Введение в тестирование

**Тестирование в жизненном цикле разработки ПО**

# На этом уроке

* узнаем, что такое жизненный цикл ПО;
* поймём, из каких этапов состоит разработка ПО;
* узнаем о моделях жизненного цикла ПО;
* узнаем, какие методологии используются для разработки;
* познакомимся с жизненным циклом тестирования.

# Оглавление

[Глоссарий](#_4d34og8)

[Жизненный цикл ПО](#_cbxx168s8aua)

[Модели жизненного цикла ПО](#_2et92p0)

[Каскадная модель, или «водопад»](#_rnq1jypr95ip)

[V-модель](#_l9qdzmscvk51)

[Инкрементная модель](#_ml0a749vht3m)

[Итеративная модель](#_7ynqga2avga7)

[Итеративно-инкрементная модель](#_d0k89d2s4vze)

[Спиральная модель](#_njhd3s6ggm4z)

[Гибкие методологии разработки](#_tyjcwt)

[Использование гибких методологий в разработке ПО](#_4parqlnp5x5)

[Жизненный цикл тестирования ПО](#_3dy6vkm)

[Дополнительные материалы](#_2s8eyo1)

# Глоссарий

**Жизненный цикл ПО** — ряд событий, происходящих с системой в процессе её создания и дальнейшего использования.

**Модель жизненного цикла ПО** — описание того, как проходит процесс разработки ПО, какие этапы проходит ПО от рождения идеи до завершения использования и что происходит с ПО на этих этапах.

**Методология разработки** — набор методов по управлению разработкой ПО, практических правил и техник разработки. Или, по-другому, **методология** — детализированный набор правил, практик и принципов как способ реализации той или иной модели.

**Фреймворк процессов** — методология, содержащая большое количество правил; не обязательно использовать их все, выбираются лишь наиболее подходящие к требованиям проекта и процессу разработки. Имеют в своём составе специальные приложения, позволяющие просматривать и редактировать правила.

**Гибкие методологии разработки (Agile software development)** — обобщающий термин для целого ряда подходов и практик, основанных на ценностях манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе.

**Agile software development** — серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.

**SCRUM —** набор принципов, ценностей, политик, ритуалов, артефактов, на которых строится процесс SCRUM-разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами, предоставлять конечному пользователю работающий продукт с новыми бизнес-возможностями, для которых определён наибольший приоритет.

**Kanban** —методология гибкой разработки, в основе которойлежит быстрота поставки готового продукта на рынок, а также равномерное распределение нагрузки между членами команды. Процесс разработки прозрачен для всех участников.

# Жизненный цикл ПО

**Жизненный цикл в целом** — это ряд этапов, которые проходит живой организм, предмет или система в процессе развития, от момента появления до гибели. То же можно сказать про **жизненный цикл ПО** — это ряд событий, происходящих с системой в процессе её создания и дальнейшего использования.

Понятие жизненного цикла ПО, как и многие процессы управления разработкой ПО, было заимствовано из опыта индустриальных производств.

Жизненный цикл ПО состоит из ряда этапов, знание и понимание которых необходимо для дальнейшего знакомства с моделями жизненного цикла ПО.

Коротко определим, какие общие этапы включает в себя жизненный цикл ПО. Названия этапов, их последовательность и количество в разных источниках описывается по-разному. Но в целом они довольно похожи.



1. **Подготовка.** Этап от возникновения идеи о создании ПО до описания того, каким оно должно быть и какие функции выполнять. Также на этом этапе проводят сбор и обработку требований к ПО, предварительное планирование этапов работ, сроков, ресурсов и стоимости.
2. **Проектирование.** Получение технических заданий, разработка спецификаций. Заказчик получает документальное изложение своих требований и планы проведения работ.
3. **Создание:**

* дизайн — получение графических макетов, визуальных форм, разработка интерфейсов, создание индивидуального стиля;
* кодирование — получение исходного кода;
* тестирование — проверка программы на соответствие всем предъявляемым к ней требованиям;
* документирование (опционально) — передача накопленных знаний другим участникам команды разработки.

1. **Поддержка:**

* внедрение — установка ПО, обучение пользователей при необходимости;
* сопровождение — исправление выявленных ошибок, поддержка пользователей.

Иногда отдельно в жизненном цикле выделяют такой этап, как **прекращение использования** ПО.

**Внимание!** Следующие определения очень важны для понимания этой темы в целом. Часто в статьях на тему разработки ПО эти понятия путают, что может затруднить понимание процессов управления проектами.

Первое понятие — это **модель жизненного цикла ПО.**

**Модель жизненного цикла ПО** — описание того, как проходит процесс разработки. Именно в модели жизненного цикла ПО описывается, какие этапы проходит ПО, от рождения идеи до завершения использования, и что происходит с ПО на этих этапах. Очень важно понять, что **модель** — это прежде всего **этапы** разработки и их последовательность.

Другое важное понятие — **методология.**

**Методология разработки** — это набор методов по управлению разработкой ПО, набор практических правил и техник разработки. Или, по-другому, **методология** — это детализированный набор правил, практик и принципов, применяемый при реализации той или иной модели.

Модель жизненного цикла ПО нужно *знать для общего понимания* процесса разработки, а методологии — *применять на практике*, чтобы процесс разработки был максимально эффективным и последовательным.

