# Адаптивный плейбук

**Автор: Денис Матаков, Технический директор** https://mtkv.ru

---

## 🧭 Что это за документ

Этот плейбук описывает основы **адаптивной модели разработки**, ориентированной на:

- поставку ценности,
- автономность команд,
- устойчивую и предсказуемую разработку.

Он основан на лучших практиках **Scrum**, **Scaled Agile Frameworks** и **реальном опыте внедрения** в цифровых продуктах.

Документ предназначен для всех участников процесса: от разработчиков и аналитиков до продукт-менеджеров и руководства. Его цель — **создать единый язык и мышление**, позволяющее эффективно создавать и развивать продукты в быстро меняющейся среде.

---

## 🎯 Зачем он нужен

- Установить **прозрачные и понятные процессы**
- Помочь **новым участникам быстро включиться**
- Синхронизировать **команды и руководство**
- Поддержать **самоорганизованные и устойчивые команды**
- Служить **основой для продуктового, инженерного и культурного роста**

---

## 📚 Структура плейбука

6. Инженерные практики
7. Метрики и прозрачность
8. Гильдии и развитие компетенций
9. Культура и лидерство
10. Роли в адаптивной модели
11. Процесс релизов и доставки
12. Переход от эпиков к ценности: сторителлинг сквозь стек

---

# 🔷 Раздел 1. Введение: Зачем нужен Scrum и адаптивная модель

## 📌 Цель

Объяснить, почему в современной разработке важно переходить от проектного подхода к **продуктовой модели**, построенной на:

- коротких итерациях,
- прозрачности процессов,
- регулярной обратной связи.

Цель — сформировать **общее понимание и язык** взаимодействия для всех участников команды.

---

## ❗ Проблемы традиционной модели

Традиционный (предиктивный) подход к разработке программного обеспечения приводит к:

- Долгим релизным циклам (2–6 месяцев)
- Высокой стоимости изменений на поздних этапах
- Множественным передачам задач между ролями → потеря контекста
- Неактуальности требований к моменту релиза
- Отсутствию общей ответственности за результат

---

## ⚠ Почему это больше не работает

- Рынок меняется быстрее, чем успевает писаться ТЗ

- Пользователи ждут постоянного улучшения продукта
- Конкуренты быстрее проверяют гипотезы
- Продукты терпят неудачу не из-за реализации, а из-за неверно выбранной проблемы

---

## ✅ Что даёт адаптивная модель (Scrum)

- Короткий цикл поставки — рабочий результат каждые 2 недели
- Быстрая обратная связь — видно, что работает, а что нет
- Автономные команды — меньше зависимостей, выше скорость
- Фокус на ценности — строим то, что действительно нужно
- Прозрачность — все участники понимают текущее состояние

---

## 📊 Сравнение подходов

| Аспект | Проектный подход | Адаптивная модель (Scrum) |
|---|---|---|
| Цель | В срок, в бюджет | Достижение бизнес-результатов |
| Структура | Функциональная / компонентная | Кросс-функциональные команды |
| Планирование | На 6–12 месяцев | Каждые 2 недели (инкрементально) |
| Обратная связь | Задержанная | Регулярная и быстрая |
| Финансирование | Проекты | Команды |
| Управление | Контроль сверху | Поддержка и развитие команд |

---

## 🧭 Наше намерение

Внедрение Scrum и адаптивной модели помогает:

- Повысить гибкость и адаптивность к изменениям
- Сократить путь от идеи до ценности
- Развить самостоятельность команды и её фокус на результат

# 🔷 Раздел 2. Операционная модель на уровне команды

## 📌 Цель

Показать, как функционирует Scrum-команда: какие роли в ней есть, какие встречи она проводит, какими артефактами оперирует и как отслеживает прогресс.

