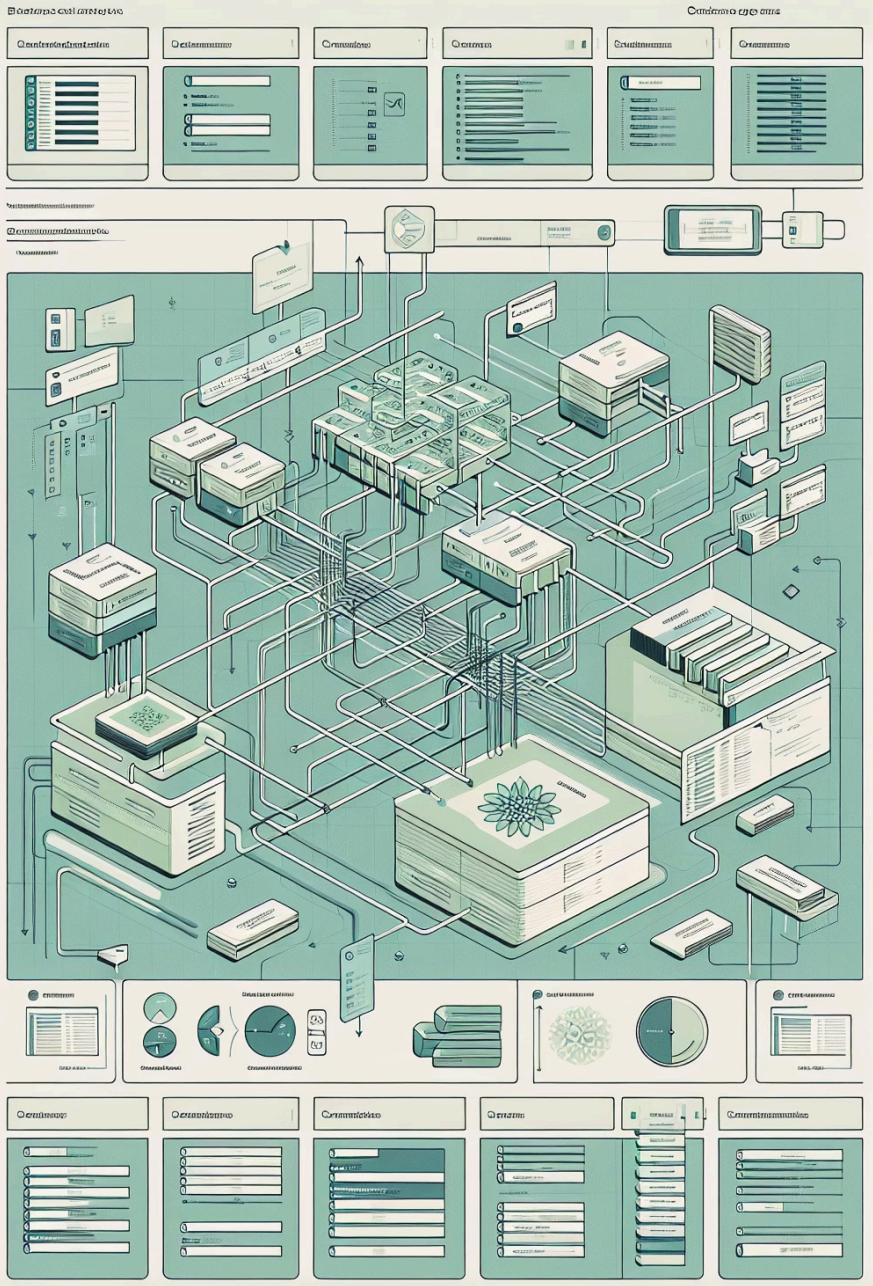
Архитектура и системное проектирование

Создание масштабируемых и надежных программных систем в современной облачной эпохе