Ещё одно понятие, которое может встретиться при знакомстве с материалами о методах и методологиях разработки ПО, — **фреймворк.**

**Фреймворк процессов** — это методология, содержащая большое количество правил. Необязательно использовать их все, можно выбрать только те, что нужны сейчас, и построить на их основе процесс разработки.

Рассмотрим модели жизненного цикла ПО, которых достаточно много. Не все они в настоящее время активно применяются на практике, какие-то более популярны, какие-то — менее. Но знать об их существовании будет полезно.

# Модели жизненного цикла ПО

Модели, которые могут использоваться при разработке ПО:

1. Code and fix (модель кодирования и устранения ошибок).
2. Waterfall Model (каскадная модель, или «водопад»).
3. V-model (V-образная модель, разработка через тестирование).
4. Incremental Model (инкрементная модель).
5. Iterative Model (итеративная, или итерационная, модель).
6. Spiral Model (спиральная модель).
7. The chaos model (модель хаоса).
8. Prototype Model (прототипная модель).

Рассматривать все не имеет смысла, поэтому остановимся на самых востребованных:

1. Каскадная модель («водопад»).
2. V-образная модель.
3. Итеративно-инкрементная модель.
4. Спиральная модель.

Начнём с модели, которая сейчас почти не применяется при разработке ПО, но именно с неё всё началось. Это модель Code and Fix. Название говорит само за себя и достаточно полно описывает её суть. На заре программирования руководители компаний, менеджеры и разработчики мало задумывались о необходимости составления качественных требований, установления сроков, выработки критериев приёмки конечного продукта. О тестировании, конечно, никто даже не помышлял. Программисты писали код, получали продукт, заказчик его оценивал, и, если его всё устраивало, продукт передавался пользователям. Если что-то было не так, то снова начиналась разработка… Это могло продолжаться до бесконечности, потому что не было приёмочных критериев и описания результата, который необходимо получить на выходе.

Естественно, такая модель разработки оказалась совершенно неконкурентоспособной при активном развитии IT-отрасли и быстром росте количества IT-проектов.

Попытки формализовать процесс разработки и задокументировать его привели в конечном итоге к появлению моделей, которые сейчас принято считать традиционными. И в первую очередь это известный многим «водопад».

## **Каскадная модель, или «водопад»**

Основная особенность этой модели — все этапы жизненного цикла выполняются последовательно и существуют определённые условия для перехода с одного этапа на другой. Невозможно начать следующий этап, не закончив предыдущий. Кроме того, после перехода на следующий этап либо невозможно вернуться назад, либо возврат назад очень сложен. Тестирование в данной модели начинается поздно.



Преимущества данной модели:

* стабильные, чётко сформулированные требования перед началом разработки;
* фиксированный порядок этапов, что позволяет провести качественное планирование разработки;
* у каждой стадии есть чёткий проверяемый результат;
* в каждый момент времени команда выполняет один вид работы.

Водопадная модель хорошо зарекомендовала себя при создании ПО, для которого в самом начале разработки можно достаточно точно и полно сформулировать все требования, а разработчики могут реализовать его как можно лучше с технической точки зрения.

Одним из существенных недостатков каскадной модели в современной разработке является позднее получение обратной связи как от тестировщиков на этапе проверки, так и от конечных пользователей. Это существенно повышает стоимость ошибки в процессе разработки.

|  |  |
| --- | --- |
| **+** | **-** |
| * простота в изучении; * легко контролировать; * может использоваться технически слабо подготовленными командами или неопытным персоналом; * раннее прогнозирование стоимости проекта | * позднее получение результата; * документация может быть избыточной; * позднее обнаружение конструктивных ошибок; * позднее представление результатов заказчику; * сложность возврата к предыдущим этапам; * отсутствие обратной связи от пользователей; * сложность изменений и сильное влияние этих изменений на сроки и стоимость проекта; * неравномерная загрузка участников проекта |

Использование:

* крупные проекты со стабильными требованиями (космическая отрасль, медицинское ПО);
* недорогие, несложные, средние проекты;
* наличие позитивного опыта разработки аналогичных систем;
* создание новой версии ранее разработанного продукта, когда вносимые изменения определены и управляемы;
* перенос уже существующего продукта на новую платформу.

## **V-модель**



Усовершенствованная водопадная модель. Здесь тестирование появляется уже в самом начале, и каждому этапу модели соответствует свой этап тестирования.

На этапе составления бизнес-требований формируется набор тестов для приёмочного тестирования. Этапу написания функциональных требований соответствует этап написания функциональных тестов. Этапу написания архитектуры системы соответствует этап написания интеграционных тестов. Этапу написания архитектуры компонентов соответствует этап написания модульных тестов.

|  |  |
| --- | --- |
| **+** | **-** |
| * простота в использовании; * планирование и проектирование тестирования на ранних этапах разработки; * верификация и аттестация всех внешних и внутренних полученных данных, а не только конечного продукта | * сложно вносить изменения; * нет действий, направленных на анализ рисков; * другие недостатки «водопада» |

Использование:

* управление разработкой ПО в немецкой администрации;
* стандарт для немецких правительственных и оборонных проектов;
* системы, в которых требуется высокая надёжность (прикладные программы для наблюдения за пациентами в клиниках);
* встроенное ПО для устройств управления аварийными подушками безопасности в автомобилях.