## 🧩 Ключевые принципы

- Команда — самоорганизующаяся единица, способная планировать и поставлять ценность самостоятельно
- Все роли и процессы подчинены одной цели — предсказуемой и частой поставке рабочего продукта
- Scrum — не рекомендации, а цельная система, где важна согласованность всех элементов

## 👥 Роли в команде

- **Product Owner** — определяет, *что* нужно делать. Владеет приоритетами и взаимодействует с бизнесом
- **Scrum Master** — обеспечивает, *как* команда работает. Помогает соблюдать Scrum, устраняет препятствия
- **Разработчики** — создают рабочий продукт. Несут ответственность за тестирование, качество и релиз
- **QA-инженеры** — отвечают за качество: тесты (ручные и автоматизированные), стратегия тестирования

## 📆 Встречи (командный ритм)

- **Планирование спринта** — раз в 2 недели. Определение цели и задач спринта
- **Дейли (Stand-up)** — ежедневно. Синхронизация, выявление блокеров
- **Ретроспектива** — раз в спринт. Рефлексия и улучшение взаимодействия и процессов
- **Обзор спринта** — раз в спринт. Демонстрация инкремента и сбор обратной связи
- **Refinement** — 1–2 раза в неделю. Подготовка и уточнение задач на будущее

*Все встречи фокусированы на ценности, а не на отчётности.*

## 📦 Артефакты

- **Product Backlog** — приоритезированный и прозрачный список задач от PO
- **Sprint Backlog** — задачи, выбранные командой для текущего спринта

- **Инкремент** — завершённый функционал, соответствующий Definition of Done
- **Scrum-доска** — визуальное отображение текущего состояния задач

## 📈 Метрики

Метрики — инструмент самоанализа, а не давления. Используются для ретроспектив и оценки устойчивости работы.

- **Velocity** — количество story points, стабильно закрываемых за спринт
- **Burndown Chart** — отслеживание прогресса выполнения задач
- **Issue Count / Rework** — число добавленных или изменённых задач в процессе
- **Список блокеров** — зафиксированные препятствия с назначенными ответственными

## 🔁 Цикл спринта (2 недели)

Refinement → Sprint Planning → Daily Scrum → Delivery (инкремент) → Sprint Review → Retrospective → снова Refinement

## 🧭 Роль команды в организации

В адаптивной модели именно команда является ядром системы. Она:

- Поставляет ценность конечному пользователю
- Несёт ответственность за результат
- Становится точкой опоры всей системы — процессы выстраиваются вокруг неё, а не наоборот

# 🔷 Раздел 3. Операционная модель на уровне компании

## 📌 Цель

Описать, как координируется работа между командами: кто отвечает за стратегию, продукт, поставку и архитектуру, как принимаются решения и обеспечивается синхронизация в масштабе всей программы.

## 🧩 Объединённая структура управления

- Каждая команда остаётся кросс-функциональной и автономной
- Программа (верхний уровень) служит командам, устраняет препятствия и обеспечивает фокус
- Финансирование направлено на команды, а не проекты

- Все решения опираются на метрики и обратную связь, а не на иерархию или интуицию

## 🤝 Ключевые роли

- **CEO / CPO** — стратегия продукта, roadmap, бизнес-метрики
- **CTO** — организация поставки, устранение блокеров, работа с талантами
- **Engineering Manager (EM)** — техническое качество, архитектура, техдолг, DevOps
- **Scrum Masters** — поддержка команд, развитие процессов, фасилитация синхронизации
- **Product Owners (POs)** — ведение командных бэклогов, детализация и приоритезация

## 📆 События уровня компании

- **Планирование компании** (раз в 2 недели) — синхронизация целей между командами и владельцами продуктов
- **Обзор инкрементов** (раз в 2 недели) — демонстрация результата, обратная связь, метрики
- **Ретроспектива программы** (раз в 2 недели) — анализ взаимодействия, процессов, зависимостей
- **Refinement бэклога программы** (еженедельно) — работа с эпиками и инициативами
- **Ежедневный дейлик компании** — синк по прогрессу и проблемам между ключевыми ролями (CEO, CTO, POs, EMs)

## 🔄 Управление зависимостями

- Используется подход **"моков и контрактов"** — не ждём, а работаем параллельно
- Межпродуктовые зависимости выявляются на этапе планирования и refinement
- Команды сохраняют автономность, но выравниваются по общей цели

## 📈 Метрики и инструменты

**Примеры метрик:**

- **Velocity** по командам
- **Стабильность релизов**
- **Количество и время устранения блокеров**
- **Уровень качества** — доля рефакторинга, покрытие тестами, количество инцидентов
- **Согласованность целей** — выполнение спринт-целей, соответствие roadmap