Что такое архитектура программного обеспечения

Внутренняя структура

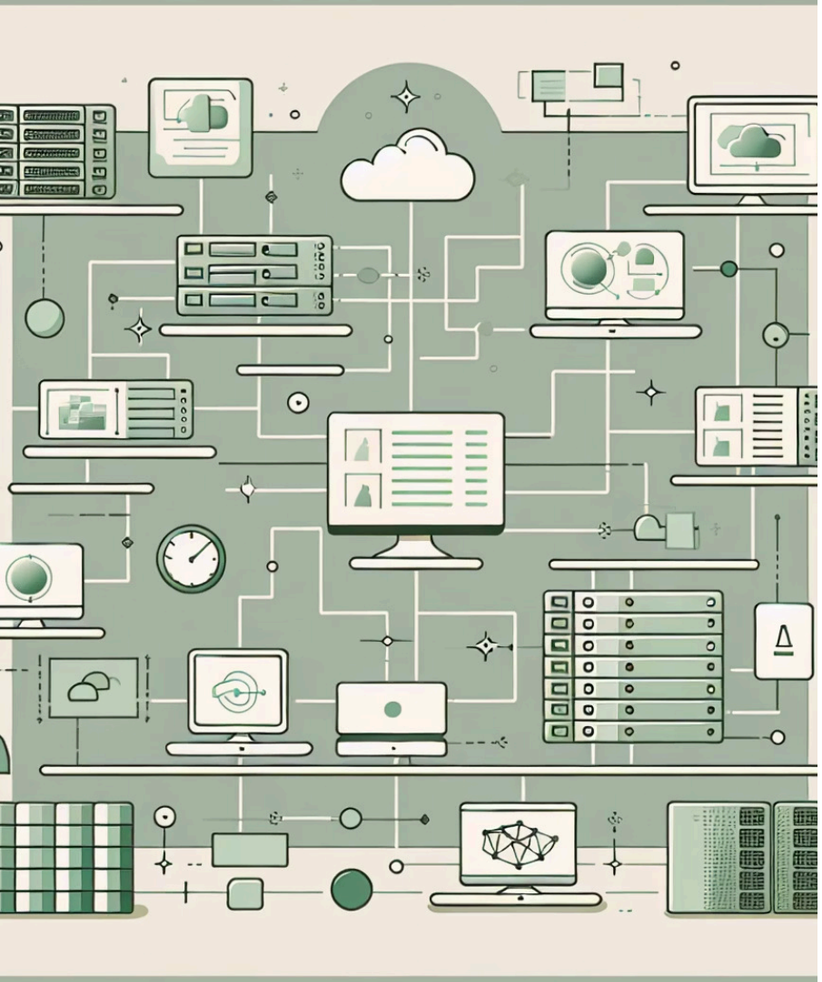
Архитектура ПО определяет, как взаимодействуют компоненты приложения — внутреннюю структуру, потоки данных и отношения между компонентами.

- Взаимодействие компонентов
- Потоки данных
- Отношения между модулями

Способность к развитию

Правильная архитектура определяет способность программного обеспечения развиваться и масштабироваться в долгосрочной перспективе.

- Масштабируемость системы
- Гибкость изменений
- Поддерживаемость кода



Системное проектирование: более широкий взгляд

Инфраструктура

Планирование и организация технической инфраструктуры для поддержки всей системы

Хранение данных

Стратегии управления данными, их хранения и обработки в масштабе

Взаимодействие сервисов

Организация коммуникации между различными компонентами системы

Развертывание

Стратегии и процессы доставки программного обеспечения в продакшн

Системное проектирование работает в более широкой области, охватывая всю технологическую экосистему и эксплуатационные вопросы.

Аналогия со строительством

Архитектура ПО

Как проектирование внутренней планировки здания

- Расположение комнат
- Внутренние коммуникации
- Функциональные зоны
- Потоки движения

"Как должно быть структурировано это приложение?"