## **Инкрементная модель**

Основная особенность модели — требования заранее известны.

Разработка идёт «по частям» — инкрементам.

**Инкремент** — это постоянно увеличивающаяся величина. То есть продукт в процессе разработки постоянно увеличивается, «наращивается».

Первое, что создаётся в этой модели, — минимально работающий продукт, который можно быстро выпустить на рынок, получить обратную связь от пользователей, а затем продолжить его развивать, постепенно расширяя и улучшая функционал приложения.

Продукт в инкрементной модели проходит всё те же стадии (этапы) разработки ПО, что и в предыдущих моделях, но здесь они многократно повторяются для каждого нового инкремента.

|  |  |
| --- | --- |
| **+** | **-** |
| * позволяет уменьшить затраты на первоначальных этапах разработки (до достижения уровня исходной производительности); * ускоряет процесс создания функционирующей системы; * позволяет получать отзывы заказчика в процессе разработки; * снижает риск неудачи при изменении требований | * не всегда возможно определить полную функциональность ПО в начале разработки; * поскольку модули разрабатываются в разное время, необходимо чётко определить интерфейсы; * сложность приоритизации инкрементов; * может возникать тенденция к откладыванию сложных решений на будущее |

Использование:

* проекты, в которых требования сформулированы заранее и необходима быстрая поставка на рынок.

## **Итеративная модель**

Разработка идёт циклами — итерациями. Каждая итерация представляет из себя мини-«водопад». Результатом каждой итерации должна быть законченная часть функционала.

**Итерация** (лат. iteratio — «повторение») — временной интервал, в течение которого проходят все фазы разработки (анализ, проектирование, разработка, тестирование) и который многократно повторяется.

На начальном этапе в этой модели требования могут быть весьма условными или их может не быть совсем. В процессе разработки требования создаются, уточняются, изменяются в зависимости от потребностей заказчика, ситуации на рынке.

|  |  |
| --- | --- |
| **+** | **-** |
| * быстрый выпуск минимального продукта; * не требуется полной спецификации требований заранее; * снижение рисков на ранних стадиях; * эффективная обратная связь с заказчиком; * фокус на наиболее важных направлениях проекта; * непрерывное тестирование, которое позволяет оценить успешность всего проекта; * раннее обнаружение ошибок в требованиях; * равномерная загрузка участников проекта | * могут возникнуть проблемы с архитектурой; * отсутствие фиксированного бюджета и сроков; * отсутствие конечной цели, может привести к затягиванию проекта |

Использование:

* большие проекты с неопределёнными требованиями;
* проекты по разработке ПО, которое носит инновационный характер и основано на бизнес-гипотезах, требующих проверки.

Чаще всего в современной разработке применяют сочетание двух моделей — инкрементной и итеративной. Это позволяет сделать процесс разработки гибким и менее рискованным.

## **Итеративно-инкрементная модель**

Фундаментальная основа современного подхода к разработке ПО.

Если рассматривать её как одну модель, то налицо определённая двойственность:

* с точки зрения жизненного цикла модель является итерационной, так как подразумевает многократное повторение одних и тех же стадий;
* с точки зрения развития продукта (приращения его полезных функций) модель является инкрементальной.

Ключевая особенность данной модели — разбиение проекта на относительно небольшие промежутки (итерации), каждый из которых в общем случае может включать в себя все классические стадии, присущие водопадной и V-образной моделям. Итогом итерации является приращение (инкремент) функциональности продукта, выраженное в новой версии.