## 🔧 Процесс устранения блокеров

1. Команда фиксирует проблему (в Jira или на доске)
2. Scrum Master эскалирует её EM или CTO
3. Все блокеры публичны, у каждого есть владелец и срок устранения

## 📌 Фокус руководства

Руководство программы:

- Не управляет задачами напрямую
- Не занимается микроменеджментом
- Обеспечивает прозрачность, инфраструктуру, устранение зависимостей и развитие людей

# 🔷 Раздел 4. Сквозные продуктовые практики

## 📌 Цель

Обеспечить единый подход к управлению продуктом: как формируются и проверяются гипотезы, как выстраивается стратегия, как приоритизируются и описываются задачи, и на чём основаны продуктовые решения.

## 🧭 1. Vision и стратегия продукта

Каждый продукт должен иметь:

- Чёткое **видение** (1–2 предложения): для кого продукт и какую ценность он создаёт
- **Цели и метрики**: бизнес-результаты — выручка, вовлечённость, NPS
- **Roadmap** по кварталам — уровень инициатив и эпиков

Пример vision:

> Сделать доставку товаров из Китая понятной и управляемой для клиентов/

## 🔬 2. Работа с гипотезами (HADI-цикл)

Каждая инициатива проходит короткий итеративный цикл:

1. **Hypothesis** — «Если мы сделаем X, то произойдёт Y»
2. **Action** — минимальные действия для проверки
3. **Data** — сбор необходимых метрик
4. **Insight** — вывод, подтвердилась гипотеза или нет

Фокус — на быстрой проверке и измеримом результате.

## 🗺️ 3. User Story Mapping

Используем user story map для планирования:

- По горизонтали — путь пользователя (flow)
- По вертикали — приоритизация (от MVP до расширений)

Цель — не просто список задач, а понимание, что даёт **реальную ценность**.

## 🧩 4. Структура backlog

- **Инициатива** — гипотеза и бизнес-цель
- **Эпик** — крупная пользовательская задача
- **User Story** — конкретная ценность, реализуемая за 1–3 дня
- **Задача (sub-task)** — технические шаги к реализации

Refinement и декомпозиция — **не разовое действие**, а регулярный процесс.

## 🔎 5. Критерии готовности

**Definition of Ready (DoR)**:

- История написана от лица пользователя
- Прописаны критерии приёмки
- Оценена командой

**Definition of Done (DoD)**:

- Протестировано (ручное или автоматическое)
- Документировано (если нужно)
- Принято PO и готово к релизу

## 🧮 6. Приоритизация

**ICE** — для гипотез:
ICE = Impact × Confidence × Ease

- **Impact** — влияние на метрики (1–10)
- **Confidence** — уверенность в данных (0–1)
- **Ease** — сложность реализации (1–10)

**MoSCoW** — для командного backlog’а:

- **Must have** — без них релиз невозможен
- **Should have** — желательно, но можно отложить
- **Could have** — nice-to-have

- **Won’t have (this time)** — вне текущей итерации

**RICE** — для крупных инициатив:
RICE = (Reach × Impact × Confidence) / Effort

- **Reach** — охват (напр. 1000 пользователей в месяц)
- **Impact** — влияние на пользователя (от 0.25 до 3)
- **Confidence** — уверенность в данных (0–1)
- **Effort** — трудозатраты (в story points или человеко-неделях)

## 📊 7. Сквозная прозрачность

- Вся работа ведётся в одной системе (например, Jira)
- Эпики и инициативы связаны с roadmap
- Бизнес, аналитики и разработка работают в **общем контексте**
- Не допускается работа вне Product Backlog или без гипотезы

Сквозные продуктовые практики обеспечивают **согласованность, фокус на ценности и прозрачность от идеи до поставки**.

# 🔷 Раздел 5. Работа с backlog'ом и refinement-практики

## 📌 Цель

Установить общие правила и ритм работы с backlog’ом: как описываются задачи, как они готовятся к спринтам, кто за что отвечает, как проводится refinement и какие признаки «готовности» используются.