Системное проектирование

Как планирование всего комплекса зданий

- Транспортные связи
- Инженерные системы
- Зонирование территории
- Внешние коммуникации

"Как это приложение должно вписаться в техническую экосистему?"

Критическая важность обеих дисциплин

Взаимное дополнение

Плохие решения в архитектуре и системном проектировании усиливают негативные эффекты друг друга

Синергия качества

Хорошие решения в обеих областях создают мощную синергию для успеха проекта

Основа масштабирования

Обе дисциплины критически важны для успеха современного программного обеспечения

Неизбежность архитектурных решений

> Каждая система имеет и архитектуру, и системный дизайн — независимо от того, планируете ли вы их намеренно или позволяете возникнуть случайно.

Осознанный выбор

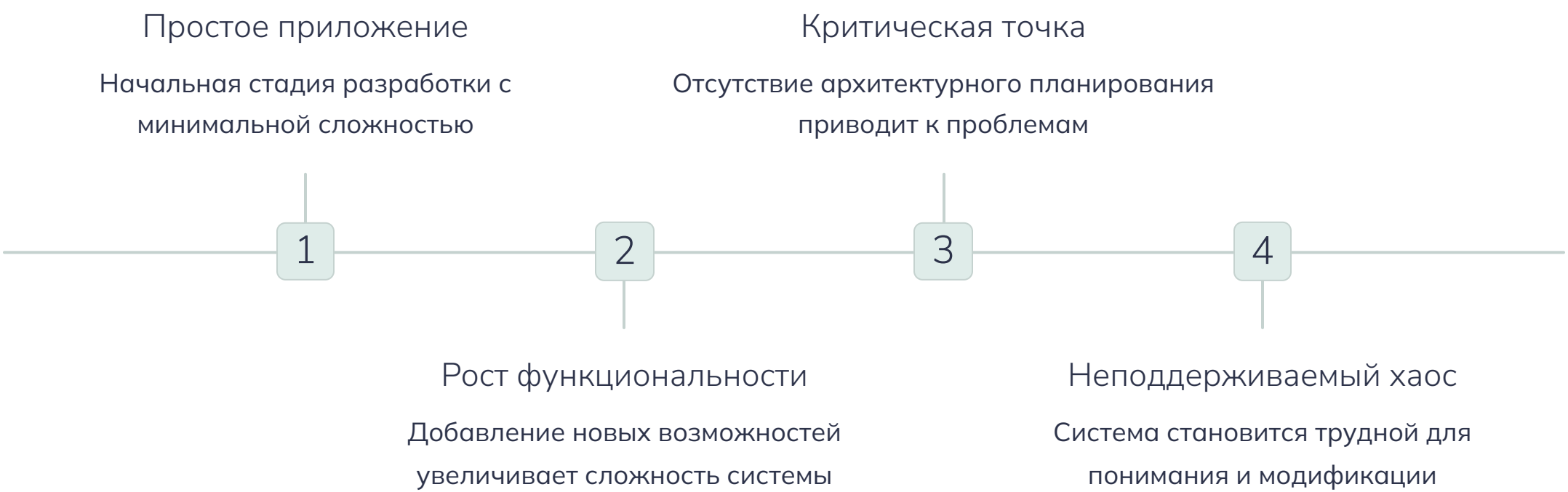
Вы активно проектируете архитектуру и системный дизайн, основываясь на требованиях и целях

Случайное развитие

Архитектура формируется хаотично через тактические решения без стратегического планирования

Выбор бинарный: либо вы выбираете свою архитектуру и системное проектирование, либо они выбирают вас.

Эволюция сложности систем



То, что начинается как простое приложение, может быстро превратиться в неподдерживаемый хаос, когда архитектурные решения откладываются или игнорируются.