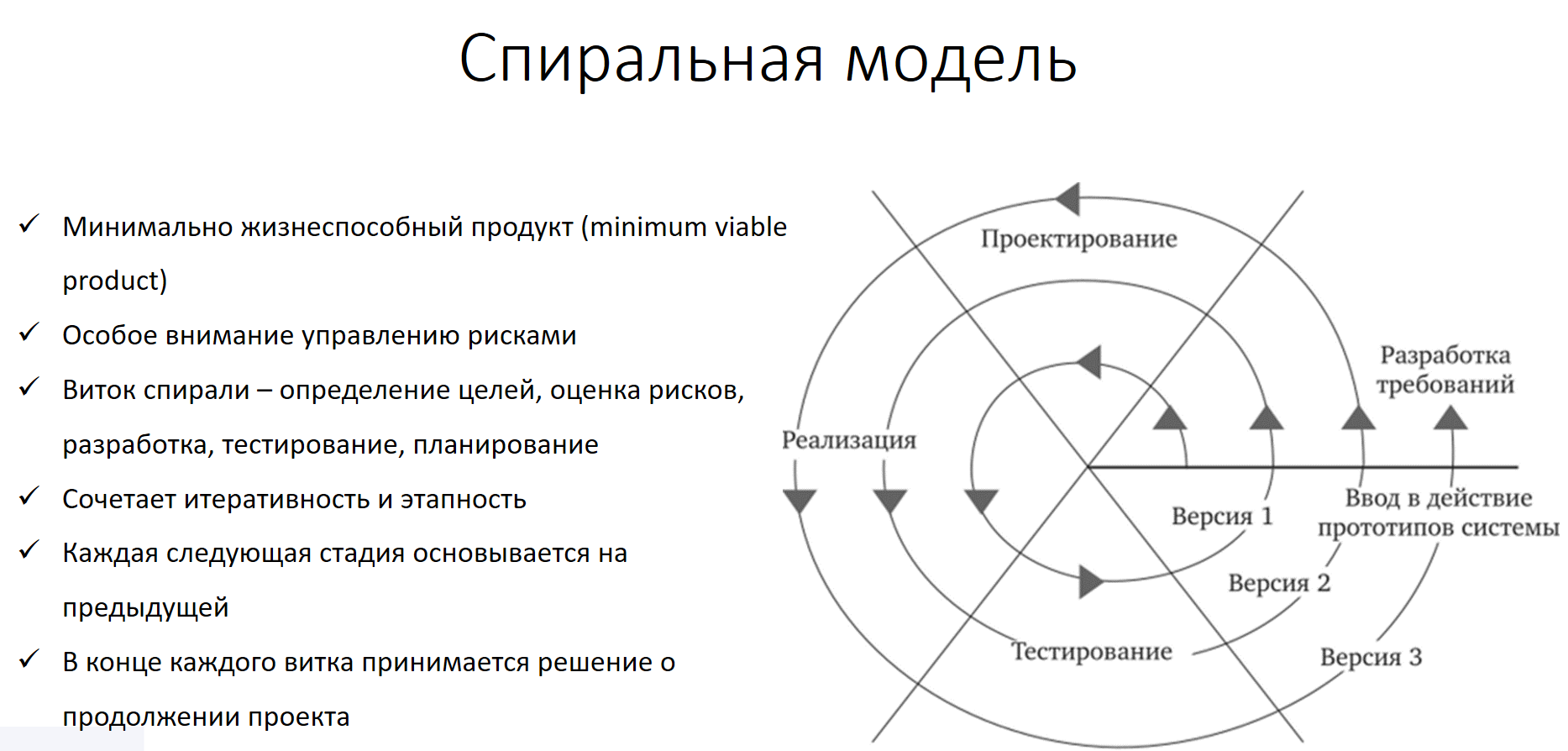
## ÐÐ°ÑÑÐ¸Ð½ÐºÐ¸ Ð¿Ð¾ Ð·Ð°Ð¿ÑÐ¾ÑÑ ÐÐ½ÐºÑÐµÐ¼ÐµÐ½ÑÐ½Ð¾-Ð¸ÑÐµÑÐ°ÑÐ¸Ð¾Ð½Ð½Ð°Ñ Ð¼Ð¾Ð´ÐµÐ»

## **Спиральная модель**

**Спиральная модель** **(spiral model)** представляет собой частный случай итерационно- инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки.

Схематично суть спиральной модели представлена на рисунке. Обратите внимание, что здесь явно выделены четыре ключевые фазы:

* проработка целей, альтернатив и ограничений;
* анализ рисков и прототипирование;
* разработка продукта (промежуточной версии);
* планирование следующего цикла.



Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, активизируя тем самым процесс уточнения и дополнения требований. То есть создать так называемый минимально жизнеспособный продукт (MVP — minimum viable product).

С точки зрения тестирования и управления качеством повышенное внимание рискам — ощутимое преимущество при использовании спиральной модели для разработки концептуальных проектов, в которых требования естественным образом будут сложными и нестабильными (могут многократно меняться по ходу выполнения проекта). Эта модель не подойдёт для малых проектов, но уместна для сложных и дорогих, таких как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование само по себе. Или проекты с высоким риском, или исследовательские проекты, или проекты, в которых изначально неизвестно, возможна ли их реализация в текущих условиях.

|  |  |
| --- | --- |
| **+** | **-** |
| * высокий уровень анализа рисков; * активное участие заказчиков, особенно на стадии планирования; * усовершенствование процесса на каждом витке | * высокая стоимость разработки; * сложно определить момент перехода на следующий виток спирали; * сложная структура, трудная для понимания; * спираль может продолжаться бесконечно |

**Использование:**

* проекты с высоким риском;
* исследовательские проекты;
* проекты, в которых изначально неизвестно, возможна ли их реализация в текущих условиях.

# Гибкие методологии разработки

Из-за большого количества разной информации и недопонимания часто можно встретить понятие Agile среди моделей или методологий разработки. Но это ошибочно: Agile — не модель и не методология. Это скорее философия разработки, на которую опираются гибкие методологии. Это важно понимать, как и разницу между моделью и методологией. Agile объединяет в себе разные гибкие методологии разработки, такие как:

* SCRUM;
* Kanban;
* Lean;
* XP — экстремальное программирование.

Гибкие методологии разработки чаще всего основываются на итеративно-инкрементной модели.

**Гибкие методологии разработки (Agile software development)** — обобщающий термин для целого ряда подходов и практик, основанных на ценностях манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе.

**Agile software development** — серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля.

**Agile-манифест** —основные идеи, на основании которых выстроены процессы в гибких методологиях разработки:

* люди и их взаимодействие важнее процессов и инструментов;
* готовый продукт важнее документации по нему;
* сотрудничество с заказчиком важнее жёстких контрактных ограничений;
* реакция на изменения важнее следования плану.

Важно помнить, что Agile не отрицает важности того, что процессы, инструменты, документация и прочее также важны для разработки качественного ПО, но приоритет всё же отдается идеям Agile-манифеста.