## 📦 Уровни детализации backlog’а

- **Product backlog** — инициативы, эпики, приоритеты на 1–3 месяца (от CEO / Product Owner)
- **Team backlog** — user stories на 1–2 спринта вперёд с критериями приёмки (от Product Owner)
- **Sprint backlog** — подзадачи, оценки, задачи с выполненным DoR (от команды разработки)

## 🔄 Как мы работаем с backlog’ом

- Регулярно обновляется, очищается и приоритизируется
- Каждый элемент привязан к метрике или гипотезе
- В него не попадают абстрактные хотелки или непроверенные идеи
- Задачи не попадают в спринт без refinement

## 🗒️ Каденция refinement

- Минимум 1 спринт вперёд проработанных задач — хорошая практика
- **Форматы:**
  - **Refinement-сессия** — 1–2 раза в неделю, участие PO, команды и Scrum Master
  - **Асинхронно** — на постоянной основе (PO, аналитик, дизайнер, архитектор)

## ✅ Definition of Ready (DoR)

User Story считается «готовой», если:

- Есть описание от лица пользователя
- Указаны критерии приёмки
- Есть оценка команды
- Устранены или зафиксированы зависимости
- Понятны UX / API аспекты (если применимо)

Если хотя бы один пункт не выполнен — история **не попадает в спринт**.

## ✔ Definition of Done (DoD)

User Story считается завершённой, если:

- Реализована и покрыта тестами (unit / интеграционные / e2e)
- Acceptance criteria выполнены
- Принята Product Owner’ом
- Вошла в инкремент или релиз
- Документация обновлена (если необходимо)

## 🧠 Кто за что отвечает

- **Product Owner** — формулирует требования, критерии приёмки, задаёт приоритеты
- **Команда** — уточняет, оценивает, выявляет блокеры, предлагает альтернативы
- **Scrum Master** — отвечает за процесс, фасилитирует сессии refinement

## 🧮 Оценка задач

- Используются **story points**, ориентируясь на скорость команды
- Метод оценки выбирает команда: Planning Poker, T-shirt sizing и др.
- Истории с оценкой более **13 SP** — разбиваются или исключаются из спринта

## 🛑 Почему это важно

- Хороший refinement = предсказуемость, спокойствие, фокус на реализации
- Плохой backlog = стресс, хаос, срыв целей
- Меньше времени уходит на «что делать», больше — на «как сделать»

# 🔷 Раздел 6. Инженерные практики

## 📌 Цель

Обеспечить техническую устойчивость, высокое качество кода и быструю поставку за счёт стандартизированных практик, автоматизации и культуры инженерной ответственности.

## ✅ Definition of Done (DoD)

Задача или User Story считается завершённой, если:

- Протестирована (юнит, интеграционные, e2e — в зависимости от уровня)
- Смёржена в основной релизный поток (main/trunk)
- Принята Product Owner’ом
- Попадает в сборку и доступна для релиза
- Не требует ручной доработки
- Задокументирована при необходимости (в коде, API, описании)

## 🧪 Автоматизация тестирования

- Юнит-тесты обязательны для новых модулей (в зависимости от языка)
- Интеграционные тесты — для ключевых сценариев
- E2E — для пользовательских флоу (особенно web и mobile)
- Тесты — часть DoD, а не отдельная задача
- Качество — коллективная ответственность. Нет роли "просто тестировщик"

## 🌲 Стратегия ветвления (branching)

- Основной поток: `main` или `develop` — зависит от модели
- Работа ведётся в feature-ветках
- Коммиты атомарные, сообщения оформлены по шаблонам (`feat:`, `fix:`, `refactor:`)

Перед слиянием:

- Pull Request
- Code Review от 1–2 человек
- Прохождение CI

## 🔁 Code Review

- Цель — не просто поиск ошибок, а обеспечение общего понимания и качества
- Каждый член команды регулярно делает ревью
- Стиль — доброжелательный и конструктивный

Типичные чек-листы:

- Читабельность
- Покрытие тестами
- Обработка ошибок и логирование
- Безопасность (если применимо)

## ⚙ CI/CD

- Единый pipeline (например, GitHub Actions, GitLab CI и др.)
- Включает: линтинг, сборку, тесты
- Статический анализ (если доступен)
- Автодеплой на stage / dev / preview
- Продакшн-деплой — автоматизированный, но управляемый

Быстрый CI/CD = быстрая обратная связь. Цель — приближение к trunk-based development.