Последствия плохого управления архитектурой

Снижение скорости разработки

Команда тратит больше времени на обход архитектурных ограничений, чем на создание новых функций

Рост количества багов

Плохая структура кода приводит к непредсказуемым взаимодействиям и ошибкам

Рискованное развертывание

Отсутствие четких границ делает релизы непредсказуемыми и опасными

Рост эксплуатационных расходов

Неэффективная архитектура требует больше ресурсов для поддержания работоспособности

Хрупкая инфраструктура

Система становится уязвимой к отказам и трудной для масштабирования

Результаты плохой архитектуры

Большой ком грязи (Big Ball of Mud)

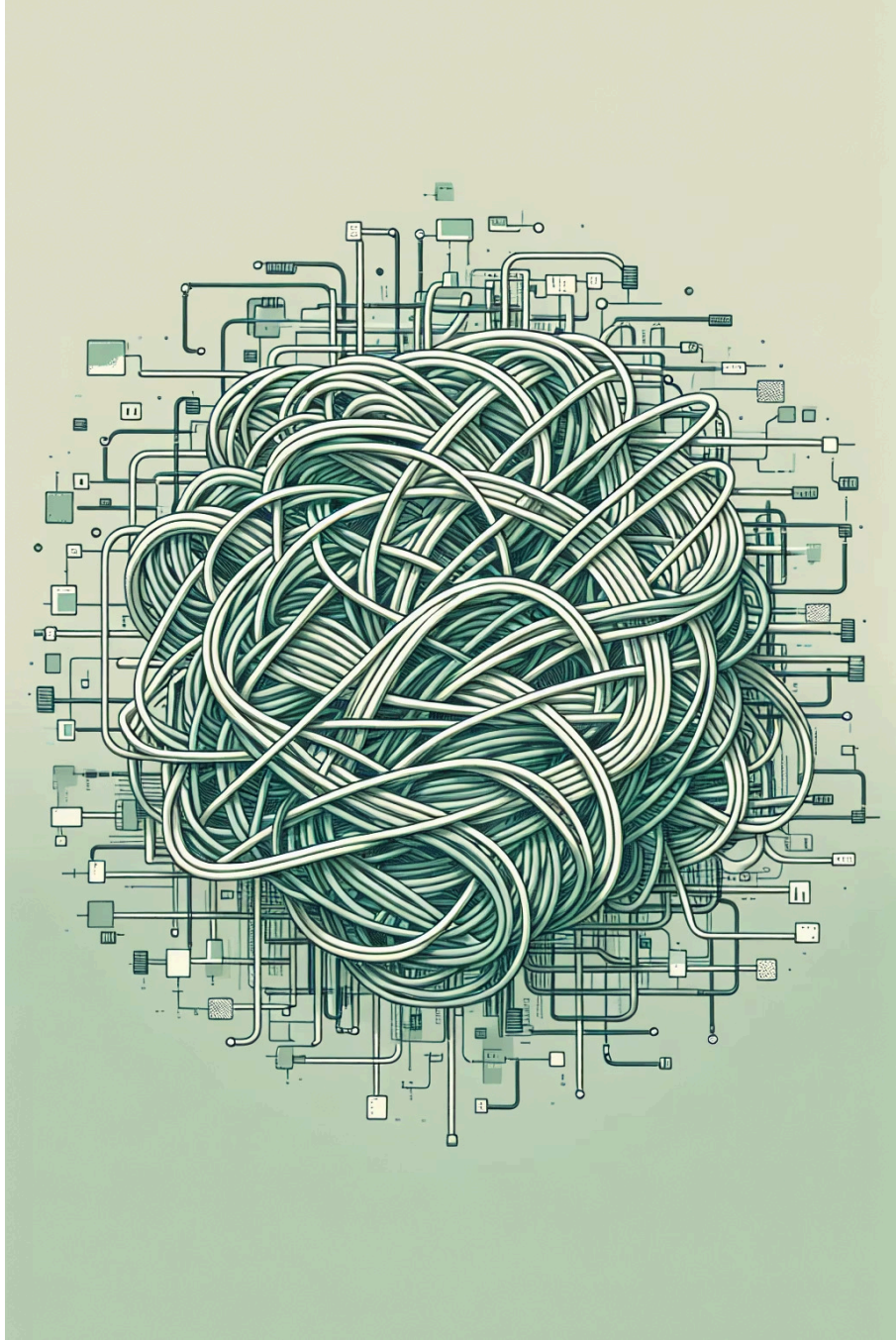
Системы лишены четких границ, с компонентами, жестко связанными между собой