Кроме манифеста существуют базовые принципы, которые лежат в основе гибкой разработки. Основные из них:

1. Наивысший приоритет — удовлетворение потребностей заказчика благодаря регулярной и ранней поставке нового функционала. Чем раньше заказчик увидит новую функциональность, тем быстрее команда проекта получит обратную связь и скорректирует свою работу.
2. Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до нескольких месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе — чтобы получать обратную связь, корректировать действия команды и разрешать возникшие трудности и недопонимания в процессе разработки.
5. Над проектом должны работать мотивированные профессионалы. Этот пункт очень важен для гибких методологий, так как процессы в них менее формализованы, нет чётких критериев оценки работы, отчётов о рабочем времени.
6. Непосредственное общение — наиболее важный и эффективный способ обмена информацией участников команды, а также команды и заказчика.
7. Работающий продукт — основной показатель прогресса.
8. Инвесторы, разработчики и другие члены команды должны иметь возможность поддерживать постоянный ритм работы. Так как работа над проектом обычно занимает много времени (от нескольких месяцев до нескольких лет), необходимо создавать условия работы и организовывать процессы таким образом, чтобы команда работала без сбоев, авралов, чтобы не происходило профессионального выгорания членов команды и всем было комфортно работать.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта — для долгосрочной работы нужно, чтобы приложение было качественно и технически грамотно разработано.
10. Простота — искусство минимизации лишней работы — очень важна для проектов с гибкой разработкой. Не нужно выполнять ту работу, которую можно не делать (например, составлять детальный план тестирования).
11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и корректировать процессы и стиль работы.

Особенности Agile-команд:

* при планировании работ учитываются интересы всех членов команды;
* работа над завершением задач более приоритетного уровня идёт в первую очередь;
* признание проблем;
* члены команды помогают друг другу;
* тестировщик — полноценный член команды.

**Внимание!** Важно помнить — за качество отвечает вся команда!

**Agile** —подходы к созданию ПО путём непрерывной, быстрой поставки ценного рабочего функционала самоорганизующейся командой профессионалов в сотрудничестве с заказчиком.

|  |  |
| --- | --- |
| **+** | **-** |
| * быстрая поставка рабочего функционала; * взаимодействие с заказчиком; * возможность получения быстрой обратной связи от конечных пользователей; * адаптируемость к изменению внешних условий; * возможность корректировки требований в процессе разработки | * сложно применять в непростых проектах; * опасность возникновения неучтённых рисков; * может быть недостаточно документации |

## **Использование гибких методологий в разработке ПО**

Согласно ежегодному отчету Agile-комьюнити, самой используемой гибкой методологией разработки в 2018 году был SCRUM, а также «гибридные» методологии на его основе.

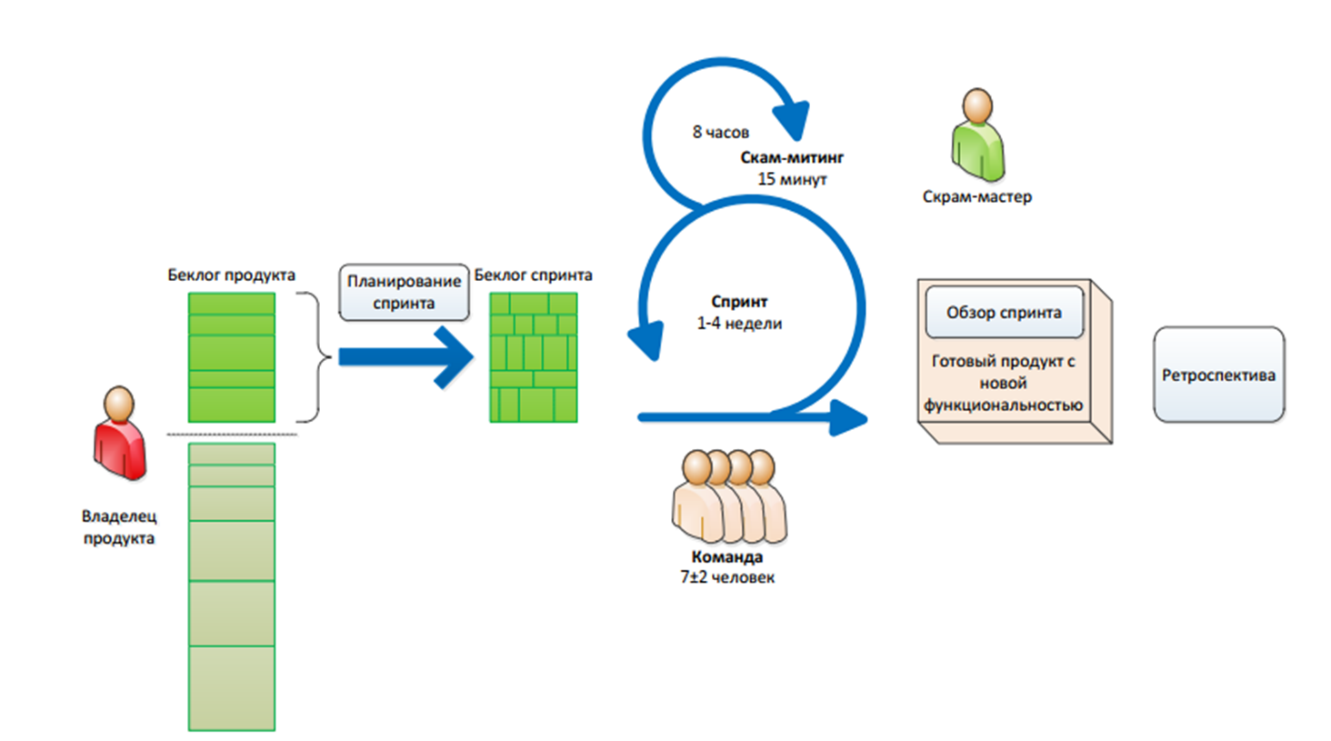
**SCRUM** —набор принципов, ценностей, политик, ритуалов, артефактов, на которых строится процесс SCRUM-разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами, предоставлять конечному пользователю работающий продукт с новыми бизнес-возможностями, для которых определён наибольший приоритет.

