alitenicole
24 мар 2025 в 16:09

База про жизненный цикл разработки ПО (SDLC): этапы, виды моделей и их различия

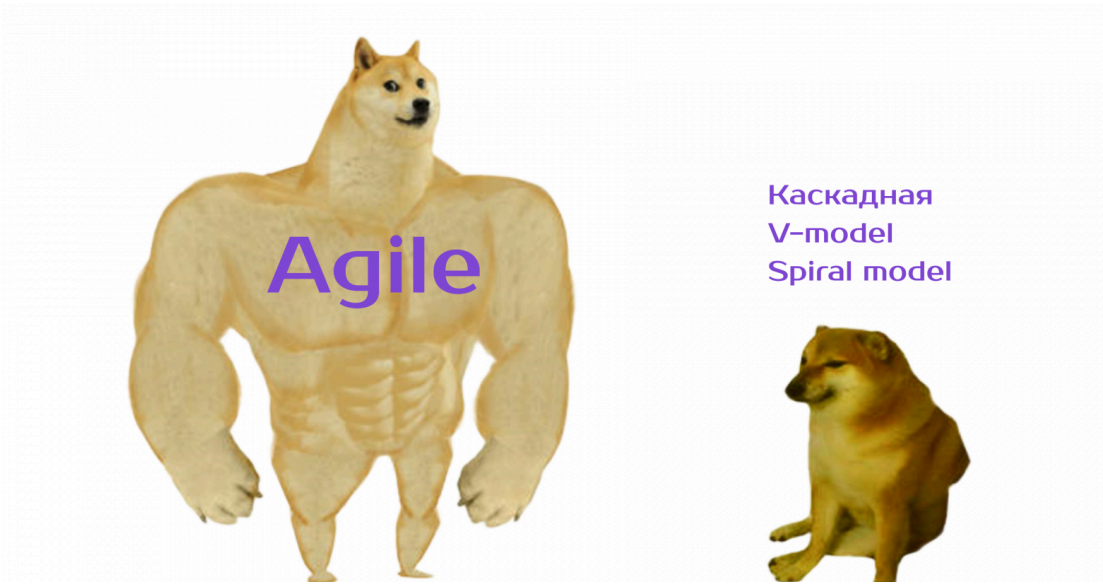
Простой 15 мин 79K

Блог компании Kaiten, Agile*, Управление разработкой*, IT-компании

Software Development Life Cycle (SDLC) — это фундамент, на котором строится разработка. Он помогает выстроить процессы так, чтобы команда четко понимала, что и когда ей нужно делать, а заказчик знал, на каком этапе находится работа.

И если с этапами работы чаще все понятно, то с жизненными моделями SDLC возникает путаница. В некоторых статьях могут писать, что какие-то из моделей устарели и нежизнеспособны, или просто неверно называть их принципы. Поэтому мы решили собрать основную информацию про SDLC в одном тексте.

А еще пообщались с командой AGIMA — интегратором, который более 15 лет создает веб-решения и мобильные приложения для клиентов. Компания показала, как выстроила процесс разработки и как она управляет командой.



Что такое SDLC и зачем он нужен

Software Development Life Cycle, или жизненный цикл разработки программного обеспечения — это пошаговый процесс разработки, который начинается с идеи и заканчивается готовым продуктом. Учитывать жизненный цикл нужно, чтобы хотя бы примерно начертить план работ и распределить ресурсы сотрудников, иначе разработка превратится в хаос.

Этапы жизненного цикла разработки ПО (SDLC)

Жизненный цикл подразумевает деление разработки на этапы. Каждый из них базируется на результатах предыдущего, поэтому командам проще работать слаженно, ведь процессы становятся предсказуемыми. Ниже разобрали основные этапы:

Планирование и анализ требований

У инициатора работ собирают бизнес-требования к ПО. Это важно, чтобы каждый в команде точно знал, что нужно сделать, и избежать недопонимания в будущем. Вот какие работы включает этот этап:

- Описание задач, которые должен решать продукт;
- Опрос стейкхолдеров;
- Выявление аудитории, которая будет пользоваться решением;
- Оценка ресурсов, которые нужны для реализации: количество сотрудников, время, бюджет;
- Согласование критериев, по которым будут оценивать успешность решения.

→ Кто участвует: руководитель проекта, владелец продукта, заказчик, бизнес-аналитик, стейкхолдеры.

Важно учесть, что на каждом этапе могут быть свои задачи, так как все зависит от компании. В AGIMA, например, также проводят обширные продуктовые исследования:

> «*С 2014 года у нас есть отдел количественных качественных исследований. Он помогает строить продукты клиентов не только на основе их понимания того, каким должен быть проект, но и на основе фидбэка и поведения текущих или потенциальных клиентов. Для этого мы проводим полевые исследования, юзабилити-тестирования, опросы и совмещаем их с количественными методами: разметка веб-аналитики, мобильной аналитики опросы, количественные исследования в полях. В итоге у нас появляются, например, CJM, барьеры, потребности и шаги, которые надо будет учесть в проекте. И все это попадает в видение проекта*», — команда AGIMA.

Определение требований

Когда цель работ стала примерно понятной, нужно ее детализировать и задокументировать. Тут тоже несколько подзадач:

- Оценка рисков;
- Выбор методологии работы;
- Описание пользовательских сценариев;
- подготовка SRS — документации с информацией о функциональных и нефункциональных требованиях;
- подготовка ИСР — иерархической структуры работ. Процесс разделяют на более мелкие элементы, чтобы упростить управление и понять, в каком порядке нужно выполнять проект, и какие задачи можно делать параллельно друг другу.

→ Кто участвует: руководитель проекта, владелец продукта, команда разработки, архитектор.

Проектирование

Чтобы разработчики понимали, как система будет работать, и чтобы избежать ошибок на этапе разработки, создают архитектуру системы. Для этого занимаются верхнеуровневым и низкоуровневым проектированием.

При верхнеуровневом создают общую структуру системы, а низкоуровневое проектирование отличается большей детализацией. В таблице показали, какие работы включает каждый вид проектирования:

Верхнеуровневое проектирование	**Низкоуровневое проектирование**
Основные компоненты системы — например, модули, сервисы или подсистемы	Принцип работы отдельных компонентов системы
Принцип взаимодействия этих компонентов между собой	Алгоритмы, структуры данных и логика работы внутри каждого модуля
Общая архитектура системы — например, клиент-серверная архитектура или микросервисы	Детали реализации — например, какие классы, методы или функции будут созданы
Зависимость решения от других систем	Взаимодействие с базами данных, API или другими внешними системами
Прототип интерфейса	

Результатом этапа будет проектная документация с информацией, которая необходима для реализации решения.

→ Кто участвует: архитектор, UX/UI-дизайнер, QA-инженер, команда разработки.

Разработка

Следующий шаг — превратить идею в реальный продукт. Разработчики пишут код, интегрируют компоненты и проводят модульное тестирование. Итогом становится рабочее решение в соответствии с техническими спецификациями.

→ Кто участвует: Разработчики, техлид, DevOps-инженер, QA-инженер.

Тестирование

Теперь нужно убедиться, что продукт соответствует требованиям, работает без сбоев и решает задачи пользователей. Для этого готовую программу проверяют на ошибки. Проверка включает интеграционное, системное и пользовательское тестирование. Если находят ошибки, их передают разработчикам на исправление.

→ Кто участвует: QA-инженеры, разработчики, автоматизаторы тестирования, потенциальные пользователи в случае User Acceptance Testing.

Развертывание

Программу передают заказчику и выпускают в «боевую среду» — устанавливают на серверы или выкладывают в магазины приложений, чтобы аудитория могла им пользоваться. Дополнительно готовят пользовательскую документацию.

→ Кто участвует: DevOps-инженер, разработчики, руководитель проекта.

Поддержка и сопровождение. Работа с ПО продолжается — его обновляют под запросы пользователей, устраняют ошибки при появлении и оказывают техническую поддержку пользователям.

→ Кто участвует: техническая поддержка.

