**Этапы жизненного цикла ПО**

У любого программного обеспечения есть жизненный цикл — этапы, через которые оно проходит с начала создания до конца разработки и внедрения. Чаще всего это подготовка, проектирование, создание и поддержка. Этапы могут называться по-разному и дробиться на более мелкие стадии.



Рассмотрим эти этапы на примере жизненного цикла интернет-магазина.

**Подготовка.** Иван решил запустить книжный интернет-магазин и начал анализировать, какие подобные сайты уже представлены в сети. Собрал информацию об их трафике, функциональности.

**Проектирование.** Иван выбрал компанию-подрядчика и обсудил с её специалистами архитектуру и дизайн будущего интернет-магазина.

**Создание.** Иван заключил с разработчиками договор. Они начали писать код, отрисовывать дизайн, составлять документацию.

**Поддержка.** Иван подписал акт сдачи-приёмки, и подрядчик разместил интернет-магазин на «боевых» серверах. Пользователи начали его посещать и сообщать о замеченных ошибках в поддержку, а программисты — оперативно всё исправлять.

*Модель*разработки программного обеспечения описывает, какие стадии жизненного цикла оно проходит и что происходит на каждой из них.

А*методология* включает в себя набор методов по управлению разработкой: это правила, техники и принципы, которые делают её более эффективной.

**Основные модели разработки ПО**

* Code and fix — модель кодирования и устранения ошибок;
* Waterfall Model — каскадная модель, или «водопад»;
* V-model — V-образная модель, разработка через тестирование;
* Incremental Model — инкрементная модель;
* Iterative Model — итеративная (или итерационная) модель;
* Spiral Model — спиральная модель;
* Chaos model — модель хаоса;
* Prototype Model — прототипная модель.

Из этих моделей наиболее популярны пять основных: каскадная, V-образная, инкрементная, итерационная и спиральная. Разберём их подробнее.

**Waterfall (каскадная модель, или «водопад»)**

В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. Если всё делать правильно, «водопад» будет наиболее быстрой и простой моделью. Применяется уже почти полвека, с 1970-х.



**Преимущества «водопада»**

* *Разработку просто контролировать.* Заказчик всегда знает, чем сейчас заняты программисты, может управлять сроками и стоимостью.
* *Стоимость проекта определяется на начальном этапе.* Все шаги запланированы уже на этапе согласования договора, ПО пишется непрерывно «от и до».
* *Не нужно нанимать тестировщиков с серьёзной технической подготовкой.* Тестировщики смогут опираться на подробную техническую документацию.

**Недостатки каскадной модели**

* *Тестирование начинается на последних этапах разработки.* Если в требованиях к продукту была допущена ошибка, то исправить её будет стоить дорого. Тестировщики обнаружат её, когда разработчик уже написал код, а технические писатели — документацию.
* *Заказчик видит готовый продукт в конце разработки и только тогда может дать обратную связь.*Велика вероятность, что результат его не устроит.
* *Разработчики пишут много технической документации, что задерживает работы.* Чем обширнее документация у проекта, тем больше изменений нужно вносить и дольше их согласовывать.

«Водопад» подходит для разработки проектов в *медицинской и космической отрасли, где уже сформирована обширная база документов*(СНиПов и спецификаций), на основе которых можно написать требования к новому ПО.

При работе с каскадной моделью основная задача — написать подробные требования к разработке. На этапе тестирования не должно выясниться, что в них есть ошибка, которая влияет на весь продукт.

**V-образная модель (разработка через тестирование)**

Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. История этой модели начинается в 1980-х.



**Преимущества V-образной модели**

* Количество ошибок в архитектуре ПО сводится к минимуму.

**Недостатки V-образной модели**

* Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде».

V-модель подходит *для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках.*

**Incremental Model (инкрементная модель)**

Это модель разработки по частям (increment в переводе с англ. — приращение) уходит корнями в 1930-е. Рассмотрим её на примере создания социальной сети.

1. Заказчик решил, что хочет запустить соцсеть, и написал подробное техническое задание. Программисты предложили реализовать основные функции — страницу с личной информацией и чат. А затем протестировать на пользователях, «взлетит или нет».
2. Команда разработки показывает продукт заказчику и выпускает его на рынок. Если и заказчику, и пользователям социальная сеть нравится, работа над ней продолжается, но уже по частям.
3. Программисты параллельно создают функциональность для загрузки фотографий, обмена документами, прослушивания музыки и других действий, согласованных с заказчиком. Инкремент за инкрементом они совершенствуют продукт, приближаясь к описанному в техническом задании.