Основные роли в SCRUM:

* скрам-мастер (SCRUM master) — следит, чтобы команда работала по SCRUM, и помогает решать возникающие проблемы;
* владелец продукта (product owner) — заказчик или человек, который определяет основные цели в реализации проекта;
* команда разработки (development team).

Скрам-мастером может выступать любой член команды, который хорошо понимает, как организуются процессы в SCRUM, и может ими управлять.

Владелец продукта определяет все задачи, которые должна выполнить команда проекта на протяжении разработки. Эти задачи формируют бэклог продукта.



**Из чего состоит SCRUM**

* итерации или спринты;
* планирование итерации (sprint planning meeting);
* ежедневный стендап (standup);
* демо (demo);
* ретроспектива (retro);
* доска задач (SCRUM board).

Итерация или спринт в SCRUM — это промежуток времени от недели до месяца, в течение которого происходит разработка работоспособной части продукта или новой функциональности. В конце каждой итерации команда должна предоставить заказчику либо другому заинтересованному лицу готовый инкремент продукта. Принятая на проекте длина итерации должна быть постоянной, чаще всего она составляет две недели.

Перед стартом каждой новой итерации происходит планирование, то есть определение набора задач, который планируется реализовать в процессе итерации. Обычно это называется бэклогом спринта.

В итерацию (спринт) берётся часть функций, которые необходимо реализовать в этом спринте. Эти функции уже формируют бэклог спринта. Чтобы сформировать бэклог спринта, в начале каждой итерации проводится встреча всех членов команды, а также скрам-мастера, владельца продукта и других заинтересованных лиц. Эта встреча называется планированием спринта. На ней обсуждаются все задачи и цели команды и определяется, какой объём работы может быть выполнен командой в процессе предстоящего спринта. Все члены команды должны быть согласны с объёмом работы, который может быть предварительно оценён в человеко-часах, стори-поинтах либо каким-то другим способом, предусмотренным на проекте. Существуют разные варианты оценки задач, которые формируют бэклог спринта, подробнее о них можно прочитать в руководстве по SCRUM (ссылка на него — в дополнительных материалах).

Все задачи, запланированные для спринта, должны быть приоритизированы и выполнены в том спринте, в бэклог которого они включены.

Во время прохождения спринта важным организационным мероприятием становится ежедневный скрам-митинг. Это встреча всей команды для обсуждения текущей ситуации на проекте. Основная её цель — оценка прогресса команды и оперативное выявление возникающих проблем. Обычно такая встреча не должна длиться больше 15 минут.

После спринта проводится демонстрация полученных результатов — демо. На демонстрацию приглашаются все заинтересованные лица и владелец продукта. На встрече команда показывает владельцу продукта результат и получает рекомендации для дальнейшей работы. На основании проведенного демо владелец продукта может скорректировать весь бэклог продукта.

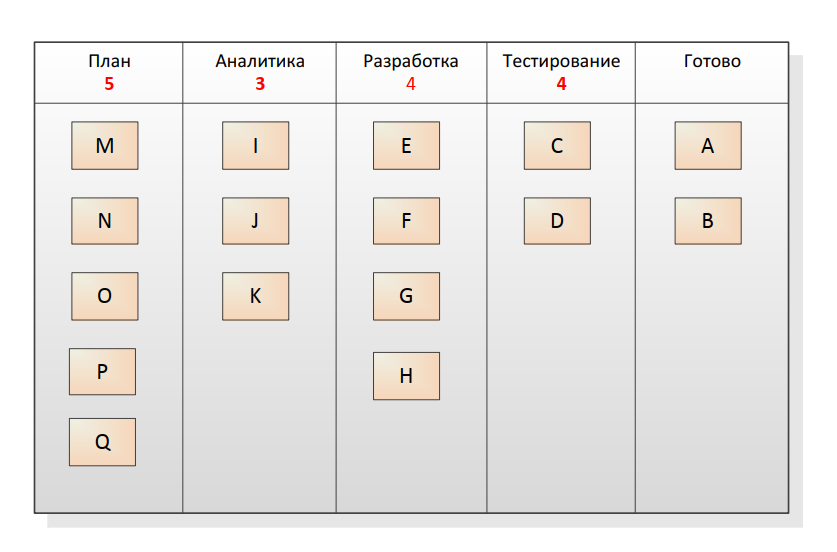
Отдельно для команды проекта в SCRUM предусмотрено проведение ретроспективы по итогам спринта и демо. На встрече подводятся итоги спринта, обсуждаются возникшие проблемы либо полученные успехи.