«Смерть» ПО

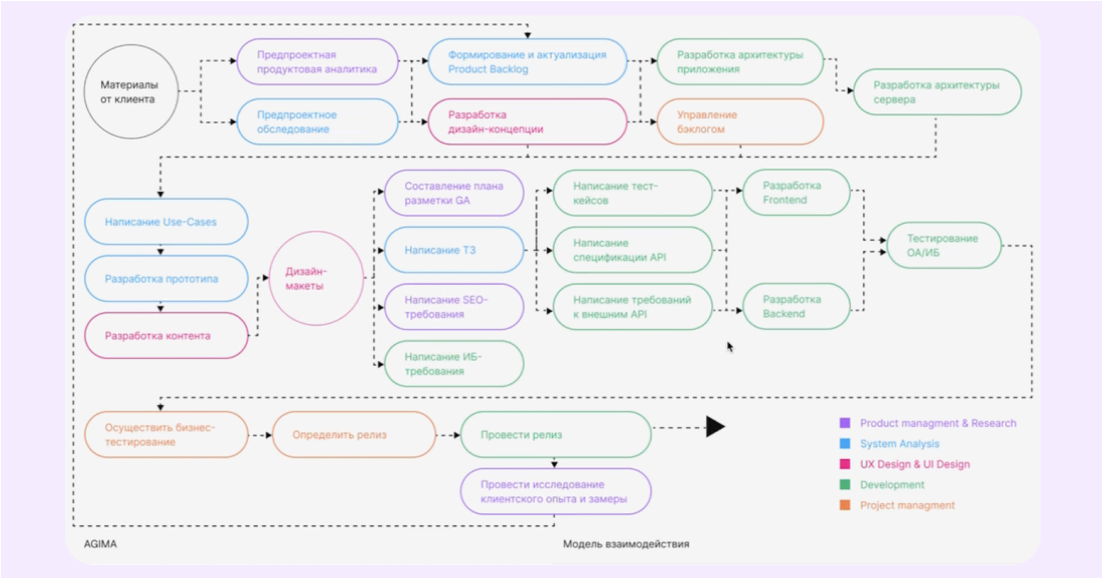
На этом этапе команда прекращает обслуживать продукт или заменяет его новой версией, потому что ПО устарело или такая инициатива поступила от заказчика. Процесс может включать архивирование данных и уведомление пользователей.

Детали каждого этапа могут отличаться и это нормально

Продакт-менеджеры могут использовать концепцию SDLC как памятку, чтобы понимать общий принцип разработки. Но хоть SDLC считается стандартом, в каждой компании процесс может называться по-своему и при необходимости включать дополнительные этапы или иметь другую последовательность выполнения подзадач. Главное — чтобы разработка шла по плану, во взаимодействии команды была логика, а результат приносил ценность заказчику и пользователям.

> *«Мы организовали свою работу как взаимосвязь из нескольких сервисов, которые взаимодействуют между собой. Один из сервисов, который раньше у нас назывался Value Delivery — это сервис первичной поставки ценности, когда после продажи мы показываем клиенту себя в деле»*, — команда AGIMA.

Процесс Value Delivery у AGIMA очень напоминает последовательность по SDLC и выглядит так:



После развертывания команда собирает фидбек и на его основе выпускает новые релизы

Что такое модели жизненного цикла разработки ПО и зачем они нужны

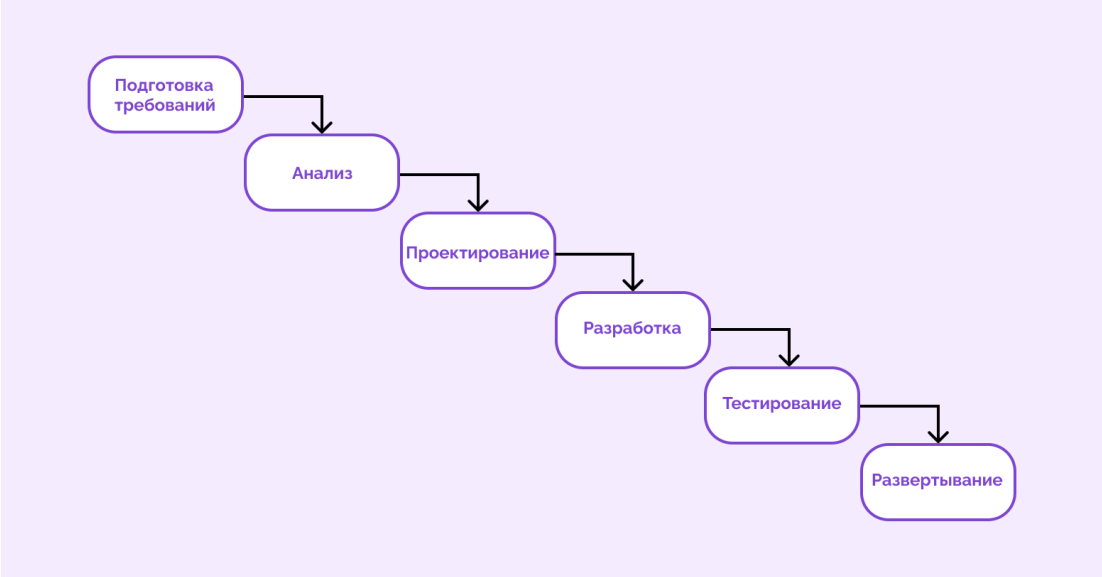
Между некоторыми IT-продуктами сильная конкуренция — команды пытаются определить конкурентов, быстрее внедрить новую функциональность и подстроиться под запросы рынка. И если разработка вовсю идет, а заказчик приходит с новыми требованиями, то план работ постепенно превращается в кашу из разных запросов с постоянно меняющимися приоритетами. В итоге жизненный цикл ломается.

Чтобы такого не было, нужно заранее договориться о формате работы. В этом помогают модели жизненного цикла ПО. По сути, это методы организации и шаблоны процессов, логика которых отличается в зависимости от того, на чем нужно сделать акцент: предсказуемости результата или гибкости процессов. Команды могут взять за основу один из шаблонов, который им больше подходит, и согласовать подход к работе вместе с заказчиком. Обзорно рассмотрим основные модели SDLC ↓

Каскадная, или водопадная, модель

Каскадная модель устроена как классический последовательный процесс. Это значит, что движение происходит только вперед от одного этапа к следующему. При работе по каскадной модели на последнем этапе заказчик получает готовое решение, которое не требует доработок.

Принцип простой, но строгий, поэтому некоторые считают его устаревшим. Однако он все еще подходит для некоторых проектов в самостоятельном виде или в сочетании с гибкими методиками.



Один этап перетекает в другой, и так вплоть до релиза

Плюсы:

- Полное документирование каждого этапа до начала разработки;
- Низкий риск ошибок за счет детальной документации;
- Прозрачность процессов для заказчика — он знает, сколько времени уйдет на каждый этап.

Минусы:

- Перед стартом нужно подготовить обширную техническую документацию, что занимает много времени;
- Сложно учесть все требования до старта разработки;
- Нет гибкости — если появились новые требования на этапе разработке, то откатить работу назад не получится;
- Заказчик видит результат в конце разработки — если итог его не устроит, придется начинать сначала.

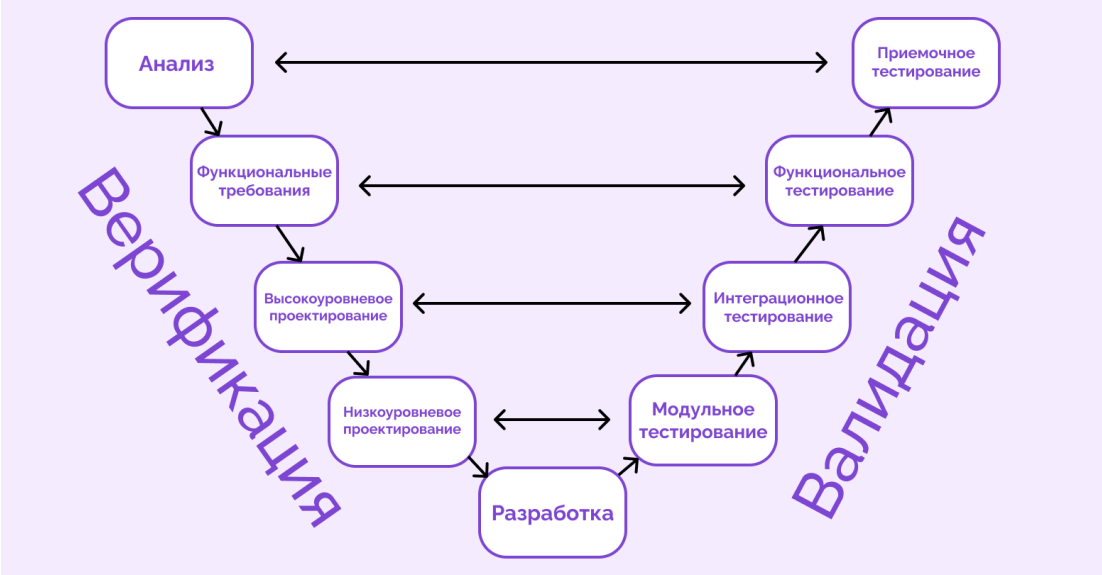
Когда подходит водопадная модель:

- Есть четкие и заранее известные требования, которые не будут меняться на этапе разработки.
- Работа должна идти по строгим регламентам — такое требование может быть актуальным для госучреждений или банковских систем.