## 🔐 Безопасность и качество

- Включён автоматический линтинг и сканирование уязвимостей
- Зависимости регулярно обновляются (Dependabot, Renovate)
- Централизованное логирование (например, Grafana stack, Sentry)
- Все изменения отслеживаются (audit trail, git blame, CI-метки)

## 🔁 Технический долг

- Фиксируется как технические задачи в backlog
- Приоритизируется наряду с обычными задачами
- В каждом спринте резервируется 10–20% ёмкости на устранение техдолга
- Регулярно проводятся архитектурные ревью и ревизии решений

## 🧭 Культура инженерного качества

- Код должен быть не только рабочим, но и читаемым
- Тесты — это инструмент уверенности, а не страховка
- Платформа и инфраструктура — полноценная часть продукта

- За качество отвечает вся команда, включая PO и дизайнеров

# 🔷 Раздел 7. Метрики и прозрачность

## 📌 Цель

Определить, какие метрики используются для оценки эффективности команд, продукта и процессов. Обеспечить прозрачность, ориентированную на **улучшение**, а не на контроль.

## 🎯 Общие принципы

- Метрики — это **инструмент обратной связи**, а не контроль сверху
- Не существует «одной цифры» — важно смотреть на **систему показателей**
- Метрики обсуждаются **на ретроспективах и при планировании**
- Главное — **действия по результатам**, а не само измерение

## 👥 Командные метрики

- **Velocity** — стабильность и прогнозируемость (в story points)
- **Burndown Chart** — прогресс по задачам спринта
- **Issue Count / Scope Change** — сколько задач добавлено или удалено в спринте
- **Время от идеи до релиза (Lead Time)**
- **% завершённых задач по плану** — отражает реализм оценок и фокус команды

## 🧪 Метрики качества и инженерии

- **Code coverage** — покрытие тестами
- **Количество багов** — включая найденные в продакшене
- **Степень автоматизации тестов** — unit, integration, E2E
- **Инциденты в проде** — частота и влияние
- **MTTR (Mean Time to Recovery)** — скорость восстановления после сбоя

## 📈 Продуктовые метрики

- **DAU / WAU / MAU** — активность пользователей
- **Conversion Rate** — эффективность ключевых пользовательских сценариев
- **Retention** — удержание пользователей
- **NPS / CSAT** — удовлетворённость пользователей
- **Успешные заказы / возвраты** — надёжность сервиса

## 📊 Метрики синхронизации и управления

- **Количество блокеров** — уровень «здоровья» команды
- **Скорость устранения блокеров** — эффективность работы PO, SM, архитекторов
- **Согласованность целей** — насколько цели команды соответствуют продуктовой стратегии
- **Прозрачность backlog'а** — доступность и актуальность задач

## 🔍 Где отображаются метрики

- **Jira dashboards** — velocity, scope change, выполнение задач
- **Grafana** — технические и инфраструктурные метрики
- **Retrospective boards** — наблюдения и выводы команд
- **Product dashboards** — активность пользователей, фичи, эксперименты

## ❗ Метрики ≠ KPI

Мы не превращаем метрики в KPI, не штрафуем за просадку velocity, не гоним команды по цифрам. Для нас важны:

- **Тренды**, а не единичные всплески
- **Причины**, а не симптомы
- **Командные обсуждения**, а не внешняя оценка

# 🔷 Раздел 8. Гильдии и развитие компетенций

## 📌 Цель

Создать устойчивую и масштабируемую модель роста экспертизы внутри команды и всей организации. Обеспечить условия для обмена знаниями, профессионального развития и стандартизации практик.

## 🧩 Что такое гильдии

**Гильдия** — это горизонтальное сообщество по интересу или экспертизе, объединяющее участников из разных команд и направлений. Гильдии не управляют, а **создают стандарты, делятся знаниями и развивают навыки**.