Для визуализации задач и контроля прогресса в SCRUM используется доска, на которой размещаются все задачи, взятые в бэклог спринта.

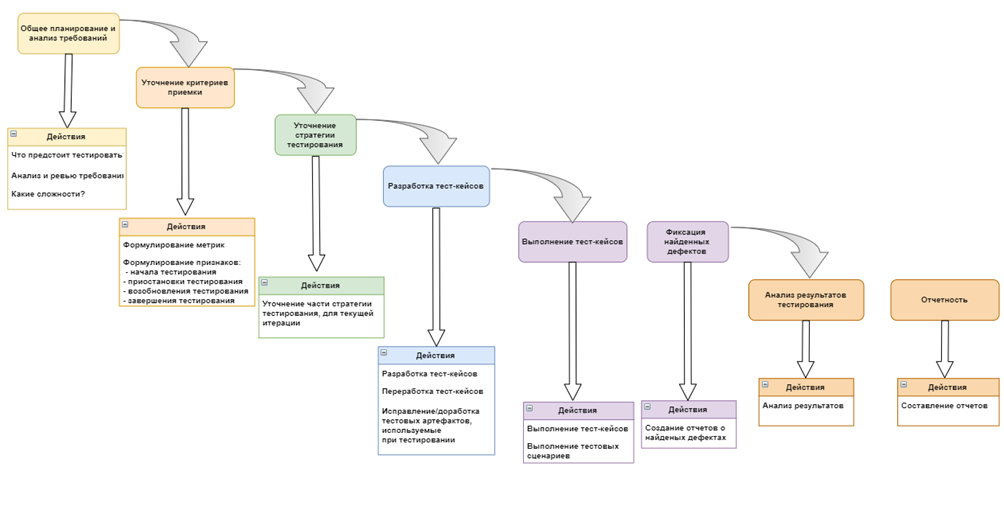
**Kanban** —методология гибкой разработки, в основе которойлежит быстрота поставки готового продукта на рынок, а также равномерное распределение нагрузки между членами команды. Процесс разработки прозрачен для всех участников.

Основное преимущество разработки по методологии Kanban — скорость поставки законченной функциональности пользователю. Здесь нет спринтов и каких-либо других временных интервалов. Время от появления у заказчика идеи до выхода готовой функциональности пользователю может быть минимальным. Например, заказчик хочет изменить процедуру оплаты товара, приобретённого пользователем в интернет-магазине. Он сообщает об этом аналитику команды, аналитик создаёт задачу для разработчика и назначает её наивысший приоритет, разработчик реализует новую функциональность и сразу передаёт её на тестирование. Если в ходе тестирования не будет выявлено критических или средних дефектов, то новая функциональность сразу же устанавливается на продуктовые серверы и пользователи могут с ней работать.

Основной инструмент в Kanban — доска. Главная её особенность — ограничение задач, которые могут быть одновременно в разработке или тестировании. Поэтому важно следить, чтобы в процессе работы не возникало «бутылочных горлышек» и большое количество задач не скапливалось на одном этапе разработки.



# Жизненный цикл тестирования ПО



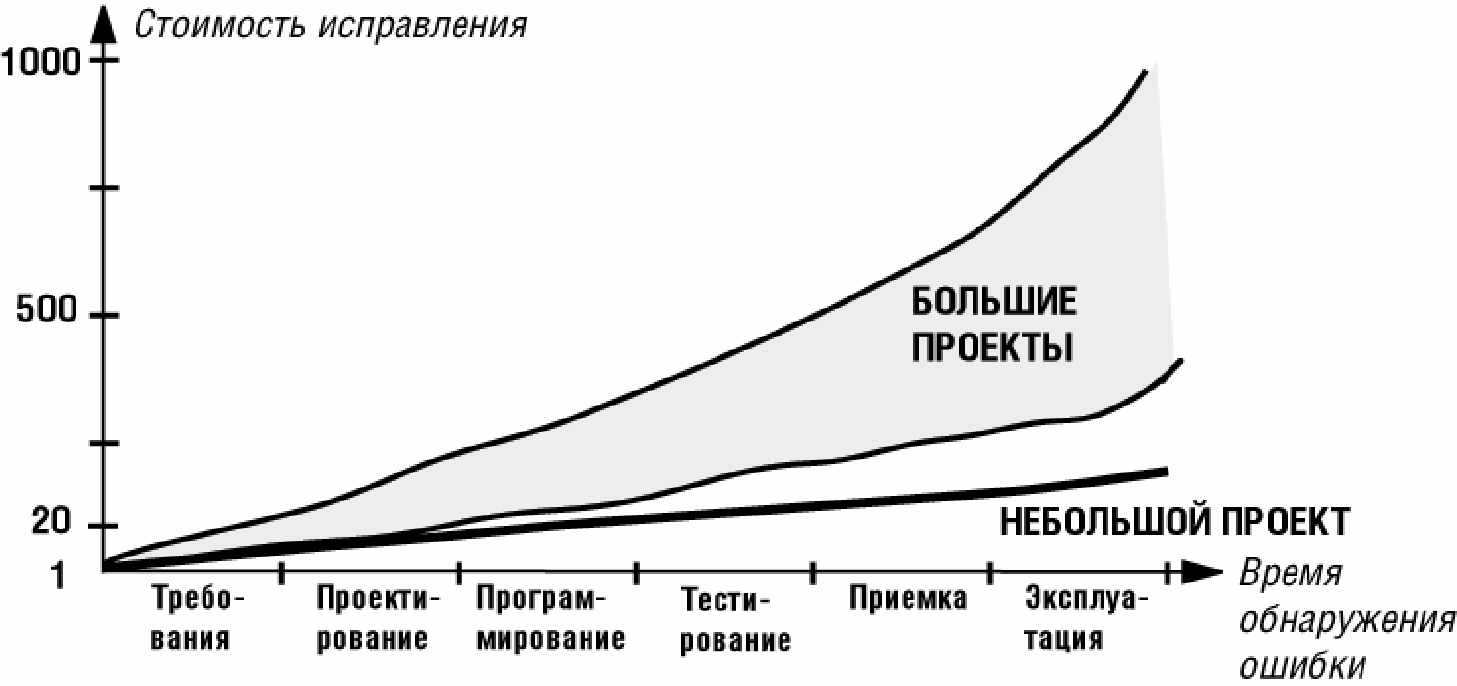
Жизненный цикл тестирования связан с моделью разработки, которая применяется на проекте. Если разработка ведётся по каскадной модели, тестирование проводится один раз на соответствующем этапе и все фазы его жизненного цикла повторяются только один раз. В итерационно-инкрементной модели этапы циклично повторяются на протяжении проекта. В этом случае длина жизненного цикла тестирования может зависеть от длины итерации либо от фазы разработки, на которой проект находится в определённый момент времени. Например, на начальном этапе разработки больше внимания уделяется тестированию требований, ближе к окончанию проекта — составлению и анализу отчётов по результатам тестирования, а также приёмочным тестам.