V-образная модель

Это расширенный вариант каскадной модели, который делает упор на тестировании. Устроена модель так: до начала разработки после каждого этапа проводят тестирования разного уровня, а после разработки ПО проходит те же самые тестирования в обратном порядке и только потом уходит в релиз. Тестирования до разработки нужны для верификации, а тестирования после для валидации:

- Верификация — проверка ПО соответствествие правилам и требованиям в документации.
- Валидация — проверка готового ПО на соответствие ожиданиям заказчика.



Название «V-образная» происходит от визуального отображения процесса: слева идут этапы разработки, а справа — соответствующие им этапы тестирования, образуя букву V

Плюсы:

- Ошибки выявляют на ранних этапах, что снижает затраты на их исправление;
- За счет четкой структуры подходит для проектов с ясными и неизменными требованиями;
- Акцент на тестировании обеспечивает надежность продукта;
- Высокая степень анализа рисков.

Минусы:

- Такая же строгая, как и каскадная, поэтому при появлении новых требований все придется начинать заново;
- Разработка обходится дорого.

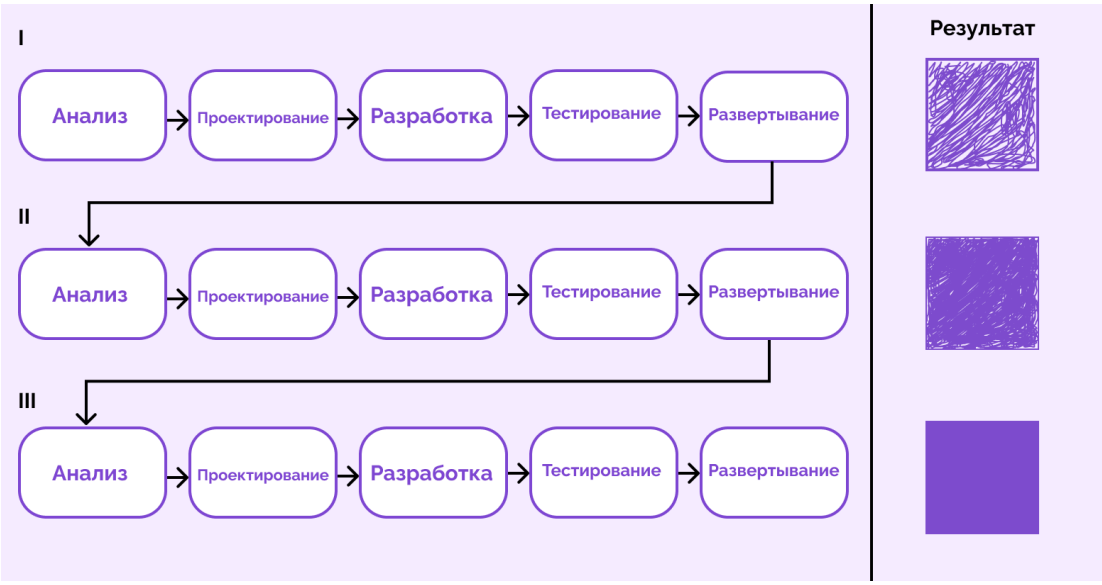
Когда подходит модель:

- Есть четкие и заранее известные требования, которые не будут меняться во время разработки;
- Работа должна идти по строгим регламентам;
- Проект предполагает большое тестовое покрытие.

Итеративная модель

В отличие от водопадной модели, итеративная позволяет обновлять требования к продукту после старта разработки. Для этого проект дробят на части и сначала выпускают MVP-версию, а затем итерациями доводят решение до ума. По ходу разработки требования к ПО можно менять в зависимости от обратной связи пользователей, заказчика или изменений на рынке. Цикл повторяется до тех пор, пока вся система не будет готова.

Получается так, что каждая итерация — это мини-проект, который включает анализ, проектирование, разработку, тестирование и выпуск готового к эксплуатации продукта.



При итеративном подходе пользователям как можно быстрее поставляют весь продукт, пусть и не в идеальном виде. Затем с каждой итерацией он становится лучше и лучше

Плюсы:

- Возможность быстро выкатить ПО;
- Заказчик видит промежуточные результаты и может корректировать требования;
- Быстрая обратная связь от пользователей;
- Проще находить конфликты между требованиями.

Минусы:

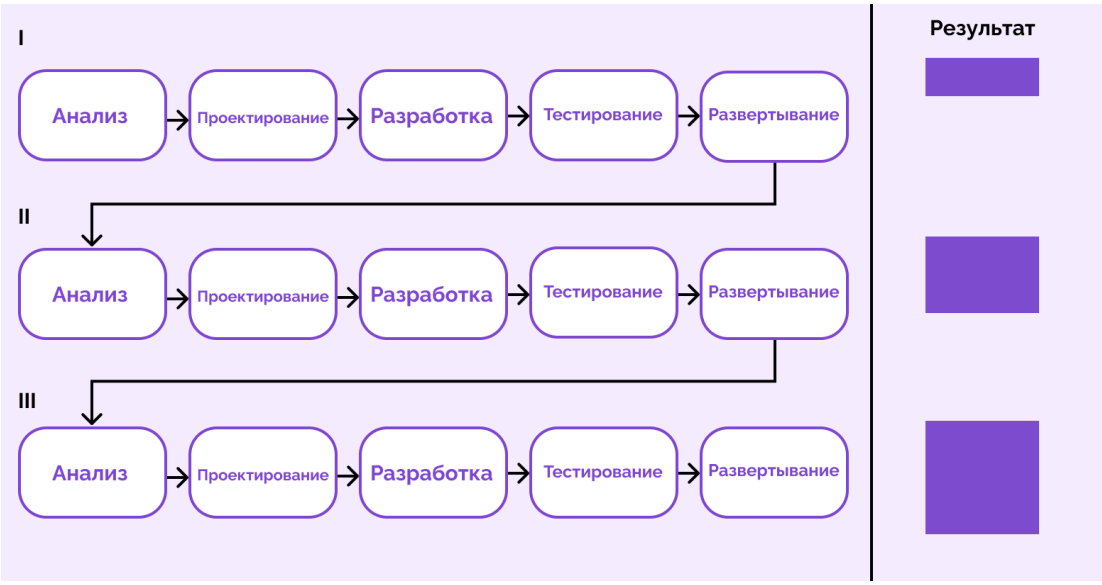
- Могут возникнуть сложности с созданием рабочей архитектуры, так как изначально не всегда известны все требования.

Когда подходит модель:

- Рабочее решение нужно в короткие сроки;
- Требования не до конца ясны и могут меняться в процессе работы;
- Проект очень большой, а ресурсов немного, поэтому приходится разбивать продукт на части и делать его постепенно.

Инкрементная модель

При работе по этой модели продукт создают по частям, или инкрементам. Каждая часть добавляет новую функциональность к уже существующей системе. В отличие от итеративной модели, где каждая итерация может пересматривать и улучшать предыдущие результаты, в инкрементной модели каждая часть — это законченный кусок функционала, который можно использовать.



Если итеративная модель подразумевает выпуск продукта целиком, но в неидеальном виде, то инкрементная подразумевает выпуск продукта частями, где каждая часть не требует доработо

Плюсы:

- Пользователи могут начать использовать продукт уже после первого инкремента;
- Заказчик видит прогресс и может вносить правки в следующие инкременты.

Минусы:

- Если команды параллельно работают над разными инкрементами, есть риск того, что модули будет сложно связать;
- Необходимо тщательное планирование, чтобы заранее определить, какие функции будут в каждом инкременте, и учесть их в архитектуре.