Типичные направления гильдий:

- Frontend / Backend / Mobile
- QA / Automation
- DevOps / SRE
- UX / UI
- Аналитика
- Продуктовая гильдия

## 🎯 Цели гильдий

- Обмен лучшими практиками
- Введение и поддержка стандартов
- Онбординг новых участников
- Выравнивание по качеству
- Поддержка карьерного развития
- Обсуждение новых инструментов и технологий

## 🧭 Форматы активности

- Регулярные встречи (раз в 2–4 недели)
- Внутренние митапы и демонстрации
- Коллективная работа над стандартами и гайдами
- Проведение аудитов и ревью на уровне практики

## 🔗 Связь с развитием компетенций

- Гильдии помогают определить **матрицы компетенций**
- Участники могут планировать развитие по понятным **трекам**
- Поддерживается культура **непрерывного обучения**

## 🧠 Как это влияет на продукт

- Повышается **качество реализации**
- Снижается **технический разнобой**
- Повышается **скорость онбординга**
- Улучшается **взаимопонимание между командами**

## 🚀 Роль руководства

- Не управляет гильдиями напрямую
- Создаёт условия: поддерживает время, инфраструктуру, поощряет участие
- Использует гильдии как источник экспертизы для продуктовых решений и технической стратегии

Гильдии — это не бюрократия, а **живой организм** внутри организации. Их сила — в добровольности, открытости и фокусе на развитии.

# 🔷 Раздел 9. Культура и лидерство

## 📌 Цель

Формировать культуру, которая поддерживает адаптивную разработку: безопасную, вовлекающую, ориентированную на рост и совместную ответственность. Лидерство здесь — не власть, а служение команде и цели.

## 🧭 Принципы лидерства

- **Служащее лидерство** — лидер не управляет сверху, а убирает препятствия и создаёт условия для команды
- **Прозрачность** — открытость в решениях, ошибках, целях
- **Доверие** — вместо микроменеджмента
- **Ответственность** — каждый понимает, за что он отвечает и как это влияет на результат
- **Участие** — лидер не изолирован, а включён в рабочий ритм команды

## 🛡️ Психологическая безопасность

- Ошибки не наказываются — они анализируются
- Каждый может говорить открыто и быть услышан
- Критика идей не превращается в критику людей
- Безопасность = основа для инноваций и улучшений

## 🚀 Рост и эволюция команды

- Команда растёт не только в скиллах, но и в **зрелости**
- Регулярная рефлексия: что мешает, что помогает
- Участники переходят от «делаю, что скажут» к «ищу, как принести пользу»
- Лидер поддерживает рост, а не оценивает сверху

## 🌱 Вовлечённость и смысл

- Участники понимают, **зачем** они делают продукт
- Видят ценность своей работы для пользователей
- Чувствуют, что могут влиять на результат
- Инициатива приветствуется и вознаграждается

## 🤝 Культура всей организации

- Открытые обсуждения — даже спорных тем
- Никаких «нельзя обсуждать» — есть формат и уважение
- Поддержка обмена знаниями между командами
- Лидерство — это не позиция, а поведение, доступное каждому

Культура — это не «мягкое дополнение», а **фундамент устойчивой и сильной организации**, особенно в условиях высокой неопределённости.

# 🔷 Роли в адаптивной модели: кто за что отвечает

## 📌 Цель

Создать общее понимание ролей в процессе разработки: кто за что отвечает и что не входит в его зону влияния. Это снижает недопонимание и повышает эффективность взаимодействия.

---

## 👨‍💼 Product Owner (PO)

**Что делает:**

- Отвечает за продуктовую ценность на уровне команды
- Формирует backlog, приоритизирует задачи
- Уточняет требования и описывает acceptance criteria
- Взаимодействует с пользователями и бизнесом
- Доступен команде на ежедневной основе

**Что не делает:**

- Не управляет людьми
- Не принимает технические решения
- Не вмешивается в самоорганизацию команды

---

## 👥 Scrum Master

**Что делает:**

- Поддерживает Scrum-процессы
- Убирает блокеры и фасилитирует встречи
- Помогает команде становиться самоорганизованной
- Защищает от внешнего давления
- Содействует ретроспективам и менторству

**Что не делает:**

- Не управляет задачами
- Не даёт указаний «что делать»
- Не принимает архитектурные решения

