claude требует подключения к векторной базе данных для поддержания памяти диалога.
- B) Контекстное окно модели было превышено.
- C) API Claude требует параметр `session_id`.
- D) Ваше приложение не включает предыдущие сообщения в массив `messages`. **[ВЕРНО]**

Почему D: Claude не имеет серверной памяти — каждый вызов API не сохраняет состояния. Без включения полной истории диалога в массив `messages` каждого запроса Claude не знает о предыдущих ходах. Векторные базы данных (A) и `session_id` (C) не являются частью архитектуры Claude; переполнение контекстного окна (B) невозможно для обмена из двух сообщений.

Вопрос 65 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: После 40-минутной кулинарной сессии диалог достигает 78 000 токенов. История включает аллергии, масштабирование рецептов, уточнённые кулинарные термины и общее обсуждение. Вам нужно сократить токены, сохранив важную информацию.

Какой подход лучше всего балансирует между сохранением и сокращением токенов?

- A) Резюмировать всю историю диалога.
- B) Сохранить только самые последние 20 000 токенов.
- C) Извлечь критические структурированные данные (аллергии, количества, предпочтения), резюмировать общее обсуждение и сохранить недавние обмены дословно. **[ВЕРНО]**
- D) Хранить полный диалог во внешнем хранилище и извлекать релевантные части с помощью семантического поиска.

Почему C: Гибридный подход сохраняет наиболее ценную информацию с наименьшими затратами. Критические факты, такие как аллергии и количества в рецептах, извлекаются в компактный структурированный блок (предотвращая потерю точности, возникающую при резюмировании), общее обсуждение резюмируется, а недавние обмены сохраняются дословно для диалоговой связности. Варианты A и B рискуют потерять критическую диетическую информацию; D избыточен для одной кулинарной сессии.

Вопрос 66 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи сообщают, что в ходе длительных диалогов ассистент теряет нить ранних тем и предпочтений. Ваша текущая реализация хранит только последние 25 пар сообщений.

Какое решение наиболее эффективно?

- A) Гибридный подход: резюмировать более старые сообщения, сохраняя недавние дословно. **[ВЕРНО]**
- B) Векторный поиск по сходству по всей истории диалога.
- C) Увеличить окно до 50 пар сообщений.
- D) Резюмировать удалённые сообщения на каждом ходу и добавлять накопленное резюме в начало.

Почему A: Гибридный подход решает обе стороны проблемы: сохраняет точный недавний контекст (критически важный для диалоговой связности) и поддерживает сжатое представление более ранних предпочтений. Увеличение окна (C) просто откладывает ту же проблему. Векторный поиск (B) может упустить важный контекст, не похожий семантически на текущий запрос.

Вопрос 67 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи сообщают о росте задержек и затрат при диалогах, превышающих 50 ходов.

Какова основная причина?

- A) Вся история диалога включается в каждый запрос к API. **[ВЕРНО]**

- B) Модель генерирует всё более длинные ответы.
- C) Операции с базой данных замедляются по мере роста истории.
- D) Модель формирует внутренний профиль пользователя, требующий большей обработки.

Почему A: API Claude полностью не сохраняет состояния — каждый запрос должен включать полную историю диалога в массиве `messages`. По мере роста диалога каждый запрос несёт больше токенов, что напрямую увеличивает задержку обработки и стоимость. Модель не поддерживает никакого внутреннего состояния между вызовами (D — ложь).

Вопрос 68 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: После трёх месяцев еженедельных сессий история диалога вырастает до 85 000 токенов. Когда пользователь спрашивает «К чему мы пришли по теме изоляции?», ассистент даёт общие ответы вместо ссылок на предыдущие обсуждения.

Какой подход наиболее эффективен?

- A) Усечение скользящего окна.
- B) Прогрессивное резюмирование с фиксацией ключевых выводов.
- C) Семантические эмбеддинги с извлечением релевантных обменов. **[ВЕРНО]**
- D) Добавить структурированные XML-теги, маркирующие выводы обсуждения.

Почему C: Семантический поиск по истории диалога — единственный подход, масштабируемый на три месяца обсуждений при возможности извлечения конкретных релевантных обменов по запросу. Усечение скользящего окна (A) отбросит большую часть истории. Прогрессивное резюмирование (B) сжимает обсуждения в абстракции, теряя конкретные выводы, о которых спрашивают пользователи.

Вопрос 69 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: В ходе QA-тестирования Claude следует правилам системного промпта в первые 10–15 ходов, но последующие ответы отклоняются. Диалог всё ещё в пределах лимитов токенов.

Какое решение оптимально?

- A) Перенести поведенческие правила в первое сообщение пользователя.
- B) Начинать новый диалог после 20 ходов.
- C) Вставлять сообщения роли пользователя, подкрепляющие правила, в переломных точках диалога. **[ВЕРНО]**
- D) Использовать постобработку ответа для регенерации несоответствующих ответов.