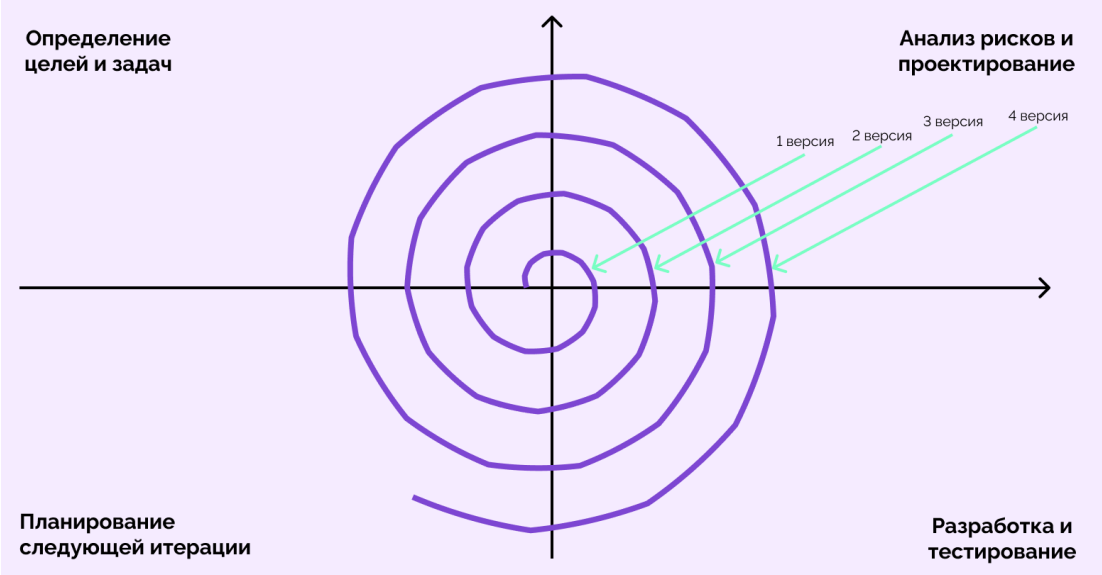
Когда подходит модель:

- Рабочее решение нужно в короткие сроки;
- Требования не до конца ясны и могут меняться в процессе работы;
- Проект большой, а ресурсов немного, поэтому приходится разбивать продукт на части и делать его постепенно.

Спиральная модель

Модель объединяет в себе преимущества каскадной и итеративной моделей, но делает большой упор на анализ рисков. То есть, команда уделяет много времени планированию и при этом в будущем может дорабатывать решение. Еще одна особенность модели в том, что процесс разработки делится на циклы, которые изображают как витки спирали. И каждый виток состоит из четырех этапов:

- планирование;
- анализ рисков;
- разработка и тестирование;
- оценка и планирование следующего витка.



Каждый виток = новая версия продукта

Плюсы:

- Риски находят на ранних этапах, что снижает вероятность проблем в будущем;
- Можно вносить изменения в проект по мере его развития;
- Подходит для крупных систем, где много неопределенности и высокие требования к надежности.

Минусы:

- Многочисленные циклы растягивают разработку и делают ее дороже;
- Каждое новое требование заказчика запускает новый виток. Это делает разработку дороже, так как нужно снова тратить ресурсы на анализ.

Когда подходит эта модель:

- Пользователи сами не до конца понимают, что им нужно;
- Требования к проекту слишком сложные и могут меняться по ходу работы;
- Успех проекта не гарантирован, и нужно заранее оценить риски, чтобы решить, стоит ли продолжать работу;
- В проекте используют новые технологии, которые еще не до конца изучены, и есть риск, что они не дадут ожидаемого результата.

Agile

Это не совсем модель, а набор методов для гибкого управления проектами. Их суть в развитии продукта через частые обновления и работе в условиях неопределенности, когда требования меняются в процессе разработки. Идеи Agile не новы — в основе методологии лежат итеративная и инкрементальная модели, которые стали еще более адаптивными к постоянному обновлению требований.

Популярность Agile начала расти в 2000-х. К тому моменту многие понимали недостатки традиционных моделей, а в 2001 году появилось решение в виде манифеста от разработчиков. Он стал основой для новой философии разработки. Вот основные принципы из манифеста:

- люди и взаимодействие важнее процессов и инструментов;

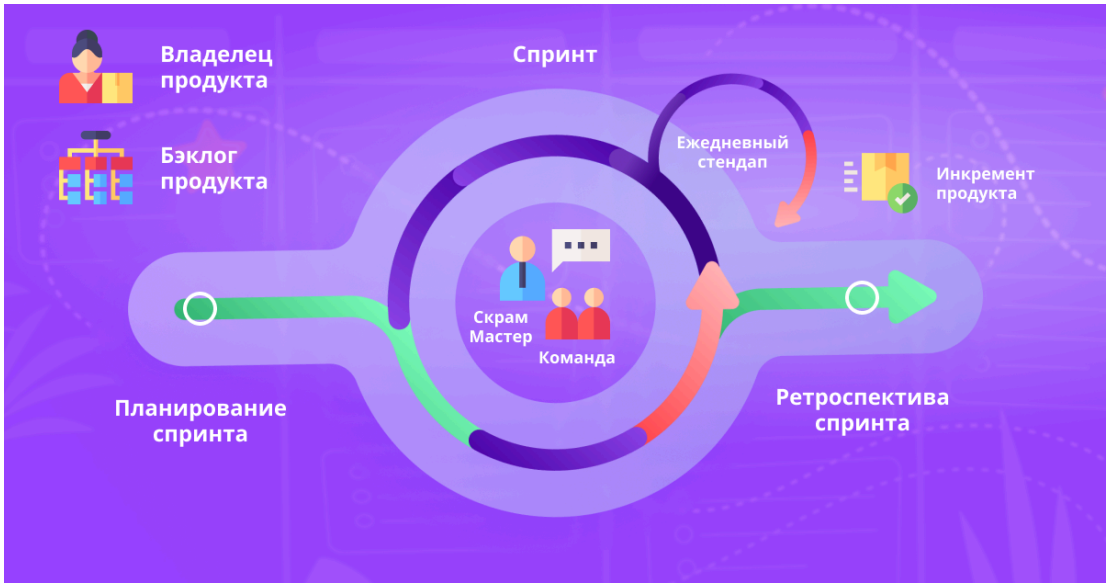
- рабочий продукт важнее документации;
- сотрудничество с заказчиком важнее согласования условий контракта;
- готовность к изменениям важнее следования плану.

> *«Agile помогает решить одну из классических ошибок — стремление как можно больше начать и по итогу как можно меньше закончить. Вместо того гибкие методики помогают сфокусировать команду на том, чтобы довести продукт до чего-то качественного, актуального и готового к выпуску. Происходит это по-разному, в зависимости от подхода. При работе по Scrum-фреймворку это происходит за счет коротких спринтов, при работе по Kanbun-методу — за счет визуализаций, WIP-лимитов и потоковых метрик»*, — команда AGIMA.

Рассмотрим эти Agile подходы подробнее.

Scrum

В основе этого фреймворка лежат короткие спринты, которые обычно длятся по 2-4 недели. Чем короче спринт, тем более гибкий процесс разработки и более быстрая обратная связь от заказчика или пользователей. В конце каждого спринта команда выпускает рабочий продукт, а затем проводит ретроспективу, где обсуждает итоги работы, сильные стороны команды и точки роста.



Из других особенностей SCRUM: наличие ежедневных стендапов и новая роль в процессе — скрам-мастер, которые координирует работу команды

Плюсы:

- Можно быстро реагировать на изменения требований;
- Заказчик видит рабочий продукт уже через несколько недель;
- Постоянная обратная связь, так как проходят регулярные демонстрации и обсуждения с заказчиком;
- Короткие итерации и видимый прогресс вдохновляют команду.

Минусы:

- Зависимость от заказчика, ведь если он не участвует в процессе, подход теряет смысл;
- Зависимость от вовлеченности сотрудников;
- Не всегда есть возможность детально оценить риски.

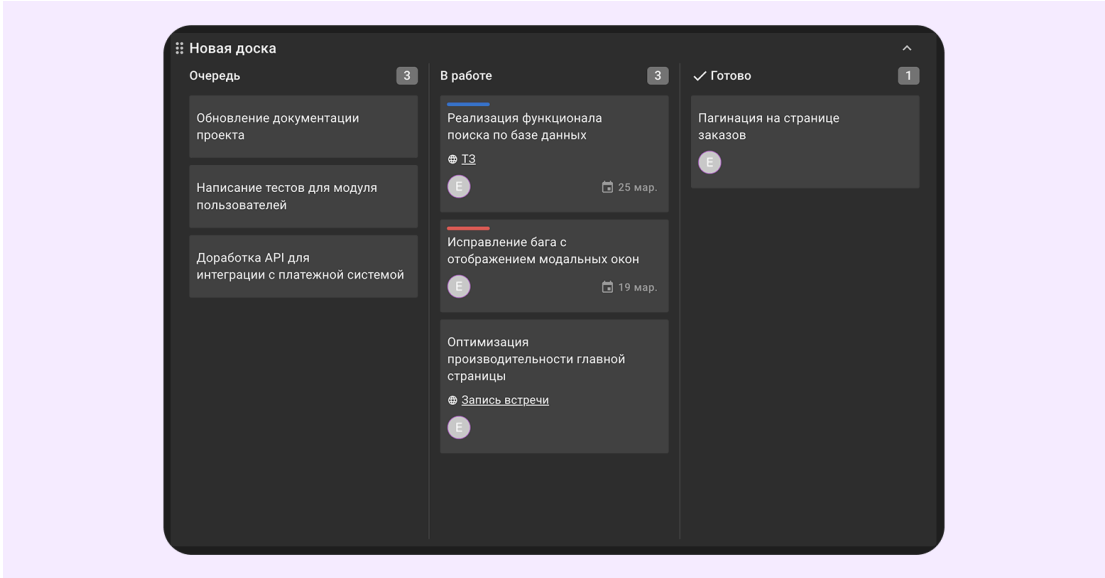
Когда подходит модель:

- Требования к проекту нечеткие или часто меняются;
- Необходима быстрая доставка ценности заказчику;
- Заказчик активно вовлечен в процесс;
- Команда готова к самоорганизации и сотрудничеству.