---

## 👨‍💻 Разработчик (Backend, Frontend, Mobile и др.)

**Что делает:**

- Реализует User Stories с соблюдением DoD
- Пишет тесты, участвует в code review
- Участвует в архитектурных решениях своей зоны
- Делится знаниями, активно участвует в гильдиях

**Что не делает:**

- Не работает «вслепую по ТЗ»
- Не ждёт указаний, а участвует в планировании
- Не снимает с себя ответственность за качество

---

## 🔎 QA-инженер

**Что делает:**

- Пишет автотесты и тест-кейсы
- Проверяет задачи по критериям приёмки
- Участвует в формировании DoD
- Вносит предложения по улучшению качества

**Что не делает:**

- Не «ловит баги за другими»
- Не тестирует вручную то, что можно автоматизировать
- Не работает изолированно от команды

---

## 👨‍💼 CEO — Генеральный директор

**Что делает:**

- Формирует vision и продуктовую стратегию

- Определяет цели и ключевые метрики
- Совместно с PO участвует в планировании roadmap
- Взаимодействует с рынком, пользователями, партнёрами

**Что не делает:**

- Не управляет командами напрямую
- Не погружается в детализацию задач (это роль PO)
- Не решает архитектурные вопросы

---

## 👨‍💼 CTO — Технический директор

**Что делает:**

- Отвечает за согласование целей, ритм поставки, синхронизацию команд
- Помогает устранять блокеры, связанные с людьми и процессами
- Работает в связке с CEO и архитекторами
- Следит за технической устойчивостью и продуктивностью

**Что не делает:**

- Не руководит командами директивно
- Не заменяет Scrum Master или PO
- Не занимается микроменеджментом

---

## 🧱 Архитектор (назначается Engineering Manager)

**Что делает:**

- Отвечает за архитектурную целостность и развитие продукта
- Помогает командам выбирать технологии и стандарты
- Влияет на инженерные практики и работу с техдолгом
- Проводит архитектурные ревью

**Что не делает:**

- Не навязывает решения без вовлечения команды
- Не делает pull-requests за всех
- Не указывает, как реализовывать каждую задачу

---

## 👨‍🎓 Глава гильдии (обычно старший инженер)

**Что делает:**

- Развивает горизонтальные компетенции
- Обновляет стандарты и матрицы навыков
- Организует митапы, демо
- Поддерживает менторство и рост экспертов

**Что не делает:**

- Не вмешивается в поставку
- Не заменяет тимлида или Scrum Master
- Не навязывает стандарты без участия команд

# 🔷 Цикл релизов и поставки (Delivery Flow)

## 📦 Формат

- 2-недельный спринт (10 рабочих дней)
- Старт: **Среда**
- Demo: **вторник второй недели**
- Релиз: **после демо или ночью**
- Цель: устойчивый ритм, контроль качества, поставка ценности на каждом этапе

## 🗓️ Ритм спринта по дням

**День 1 — Среда**
🗓️ Sprint Planning (1.5–2 ч): старт спринта, выбор задач, декомпозиция

**День 2 — Четверг**
Активная разработка

**День 3 — Пятница**
🗓️ Refinement (45 мин): подготовка задач на будущее

**День 4 — Понедельник (2 неделя)**
Завершение историй, интеграция

**День 5 — Вторник**
Тестирование, стабилизация

**День 6 — Среда**
Финализация, QA

**День 7 — Четверг**
Внутреннее демо, релизные заметки

**День 8 — Пятница**
🛑 Code Freeze, старт работы с техдолгом

**День 9 — Понедельник (3 неделя)**
🗓️ Refinement (45 мин), завершение работы с техдолгом, подготовка демо

**День 10 — Вторник**
🗓️ Sprint Review + 🗓️ Ретроспектива (1 ч + 45 мин): демонстрация, обратная связь, улучшения

---

# 🧾 Описание ключевых встреч

## 🗓️ Sprint Planning

**Когда:** Среда, день 1
**Участники:** PO, разработчики, QA
**Задачи:**

- Определить цель спринта
- Выбрать задачи
- Уточнить DoD и acceptance criteria
- Разбить задачи при необходимости