Почему C: Периодическая инъекция поведенческих напоминаний напрямую противодействует дрейфу инструкций, восстанавливая ограничения через регулярные интервалы по мере накопления

истории. Перенос правил в первое сообщение пользователя (A) снижает их авторитет. Постобработка (D) корректирующая, а не превентивная, и добавляет значительную задержку.

Вопрос 70 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Ваш ИИ-тьютор имеет системный промпт из 2800 токенов, определяющий методологию обучения и правила адаптации. После 12 ходов ассистент начинает игнорировать уровни владения.

Какое исправление наиболее эффективно?

- A) Инъецировать напоминания каждые 4–5 ходов.
- B) Заменить подробные правила few-shot примерами, демонстрирующими адаптацию по уровню владения. **[ВЕРНО]**
- C) Поместить критические правила в конец системного промпта.
- D) Оценивать ответы и регенерировать при несовпадении уровня сложности.

Почему B: Системный промпт из 2800 токенов с декларативными правилами уязвим к дрейфу, поскольку абстрактные правила требуют от модели размышления над ними на каждом ходу. Замена подробных правил конкретными few-shot примерами, демонстрирующими правильную адаптацию по уровню владения, даёт модели чёткие поведенческие паттерны для следования — это надёжнее соблюдается на протяжении многих ходов, чем абстрактные инструкции.

Вопрос 71 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Ваш ассистент должен поддерживать энтузиастический тон, объяснять своё рассуждение и задавать уточняющие вопросы. Где следует определять эти поведенческие правила?

Где следует определять эти поведенческие правила?

- A) Добавляться перед каждым сообщением пользователя.
- B) В системном промпте. **[ВЕРНО]**
- C) В первом сообщении ассистента.
- D) В переменных окружения.

Почему B: Системный промпт специально предназначен для постоянных поведенческих ограничений и правил, применяемых на протяжении всего диалога. Добавление перед каждым сообщением пользователя (A) — избыточная нагрузка. Первое сообщение ассистента (C) ненадёжно, так как модель может отклоняться от своих собственных предыдущих высказываний. Переменные окружения (D) не влияют на поведение модели.

Вопрос 72 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи сообщают о повторяющихся вступлениях ответов, например «Конечно!» и «Рад помочь!»

Какой подход наиболее эффективен?

- A) Добавить частичное сообщение ассистента с прямым вступлением ответа. **[ВЕРНО]**
- B) Снизить настройку температуры.
- C) Постобрабатывать ответы для удаления приветствий.
- D) Добавить инструкции в системный промпт для избегания этих фраз.

Почему A: Предзаполнение ответа ассистента началом прямого ответа предотвращает шаблоны приветствий на уровне генерации — модель продолжает с предзаполнения, а не генерирует новые вступительные фразы. Инструкции системного промпта (D) могут помочь, но менее надёжны. Постобработка (C) — хрупкое обходное решение. Температура (B) управляет случайностью, а не конкретными фразовыми паттернами.

Вопрос 73 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Webhook уведомляет вашу систему об отправке посылки пользователя, пока тот активно общается в чате. Вы хотите, чтобы ассистент естественно включил это в следующий ответ.

Какой подход оптимален?

- A) Добавить статус отправки в системный промпт.
- B) Отправить немедленное синтетическое сообщение пользователя.
- C) Принудительно вызывать инструмент статуса на каждом ходу.
- D) Добавить обновление статуса как префикс к следующему сообщению пользователя. **[ВЕРНО]**

Почему D: Добавление обновления статуса как префикса к следующему сообщению пользователя вводит контекст реального времени на естественной границе диалога, не нарушая поток. Изменение системного промпта (A) требует перестройки сессии. Синтетическое сообщение пользователя (B) может нарушить естественный диалоговый поток. Принудительный вызов инструмента на каждом ходу (C) расточителен, когда события редки.

Вопрос 74 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи часто отправляют запросы вроде «Забронируй место для вечеринки». Ассистент задаёт 4+ уточняющих вопроса, что приводит к 35% отказов.

Какой подход лучше всего улучшает баланс?

- A) Действовать исходя из скрытых значений по умолчанию.
- B) Задать все уточняющие вопросы в одном составном сообщении.
- C) Явно обозначить допущения и действовать, приглашая к исправлениям. **[ВЕРНО]**
- D) Использовать структурированную форму приёма.

Почему C: Явное обозначение допущений и действие даёт пользователю немедленный и полезный ответ, сохраняя возможность исправить неверные допущения. Скрытые значения по умолчанию (A) оставляют пользователя в неведении о сделанных допущениях. Составной список вопросов (B) всё равно требует усилий от пользователя. Структурированная форма (D) добавляет больше трения, а не меньше.