- Отсутствие модульности
- Жесткая связанность
- Размытые обязанности

Проблемы инфраструктуры

Плохой системный дизайн создает операционные проблемы

- Неспособность к масштабированию
- Узкие места в базах данных
- Хрупкие процессы развертывания



Распределенные монолиты

Общие базы данных

Сервисы связаны через общие хранилища данных

Синхронная коммуникация

Жесткие зависимости через синхронные вызовы

Координированные релизы

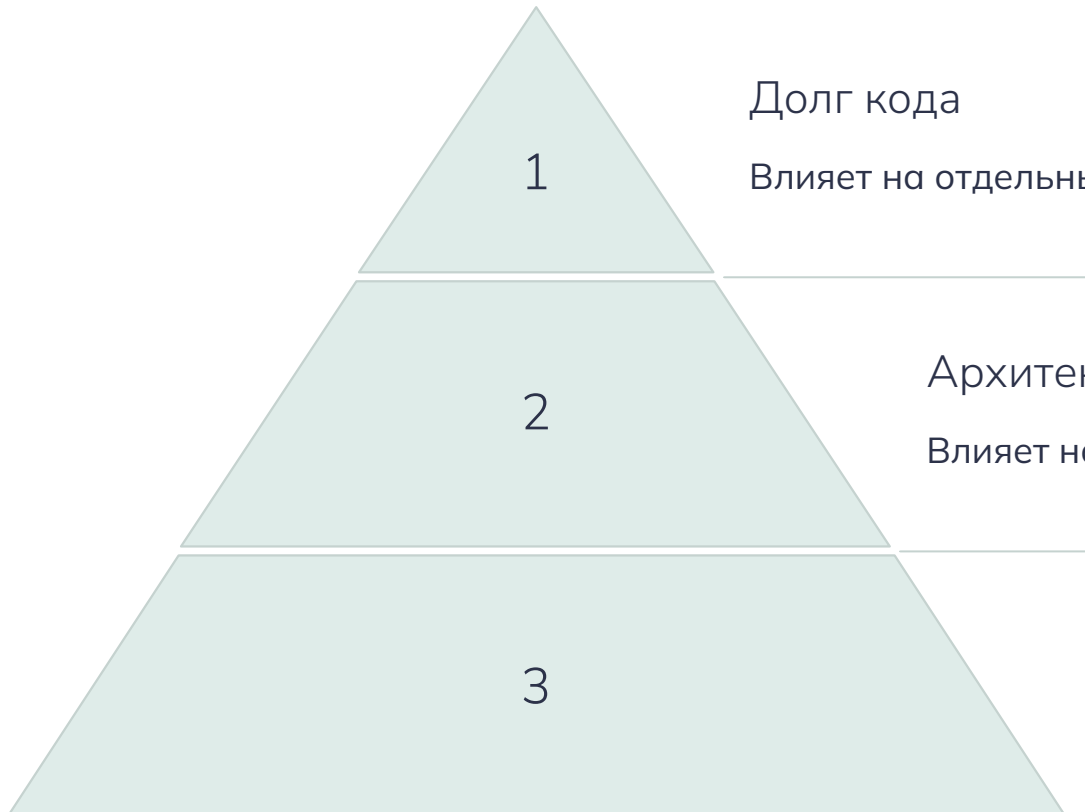
Необходимость синхронного развертывания

Хрупкие зависимости

Изменения в одном сервисе влияют на другие

В распределенных системах плохая архитектура часто приводит к "распределенным монолитам" — системам, которые кажутся независимыми, но на самом деле жестко связаны.