Жизненный цикл тестирования состоит из следующих этапов:

1. **Общее планирование и анализ требований.** На этом этапе необходимо определиться, что конкретно предстоит тестировать (какой продукт, приложение, из какой предметной области, с каким набором технологий), ожидаемый объём работы, возможные риски. Чтобы оценить перечисленное выше, необходимо провести анализ требований (если они есть) либо планирование другим возможным способом.
2. **Уточнение критериев приёмки.** Перед началом тестирования важно определиться с основными критериями оценок (метрик). Необходимо точно указать метрики и признаки, руководствуясь которыми можно начинать тестирование, метрики, на основании которых можно приостановить тестирование (например, количество дефектов к объёму пройденных тестов) и снова его возобновить, а также критерии завершения тестирования. Более подробно метрики будут рассмотрены в следующих лекциях.
3. **Уточнение стратегии тестирования.** В зависимости от стадии или этапа проекта уточняются стратегии тестирования, чтобы они были актуальны для текущей итерации.
4. **Разработка тест-кейсов.** Разработка, написание, корректировка и прочие действия с тест-кейсами, их наборами и иными тестовыми артефактами, которые будут использоваться в процессе тестирования.
5. **Выполнение тест-кейсов.** Этап непосредственного тестирования. Тесно связан со следующим этапом.
6. **Фиксация найденных дефектов.** Дефекты, обнаруженные в процессе прохождения тест-кейсов либо иным образом, должны быть обязательно зафиксированы в системе хранения дефектов (баг-трекере).
7. **Анализ результатов тестирования.** При проведении анализа результатов важно опираться на метрики и критерии, сформированные на этапах 1, 2 и 3, а также на данные тест-планов. Полученные выводы оформляются на следующей стадии.
8. **Отчётность.** Подведение итогов проведённого тестирования. В итерационно-инкрементной модели выводы, сформулированные на этом этапе, служат основой для стадий 1, 2 и 3 в следующей итерации тестирования.

Ещё раз напомним, что каждый проект уникален и приведённая схема — не эталон. Некоторые этапы могут отсутствовать в процессе разработки и тестирования. Но в целом это наиболее полное и универсальное представление жизненного цикла тестирования.

В заключение стоит сказать несколько слов о необходимости тестирования на начальных этапах разработки. Конечно, тестирование важно на протяжении всего проекта. Но именно на начальной стадии, при тестировании требований, можно обнаружить критические ошибки, например в логике разрабатываемой системы, и тем самым избежать их. Поэтому нужно помнить о повышении стоимости ошибки в процессе разработки. Чем позднее была обнаружена ошибка, тем сложнее, дольше и дороже будет её исправление.



Таким образом, при создании нового технологического продукта нужно хорошо понимать, какие модели разработки существуют и какие методологии используются, чтобы учитывать особенности выбранной модели, понимать её слабые стороны и строить процессы тестирования более эффективно.

# Дополнительные материалы

[Руководство по SCRUM](https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Russian.pdf)

[Основы Agile](https://vimeo.com/123181716)

[Методологии разработки программного обеспечения](https://zen.yandex.ru/media/id/5b7ae22633ef9b00a8cc79f3/metodologii-razrabotki-programmnogo-obespecheniia-5b8ea7265102dd00aaed8fcb)

[Борис Вольфсон. Гибкие методологии разработки](https://habr.com/ru/post/137554/)