Kanban

Это не готовая структура процессов, а подход, который улучшает работу по уже выбранной модели. То есть, Kanban не используют вместо текущей методологии, а добавляют к ней, чтобы сделать работу более гибкой и прозрачной.

Подход подразумевает использование канбан-досок, где отображают этапы работы и распределяют карточки с задачами, а также постоянный мониторинг отчетов, в частности накопительной диаграммы потока. За счет этого у команд получается визуализировать процесс, ограничивать количество задач в работе, устранять узкие места и прогнозировать сроки выполнения работ.



Самый простой способ внедрить метод — начать с малого. Будет достаточно разбить доску на три колонки: «Запланировано», «В работе» и «Готово» — а затем накладывать на эту модель свои процессы

Плюсы:

- Легко внедрить подход;
- Легко адаптируется под изменения, так как подход включает получение обратной связи;
- Все видят, какие задачи в работе и на каком этапе они находятся, что повышает прозрачность процессов;
- WIP-лимиты помогают избежать многозадачности.

Минусы:

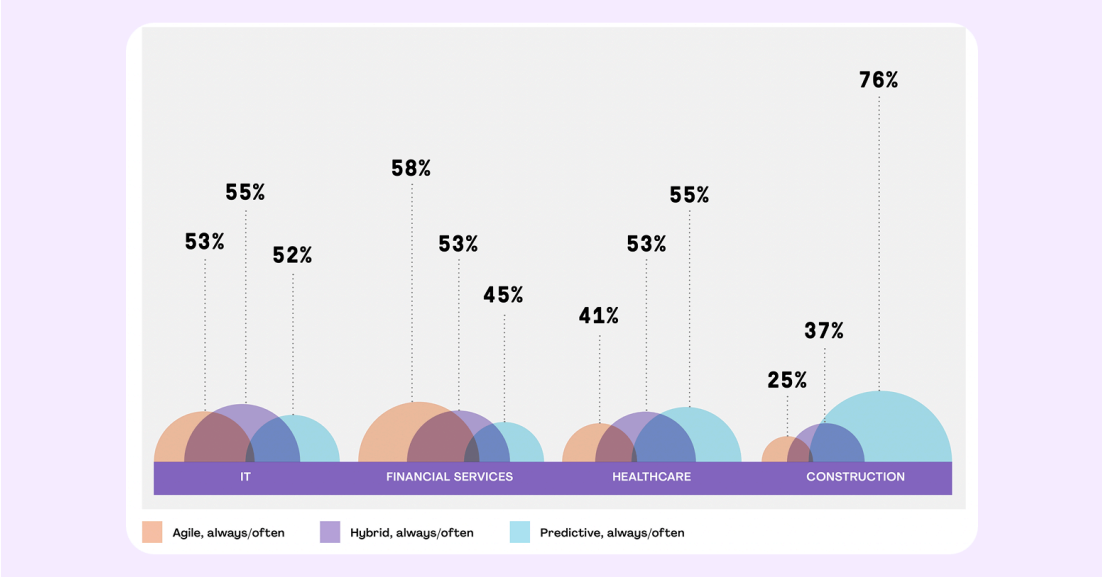
- Может быть сложно организовать работу команды от 10 человек.

Когда подходит эта модель:

- Ограничений у канбан-метода нет, но в первую очередь он подходит командам, которые делают упор на прогнозировании сроков работы.

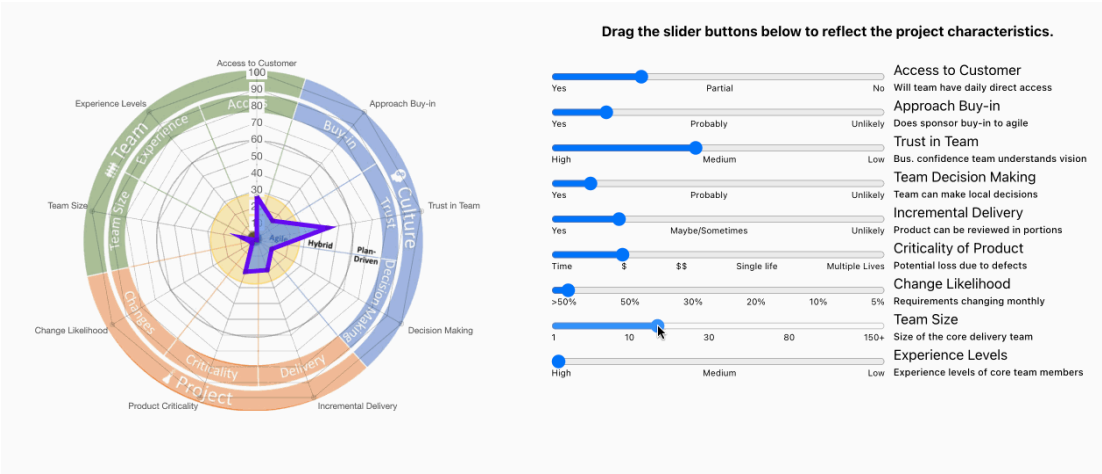
Как выбрать правильную модель SDLC

Модели SDLC полезно учитывать при организации команды, но на практике необязательно строго следовать какой-то одной из них. В компаниях могут быть устоявшиеся процессы, особые договоренности с заказчиком, а работа по текущей модели может не вредить качеству продукта, даже если она считается неподходящей. К тому же, в компаниях могут сочетать одновременно несколько подходов в зависимости от задачи.



Исследование Project Management Institute, PMI, за 2024 год показывает, что в IT примерно одинаково часто используют как традиционные, так и гибкие подходы

Но если запрос на изменение процессов есть, то выше мы рассказали, при каких обстоятельствах подойдет та или иная модель. Чтобы упростить выбор, можно использовать инструмент Agile Suitability Filter. Это небольшой опросник, который показывает, по какой методологии лучше работать команде: предиктивной, гибкой или гибридной. К предиктивной относятся каскадная и V-образная модели, к гибким — методологии Agile, а к гибридным — сочетание предиктивных и гибких.

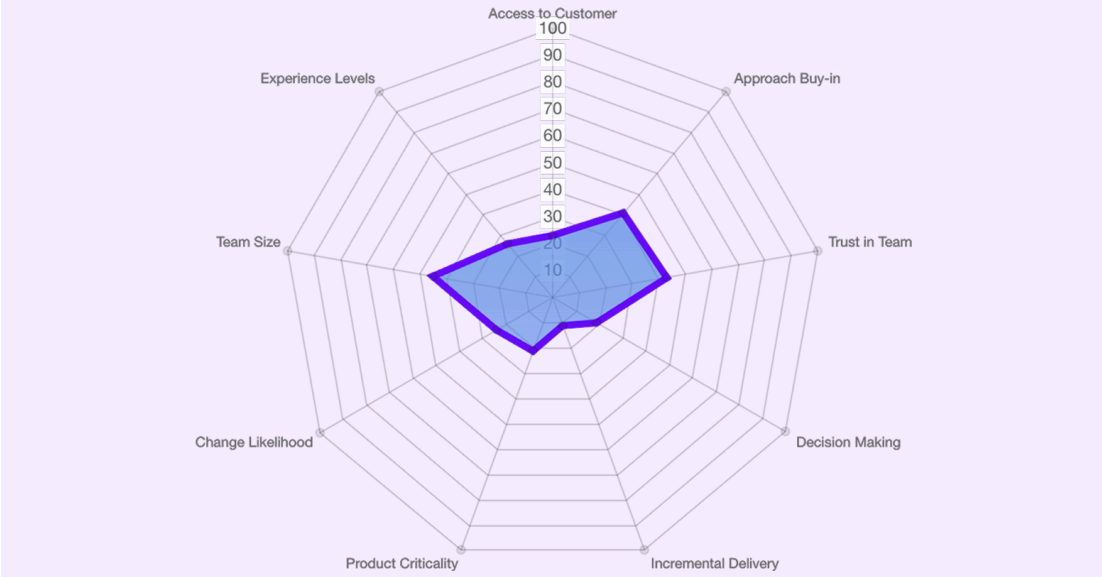


Фильтр сразу показывает, какая методология больше подходит компании

На какие вопросы нужно ответить:

- Есть ли у команда постоянный доступ к заказчику?
- Поддерживает ли спонсор гибкие методологии?