## 🗓️ Refinement

**Когда:** Пятница (1 неделя) и понедельник (2 неделя)
**Участники:** PO, разработка, QA
**Задачи:**

- Проработка будущих задач
- Добавление критериев приёмки
- Выявление зависимостей
- Первичная оценка

## 🗓️ Sprint Review / Demo

**Когда:** Вторник (2 неделя)
**Участники:** Команда, PO, стейкхолдеры
**Задачи:**

- Демонстрация инкремента

- Сбор обратной связи
- Подтверждение выполнения цели спринта
- Принятие решения о релизе

### 🗒️ Retrospective

**Когда:** Сразу после демо
**Участники:** Вся команда
**Задачи:**

- Анализ: что сработало, что нет
- Поиск улучшений
- Определение конкретных шагов

---

## 🔧 Работа с техдолгом

**Когда:**

- Начинается в пятницу (день code freeze)
- Основной объём — в понедельник следующей недели

**Что включает:**

- Рефакторинг
- Автотесты
- Обновление документации
- Улучшение логирования и мониторинга
- Обработка action items из прошлых ретро

# 📘 От эпиков к ценности: как писать истории, которые работают сквозь весь стек

## 📌 Почему это важно

Во многих командах задачи пишутся изолированно — backend-таска здесь, кнопка на frontend там, где-то отдельный тикет на тест. В итоге:

- фича не работает целиком,
- поставки не приносят ценности,
- Scrum превращается в список задач, а не в систему доставки.

Нужно это исправить.

## 🧩 Определения

- **Epic** — крупная цель, которую нельзя реализовать за один спринт. Например: «Добавить гостевую покупку», «Поддержать пуш-уведомления». Пересекает несколько систем и команд.
- **User Story** — часть эпика, которая несёт ценность пользователю и может быть завершена за спринт. Пример:
  *Как пользователь, я хочу оплачивать картой, чтобы завершить заказ.*
  История должна быть **демонстрируемой** и **готовой к поставке**.
- **Task** — технический шаг внутри User Story. Сам по себе ценности не несёт. Примеры: «создать API», «сделать верстку», «настроить фичу в DevOps».

## 🎯 Принцип: поставка полной ценности за спринт

Золотое правило:

> Если история не охватывает все необходимые компоненты — это не настоящая история.

Настоящая история объединяет задачи из:

- backend (логика, API, схема БД)
- frontend (UI, взаимодействие)
- mobile (если есть нативные клиенты)
- QA (тесты, регрессии)
- DevOps (флаги, конфигурации)

Только вместе эти части формируют реальную ценность.

## 🧠 Антипаттерн: изолированные истории

**Плохой пример:**

- Story 1: «Сделать endpoint оплаты»
- Story 2: «Показать кнопку оплаты»
- Story 3: «Написать тесты»

Что не так:

- velocity выглядит неплохо
- но пользователь не может платить
- в демо показывать нечего
- фрустрация у PM и стейкхолдеров

## ✅ Хорошая практика: тонкие вертикальные срезы

**Хороший пример:**
Story: *Как пользователь, я хочу оплатить заказ картой*

- Backend: реализовать `/pay` endpoint
- Frontend: отрисовать форму и вызвать API
- Mobile: аналогичная фича в приложении
- QA: тест на flow оплаты
- DevOps: включить фичу на staging через флаг

Все эти задачи — **части одной истории**. Она готова к показу, тестированию и релизу.

## 🔄 На практике: как структурировать работу

1. Начинай с потребности пользователя (epic → story)
2. Сформулируй story с фокусом на ценность
3. Разбей её на сквозные задачи по компонентам
4. Назначь story-владельца и отнеси в конкретный спринт
5. Проверяй готовность истории на grooming: если не готовы все компоненты — история не готова
6. На демо показывай **только завершённые истории**, не таски

## 📎 Полезные советы

- Используй Definition of Ready: все компоненты понятны, дизайн готов, критерии приёмки прописаны
- Держи истории маленькими: 2–3 дня на одного разработчика
- Если дольше — дели историю, а не команду
- Не «переноси наполовину сделанные» истории — это симптом плохой декомпозиции

Помни:
**Хорошая Scrum-команда не шипит код. Она поставляет ценность.**
А это возможно только при полном, сквозном завершении каждой истории.