Технический долг: многоуровневая проблема



Долг кода

Влияет на отдельные компоненты

Архитектурный долг

Влияет на способность системы к эволюции

Долг системного проектирования

Влияет на эксплуатационную эффективность, масштабируемость и надежность

Технический долг накапливается, когда архитектурные и системные компромиссы принимаются без осознанных trade-off'ов. Команды тратят больше времени на обход ограничений, чем на работу над функциями.

Облачная эпоха: новый уровень сложности

Беспрецедентные возможности

Облачные среды предлагают невиданные ранее возможности для масштабируемости, надежности и глобального охвата

Новые сложности

Управление распределенным состоянием, сетевые разделения, eventual consistency требуют нового подхода

Оркестрация сервисов

Автоматизация инфраструктуры и оптимизация затрат становятся критически важными

Современные архитектурные парадигмы

Микросервисы

Разделение приложения на независимые, слабо связанные сервисы

Serverless Computing

Выполнение кода без управления серверной инфраструктурой

Event-driven архитектуры

Асинхронное взаимодействие через события и сообщения

Переход к этим парадигмам сделал архитектурное мышление и системное проектирование необходимыми для любых серьезных усилий по разработке ПО.

Ключевые вопросы современных систем

01

Отказоустойчивость

Как мы справляемся с отказами сервисов и обеспечиваем непрерывность работы?

02

Согласованность данных

Как мы поддерживаем согласованность данных между распределенными сервисами?

03

Развертывание без простоев

Как мы развертываем обновления без прерывания обслуживания пользователей?

04

Автомасштабирование

Как мы автоматически масштабируем инфраструктуру в зависимости от спроса?

05

Управление затратами

Как мы управляем затратами по нескольким облачным сервисам?

06

Безопасность

Как мы обеспечиваем безопасность в распределенных системах?

Это не академические вопросы — они напрямую влияют на пользовательский опыт, эксплуатационные расходы и бизнес-успех.

Путь вперед: стратегические принципы

Понимайте требования

Глубокое понимание бизнес-требований и технических ограничений

Делайте осознанные trade-off'ы

Каждое архитектурное решение имеет компромиссы — важно их понимать

Проектируйте для изменений

Системы должны быть готовы к эволюции и адаптации

Обеспечьте совместную эволюцию

Структура ПО и инфраструктура должны развиваться синхронно

Контекст имеет огромное значение — те же паттерны, которые работают для стартапа, могут быть неподходящими для предприятия со строгими требованиями соответствия.

> Понимание тем, раскрываемых в курсе, имеет прямое отношение к успешному прохождению технических собеседований, поскольку он охватывает ключевые аспекты проектирования облачных SaaS-систем — от декомпозиции и компромиссов до масштабируемости, отказоустойчивости и безопасности — и учит мыслить архитектурно, аргументировать решения и адаптироваться к различным контекстам и ограничениям.

Как показывает данная статья, системное проектирование — это не просто набор техник или шаблонов, а способ стратегического мышления о структуре, взаимодействиях и эволюции всей технологической экосистемы. На собеседованиях кандидаты часто сталкиваются с вопросами, которые выходят за рамки конкретных технологий и требуют понимания trade-off'ов: как система масштабируется, как она справляется с отказами, как устроено хранение и передача данных, как обеспечивается безопасность, — и всё это в условиях реальных ограничений по времени, бюджету и команде. Знания, полученные в рамках курса, дают не только язык описания решений, но и практические ориентиры для аргументации своего выбора — чего и ожидают интервьюеры. Без этого системное мышление заменяется хаотичным подбором решений по принципу "слышал где-то", тогда как собеседование требует целостного, обоснованного подхода — именно такую цель мы ставили при создании данного курса.