- Насколько вы уверены в том, что команда понимает ваша видение?
- Может ли команда сама принимать решения?
- Можно ли поставлять продукт частями?
- Как часто меняются требования к продукту?
- Какой размер команды?
- Насколько критичные проблемы будут в случае дефектов?
- Насколько большой опыт у команды?



В итоге получится график с контекстом, в котором находится ваша команда. Чем больше точек рядом с центром, тем больше вам подходят гибкие методологии

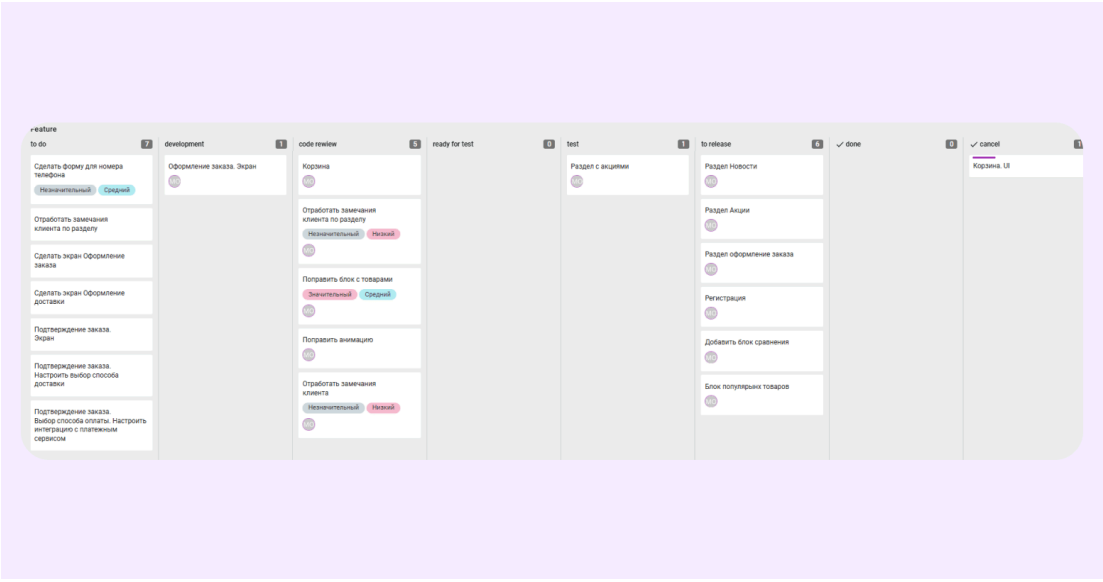
Опять же, результаты опросника — не вердикт, а только отправная точка. С ним станет понятно, к какому варианту больше относится проект, а внедрять новые процессы уже лучше, как минимум, после общения с командой.

Как эффективно управлять этапами SDLC

Для этого нужна система управления проектами. Одна из них — таск-трекер Kaiten, который подойдет для работы с любой моделью жизненного цикла. Вот что можно делать с его помощью:

Визуализировать процессы с помощью канбан-досок. Это одна из ключевых возможностей Kaiten, так как систему изначально разрабатывали на основе принципов Kanban-метода. И хоть метод рассчитан на гибкую разработку, его базовый инструмент в виде досок упростит работу и по традиционным моделям.

Например, разработку можно разбить на стадии и перемещать карточки с задачами по мере продвижения работы. Это поможет отслеживать, на каком этапе возникают сложности и какие задачи сейчас в работе у команды. Если вдруг что-то идет не так, то по меткам будет понятно, где возникают проблемы.

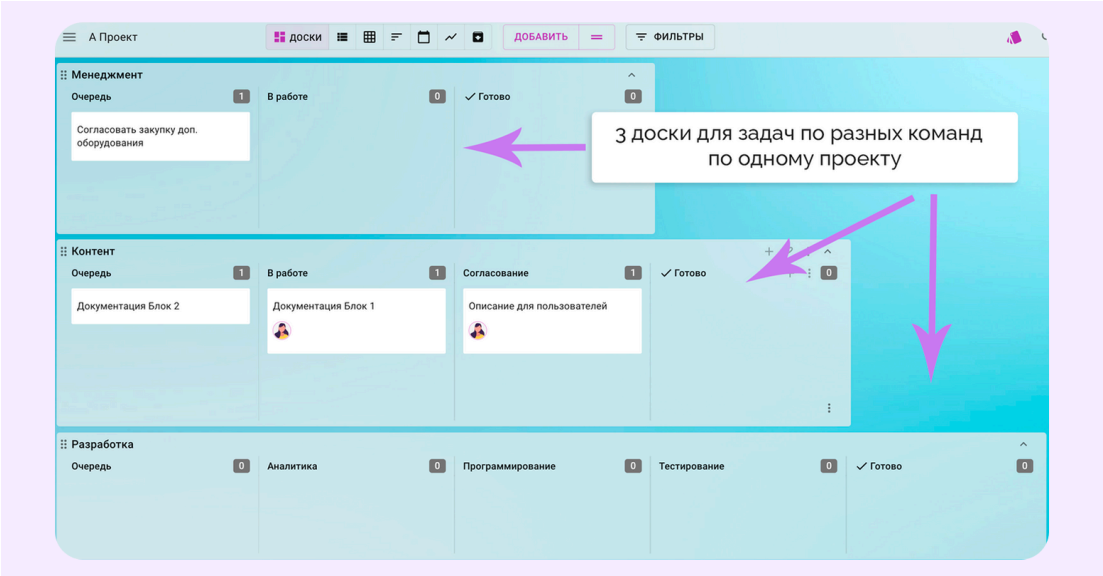


Доску можно подстроить под процессы своей компании: создать столбцы для каждого этапа, добавить собственные метки и отображение дополнительной информации

Создать несколько досок для разных команд. Работу с визуализацией можно развивать, чтобы процессы стали более прозрачными, а команде было проще работать с задачами. Например, разбить одну доску на несколько: одна доска — одно направление работ со своими этапами и стандартами.

К такому формату пришла команда AGIMA, которая создала три доски: Management, Requirements и Features. Помимо этого задачи от разных команд попадают на специальные доски, которые видят руководители подразделений.

Подбный подход можно реализовать в Kaiten. Одна из особенностей таск-трекера — возможность создать безграничное количество досок на одном пространстве. Это помогает видеть проект целиком без необходимости переключаться между вкладками.

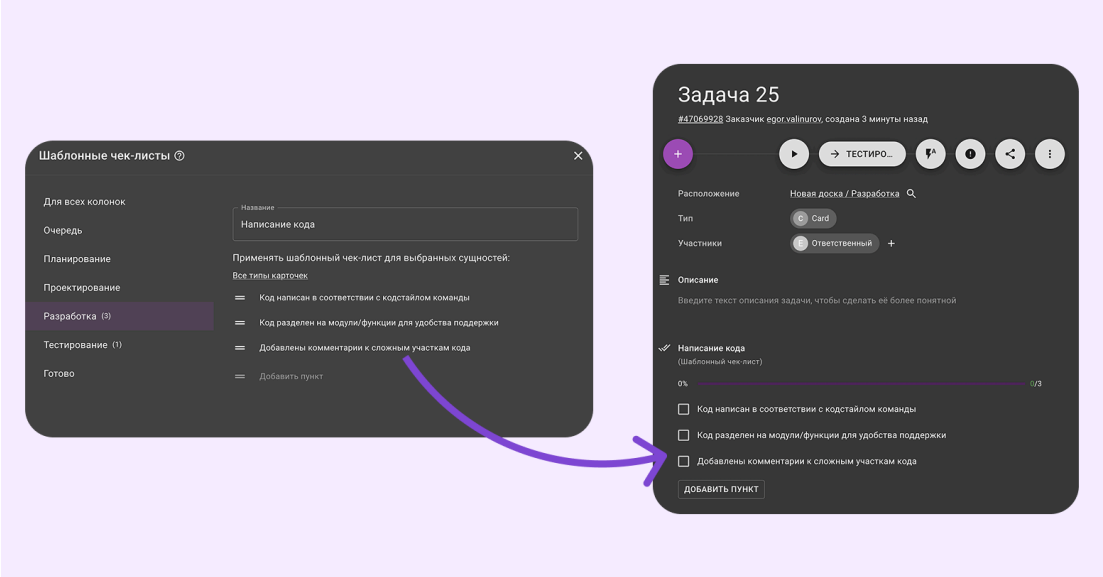


Когда на одном пространстве сразу несколько досок, у каждой может быть свое количество колонок, свои названия этапов и свои чек-листы, поэтому можно гибко настроить процессы под себя