Вопрос 75 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Ваш ассистент использует системный промпт с персоной подрядчика. Ранние ходы следуют правилам, но к ходу 7 ассистент даёт общие советы. Длина диалога всего 2500 токенов.

Какова наиболее вероятная причина?

- A) Системные промпты устанавливают только начальное поведение.
- B) Внимание модели ослабевает по мере накопления ходов.
- C) Накопленные ответы ассистента разбавляют влияние системного промпта. **[ВЕРНО]**
- D) Системный промпт отправляется только один раз.

Почему C: По мере накопления ответов ассистента в истории диалога доля текста, отражающего поведенческие ограничения системного промпта, уменьшается относительно растущего массива сгенерированного ассистентом контента. Модель всё больше подстраивается под паттерны своих предыдущих ответов, а не под инструкции системного промпта, усиливая дрейф даже при короткой длине в токенах.

Вопрос 76 (Сценарий: Архитектурные паттерны разговорного ИИ)

Ситуация: Пользователи задают расплывчатые запросы вроде «Можешь помочь с отчётом?» Ассистент отвечает несколькими вопросами (какой отчёт? какая помощь? какой срок?), что приводит к 40% отказов.

Какое решение оптимально?

- A) Делать разумные допущения, явно их обозначать и предлагать корректировки. **[ВЕРНО]**
- B) Классифицировать неоднозначность с помощью меньшей модели перед ответом.
- C) Использовать предопределённые интерпретации без обозначения допущений.
- D) Ограничить ассистента одним уточняющим вопросом за ход.

Почему A: Действие с разумными обозначенными допущениями полностью устраняет диалог с вопросами-ответами, оставляя пользователя информированным и в управлении. Молчаливые предопределённые интерпретации (C) вводят пользователей в замешательство, когда ответ не совпадает с их намерением. Лимит в один вопрос (D) всё равно требует ходов туда-обратно. Классификационная модель меньшего размера (B) добавляет задержку и инфраструктурную сложность без решения основной UX-проблемы.

Практические упражнения

Упражнение 1: Мультиинструментальный агент с логикой эскалации

Цель: Проектирование агентного цикла с интеграцией инструментов, структурированной обработкой ошибок и эскалацией.

Шаги:

1. Определите 3-4 MCP-инструмента с детальными описаниями (включите два похожих для проверки выбора)
2. Реализуйте агентный цикл с проверкой `stop_reason` (`"tool_use"` / `"end_turn"`)
3. Добавьте структурированные ответы об ошибках: `errorCategory`, `isRetryable`, описание
4. Реализуйте хук-перехватчик для блокировки операций выше порога с перенаправлением на эскалацию
5. Тестируйте с мульти-аспектными запросами

Домены: 1 (Агентная архитектура), 2 (Инструменты и MCP), 5 (Контекст и надёжность)

Упражнение 2: Настройка Claude Code для командной разработки

Цель: Настройка CLAUDE.md, пользовательских команд, path-specific правил и MCP-серверов.

Шаги:

1. Создайте проектный CLAUDE.md с универсальными стандартами
2. Файлы `.claude/rules/` с YAML frontmatter для разных областей кода (`paths: ["src/api/**/*"]`, `paths: ["**/*.test.*"]`)
3. Проектный навык в `.claude/skills/` с `context: fork` и `allowed-tools`
4. MCP-сервер в `.mcp.json` с переменными окружения + личный в `~/.claude.json`
5. Тестирование: режим планирования vs прямое выполнение на задачах разной сложности

Домены: 3 (Конфигурация Claude Code), 2 (Инструменты и MCP)

Упражнение 3: Пайплайн извлечения структурированных данных

Цель: JSON-схемы, tool_use для структурированного вывода, циклы валидации/повтора, пакетная обработка.

Шаги:

1. Определите инструмент извлечения с JSON-схемой (required/optional поля, enum с "other", nullable поля)
2. Цикл валидации: при ошибке -- повтор с документом, ошибочным извлечением и конкретной ошибкой
3. Few-shot примеры для документов с разной структурой
4. Пакетная обработка через Message Batches API: 100 документов, обработка сбоев по `custom_id`
5. Маршрутизация на человека: оценки уверенности на уровне полей, анализ по типу документа

Домены: 4 (Промпт-инженерия), 5 (Контекст и надёжность)

Упражнение 4: Проектирование и отладка мультиагентного исследовательского пайплайна

Цель: Оркестрация субагентов, передача контекста, распространение ошибок, синтез с отслеживанием источников.