→ Читайте также: Как команда AGIMA переехала на Kaiten и стандартизировала процессы

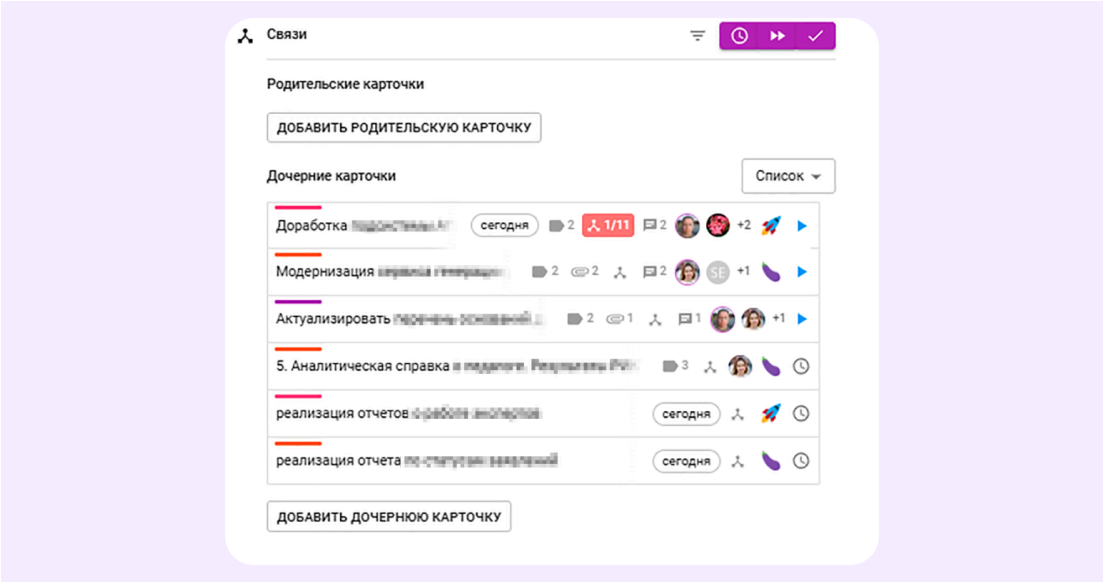
Автоматизировать контроль за соблюднием регламентов. Если у каждого шага разработки есть свои регламенты, их можно помещать в карточки в виде чек-листов. Однако когда речь идет о типовых задачах, добавлять чек-листы вручную будет не очень удобно. На такой случай у Kaiten есть шаблонные чек-листы.

Шаблонный чек-лист нужно заполнить один раз, привязать к типу задачи и указать, на каком этапе работы он должен появиться. После этого чек-листы будут добавлять автоматически. А чтобы подробнее отразить этапы выполнение задачи, можно создать по чек-листу для каждого подэтапа.



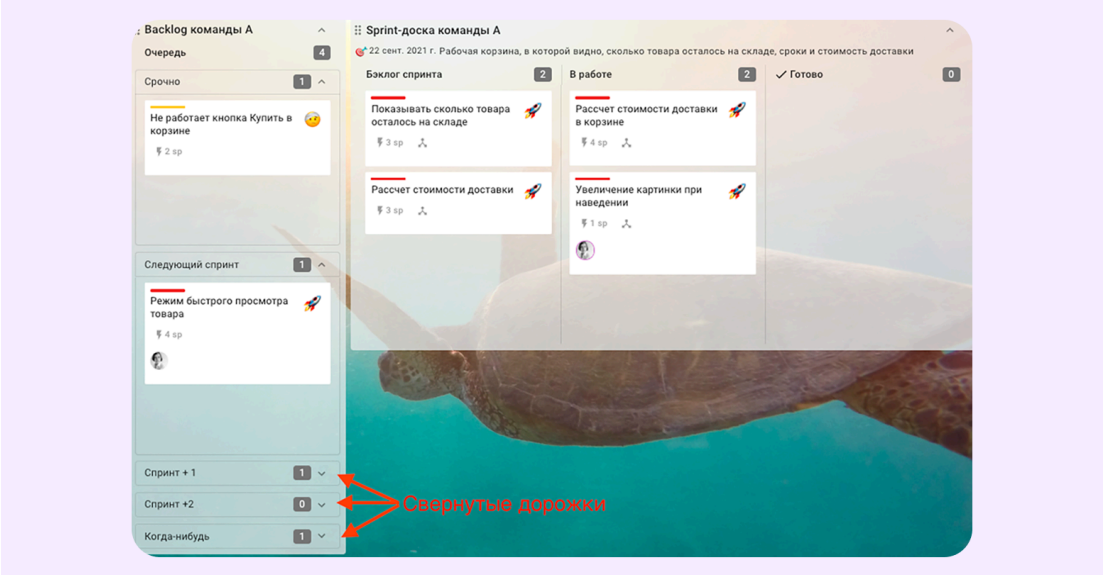
За счет чек-листов процесс разработки становится более детализированным. Для большей прозрачности для каждого пункта чек-листа можно указать свой дедлайн

Декомпозировать задачи. В Kaiten можно создавать дочерние карточки для подзадач и для каждой из них назначать своих ответственных. Это помогает точнее оценить усилия на выполнение задачи.



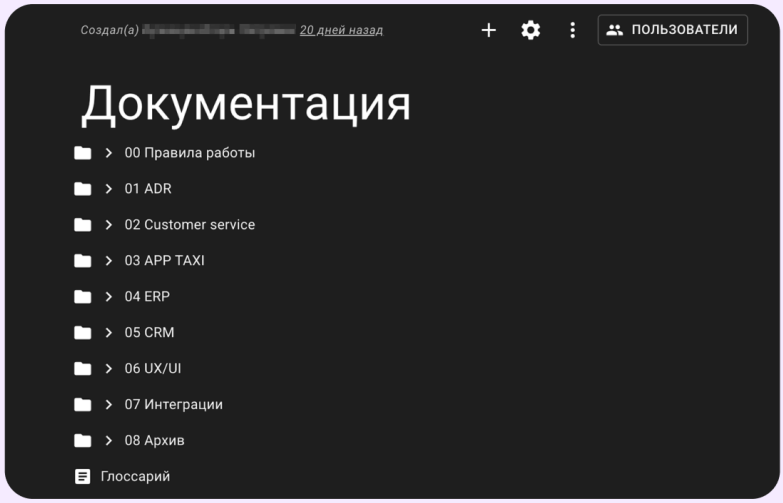
Из основной карточки можно сразу перейти в любую дочернюю и наоборот. Это упрощает навигацию между задачами

Организовывать спринты по SCRUM. Если работа идет в формате коротких итераций, то в Kaiten для этого есть преднастройка для SCRUM. Можно задавать сроки спринта, добавлять задачи в бэклоги и затем просматривать в отчетах результаты по количеству задач.



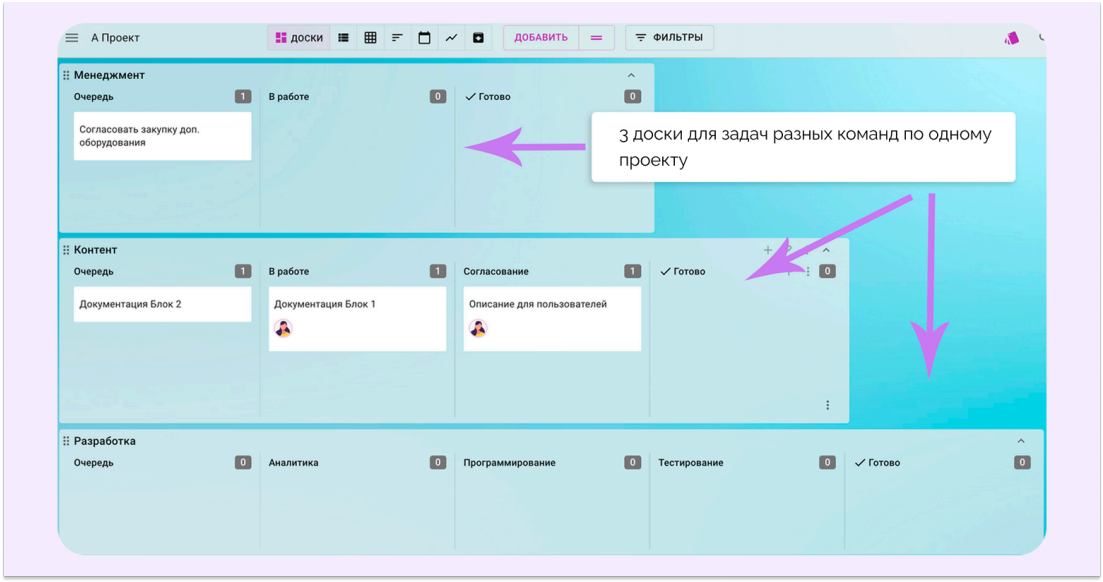
Слева на пространстве будет отдельная доска с бэклогом проекта, куда можно добавлять новые задачи и распределять их по приоритетам

Хранить всю документацию в одном месте. Чтобы ничего не потерялось и была возможность быстро свериться с планом, всю документацию можно загружать в Kaiten. Можно прикреплять туда ссылки на документы из внешних редакторов или полностью переносить документацию в сервис — для этого в Kaiten есть инструменты верстки.

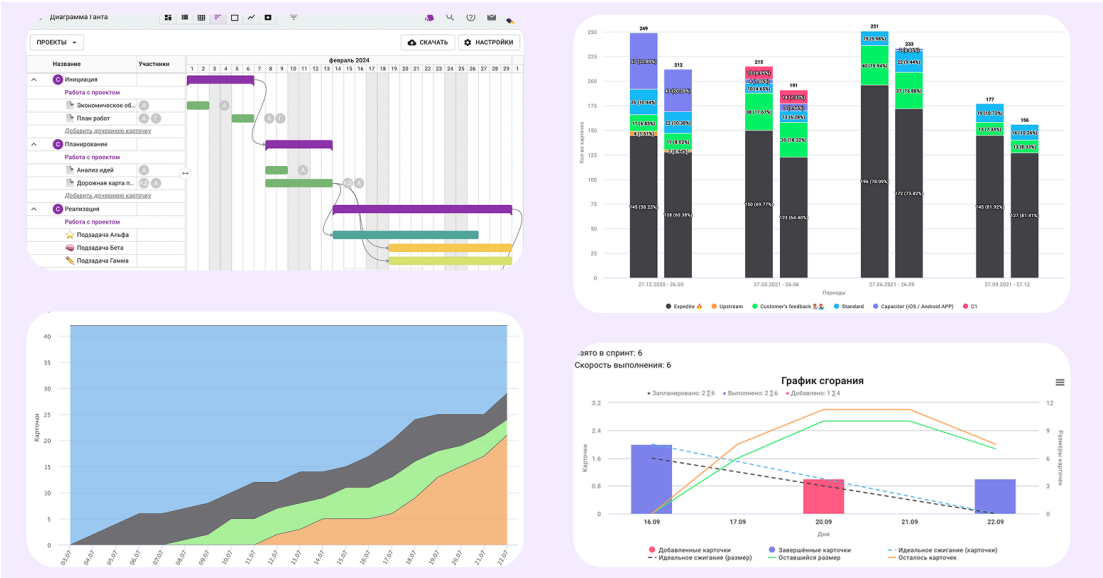


Документы можно разбить по папкам, чтобы структурировать все вводные о проекте

Отслеживать эффективность команды с помощью отчетов. В Kaiten есть общие отчеты, которые подходят для работы по любой модели SDLC — например, отчеты по распределению карточек и срокам по задачам.



Еще есть отчеты для конкретных методов работы. Например, диаграмма Ганта с ресурсным планированием подходит для линейной и долгосрочной разработки и показывает зависимости между каждым этапом. Или накопительная диаграммма потока, которую используют адепты канбан-метода.



Отчеты в Kaiten находятся в отдельном разделе, а создать их можноза пару кликов

Резюме

- SDLC, или жизненный цикл разработки ПО — последовательные этапы, из которых состоит разработка. Работа по этапам помогает избежать хаоса в процессах.
- Основные этапы разработки: анализ требований, определение требований, проектирование, разработка, тестирование, развертывание, поддержка и вывод из эксплуатации.
- Чтобы жизненный цикл не ломался, при разработке опираются на модели жизненного цикла. Самые строгие из них — каскадная и V-образная. При работе по этим моделям не получится перейти с одного этапа на предыдущий. Самые гибкие — Agile-методы. С ними можно выпускать недоработанное решение и улучшать его короткими итерациями.
- Чтобы упростить управление разработкой, можно использовать системы для управлениями проектами. Например, Kaiten подходит для работы с любой моделью SDLC. А среди возможностей таск-трекера: визуализацию с помощью канбан-досок, готовые доски для спринтов, диаграмма Ганта, инструменты для ведения и хранения документации по проекту.

Модели SDLC — не догма, можно совмещать подходы или на их основе создавать абсолютно новые, если они подходят вашей команде и дают результат. А если есть трудности с выбором, можно использовать простой инструмент Agile Suitability Filter. Он покажет, в сторону каких методологий стоит развиваться.

Теги: kaiten, планирование, sdlc, разработка по, разработка, проект, этапы, agile, жизненный цикл, kanban

Хабы: Блог компании Kaiten, Agile, Управление разработкой, IT-компании

11 мин 10K

+25 27 7

aabzel
12 часов назад

Детектор WiFi излучения

Простой 6 мин 11K

Туториал

+24 29 20

BHV_publishing
21 час назад

Как эволюционировала главная книга по Qt в России и чем удивляет 7-е издание

Простой 4 мин 9.5K

Обзор

+22 23 1

HappyTalkie
18 часов назад

Все мы родом из наших мультфильмов

Простой 7 мин 7.2K

+12 11 6

PyLounge
3 часа назад

manage.py migrate в пятницу в 17:30 на проде с 3K RPS и таблицей 200M строк

Средний 27 мин 3.9K

Туториал

+11 9 2

ASabramova
4 часа назад

Ловушка экспертности: почему мозг деградирует именно тогда, когда вы наконец всего достигли

Простой 4 мин 4.5K

Мнение

+11 11 2

lts_ryder
17 часов назад

Космический шредер: куда на самом деле деваются ваши биты информации в чёрной дыре?

Средний 4 мин 8.2K

Обзор

+10 8 10

aberglaube
3 часа назад

Почему технологии не сделали нас счастливее: ищем ответ в антиутопиях Стругацких

9 мин 4K

Обзор

+9 10 11

Baikalelec
19 часов назад

Как сделать процессор Российским

6 мин 7.9K

+8 13 23

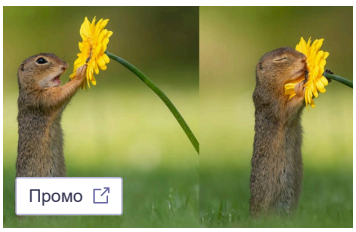
Как вернуть производительность цефа: перестаём гадать и разбираем узкие места

Турбо

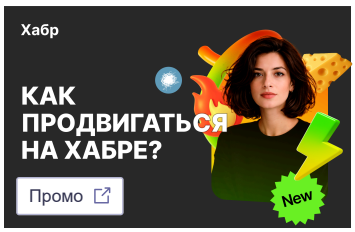
Показать еще

МИНУТОЧКУ ВНИМАНИЯ

Мы пять лет закрывали техдолг, а потом с LLM ускорились вдвое



Запахло весной и скидками в Промокодусе



Экскурсия по Хабру для маркетологов

ИНФОРМАЦИЯ

Сайт	kaiten.ru
Дата регистрации	9 октября 2023
Численность	51–100 человек
Местоположение	Россия

ССЫЛКИ

Telegram
t.me
Перейти в Kaiten
kaiten.ru

БЛОГ НА ХАБРЕ

5 мая в 13:00

Обзор 11 ITSM-систем для управления ИТ-услугами

8.1K 4

20 апр в 14:26

Обзор 10 CRM-систем: как выбрать сервис для бизнеса в 2026 году

6.2K 2

2 апр в 12:00

Как металлообрабатывающий завод ускорил выполнение задач в 3 раза с помощью Кайтена и ChatGPT

8.3K 16

19 мар в 18:12

Что такое канбан на практике: изучаем доски, WIP-лимиты и метрики

9.3K 4

10 мар в 14:38

Ресурсное планирование: как я пытался разобраться, почему сдвигаются сроки по проектам

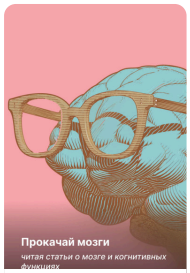
6.9K 3

ИСТОРИИ

Годнота из блогов компаний

Большие данные, алгоритмы и другие секреты ретейла

Захлестнуло радиоволной



Схватил за мозг

Ваш аккаунт

Войти
Регистрация

Разделы

Статьи
Новости
Хабы
Компании
Авторы
Песочница

Информация

Устройство сайта
Для авторов
Для компаний
Документы
Соглашение
Конфиденциальность

Услуги

Корпоративный блог
Медийная реклама
Нативные проекты
Образовательные программы
Стартапам

Настройка языка

Техническая поддержка