Шаги:

1. Координатор с 2+ субагентами (`allowedTools` включает `"Task"`, контекст передаётся явно в промпте)
2. Параллельное выполнение субагентов через несколько вызовов `Task` в одном ответе
3. Структурированный вывод субагентов: утверждение, цитата, URL источника, дата публикации
4. Симуляция таймаута субагента: структурированный контекст ошибки координатору, продолжение с частичными результатами
5. Тестирование с конфликтующими данными: сохранение обоих значений с атрибуцией, разделение на подтверждённые и спорные находки

Домены: 1 (Агентная архитектура), 2 (Инструменты и MCP), 5 (Контекст и надёжность)

Приложение: Технологии и концепции

Технология	Ключевые аспекты
Claude Agent SDK	AgentDefinition, агентные циклы, `stop_reason`, хуки (PostToolUse), порождение субагентов через Task, `allowedTools`

Model Context Protocol (MCP)	MCP-серверы, инструменты, ресурсы, `isError`, описания инструментов, `.mcp.json`, переменные окружения
Claude Code	Иерархия CLAUDE.md, `.claude/rules/` с glob-паттернами, `.claude/commands/`, `.claude/skills/` с SKILL.md, режим планирования, `/compact`, `--resume`, `fork_session`
Claude Code CLI	`-p` / `--print` для неинтерактивного режима, `--output-format json`, `--json-schema`
Claude API	`tool_use` с JSON-схемами, `tool_choice` ("auto"/"any"/принудительный), `stop_reason`, `max_tokens`, системные промпты
Message Batches API	Экономия 50%, окно до 24 часов, `custom_id`, нет multi-turn tool calling
JSON Schema	Required vs optional, nullable поля, enum типы, `"other"` + detail, строгий режим
Pydantic	Валидация схем, семантические ошибки, циклы валидации/повтора
Встроенные инструменты	Read, Write, Edit, Bash, Grep, Glob -- назначение и критерии выбора
Few-shot промптинг	Целевые примеры для неоднозначных ситуаций, обобщение на новые паттерны
Prompt chaining	Последовательная декомпозиция в фокусированные проходы
Контекстное окно	Бюджеты токенов, прогрессивная суммаризация, "потеря в середине", scratchpad-файлы
Управление сессиями	Возобновление, `fork_session`, именованные сессии, изоляция контекста
Калибровка уверенности	Оценки на уровне полей, калибровка на размеченных наборах, стратифицированная выборка

Темы вне области экзамена (Out-of-Scope)

Следующие смежные темы **НЕ** будут на экзамене:

- Файн-тюнинг моделей Claude или обучение кастомных моделей
- Аутентификация Claude API, биллинг или управление аккаунтами
- Детальная реализация на конкретных языках программирования или фреймворках (кроме необходимого для конфигурации инструментов и схем)
- Развёртывание или хостинг MCP-серверов (инфраструктура, сети, оркестрация контейнеров)
- Внутренняя архитектура Claude, процесс обучения или веса модели
- Constitutional AI, RLHF или методологии обучения безопасности
- Эмбеддинг-модели или детали реализации векторных баз данных

- Computer use (автоматизация браузера, взаимодействие с рабочим столом)
- Возможности анализа изображений (Vision)
- Streaming API или server-sent events
- Rate limiting, квоты или расчёты стоимости API
- OAuth, ротация API-ключей или детали протоколов аутентификации
- Конфигурации конкретных облачных провайдеров (AWS, GCP, Azure)
- Бенчмарки производительности или метрики сравнения моделей
- Детали реализации кэширования промптов (помимо знания о существовании)
- Алгоритмы подсчёта токенов или специфика токенизации

Рекомендации по подготовке

1. **Создайте агента с Claude Agent SDK** -- полный агентный цикл с вызовом инструментов, обработкой ошибок и управлением сессиями. Практикуйте субагентов и передачу контекста.
2. **Настройте Claude Code для реального проекта** -- CLAUDE.md с иерархией, path-specific правила в `.claude/rules/`, навыки с `context: fork` и `allowed-tools`, интеграция MCP-сервера.
3. **Спроектируйте и протестируйте MCP-инструменты** -- описания, разделяющие похожие инструменты, структурированные ошибки с категориями и флагами повтора, тестирование на неоднозначных запросах.
4. **Постройте пайплайн извлечения данных** -- `tool_use` с JSON-схемами, циклы валидации/повтора, optional/nullable поля, пакетная обработка через Message Batches API.
5. **Практикуйте промпт-инженерию** -- few-shot примеры для неоднозначных сценариев, явные критерии ревью, мульти-проходные архитектуры для больших ревью кода.
6. **Изучите паттерны управления контекстом** -- извлечение фактов из многословных выводов, scratchpad-файлы, делегирование субагентам для управления контекстными лимитами.
7. **Разберитесь в эскалации и human-in-the-loop** -- когда эскалировать (пробелы в политике, запрос клиента, невозможность прогресса), рабочие процессы с маршрутизацией на основе уверенности.
8. **Пройдите пробный экзамен** перед реальным. Он использует те же сценарии и формат.