**Преимущества инкрементной модели**

* *Не нужно вкладывать много денег на начальном этапе.* Заказчик оплачивает создание основных функций, получает продукт, «выкатывает» его на рынок — и по итогам обратной связи решает, продолжать ли разработку.
* *Можно быстро получить фидбэк от пользователей и оперативно обновить техническое задание.* Так снижается риск создать продукт, который никому не нужен.
* *Ошибка обходится дешевле.*Если при разработке архитектуры была допущена ошибка, то исправить её будет стоить не так дорого, как в «водопаде» или V-образной модели.

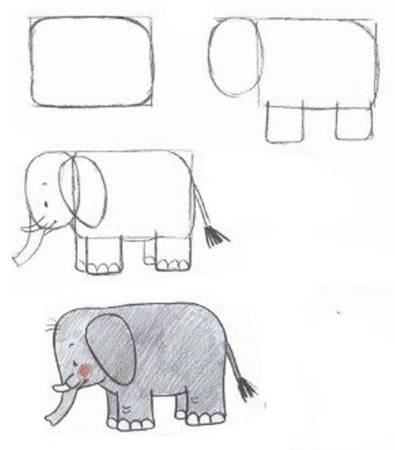
**Недостатки инкрементной модели**

* *Каждая команда программистов разрабатывает свою функциональность и может реализовать интерфейс продукта по-своему.* Чтобы этого не произошло, важно на этапе обсуждения техзадания объяснить, каким он будет, чтобы у всех участников проекта сложилось единое понимание.
* *Разработчики будут оттягивать доработку основной функциональности и «пилить мелочёвку».* Чтобы этого не случилось, менеджер проекта должен контролировать, чем занимается каждая команда.

Инкрементная модель подходит для *проектов, в которых точное техзадание прописано уже на старте, а продукт должен быстро выйти на рынок.*

**Iterative Model (итеративная модель)**

Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.



Рассмотрим на примере создания мессенджера, как эта модель работает.

1. Заказчик решил, что хочет создать мессенджер. Разработчики сделали приложение, в котором можно добавить друга и запустить чат на двоих.
2. Мессенджер «выкатили» в магазин приложений, пользователи начали его скачивать и активно использовать. Заказчик понял, что продукт пользуется популярностью, и решил его доработать.
3. Программисты добавили в мессенджер возможность просмотра видео, загрузки фотографий, записи аудиосообщений. Они постепенно улучшают функциональность приложения, адаптируют его к требованиям рынка.

**Преимущества итеративной модели**

* *Быстрый выпуск минимального продукта*даёт возможность оперативно получать обратную связь от заказчика и пользователей. А значит, фокусироваться на наиболее важных функциях ПО и улучшать их в соответствии с требованиями рынка и пожеланиями клиента.
* *Постоянное тестирование пользователями* позволяет быстро обнаруживать и устранять ошибки.

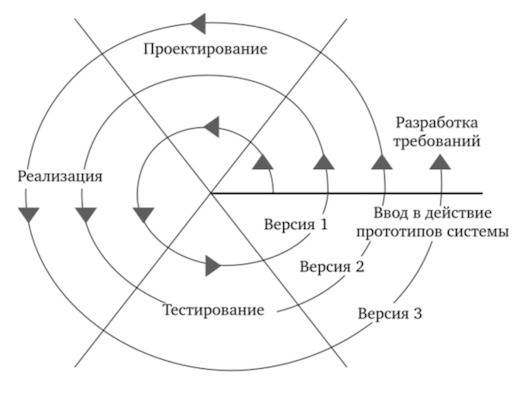
**Недостатки итеративной модели**

* *Использование на начальном этапе баз данных или серверов*— первые сложно масштабировать, а вторые не выдерживают нагрузку. Возможно, придётся переписывать большую часть приложения.
* *Отсутствие фиксированного бюджета и сроков.* Заказчик не знает, как выглядит конечная цель и когда закончится разработка.

Итеративная модель подходит для работы над *большими проектами с неопределёнными требованиями*, либо для задач с *инновационным подходом,*когда заказчик не уверен в результате.

**Spiral Model (спиральная модель)**

Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект. Эту модель начали использовать в 1988 году.



Рассмотрим, как функционирует эта модель, на примере разработки системы «Умный дом».

1. Заказчик решил, что хочет сделать такую систему, и заказал программистам реализовать управление чайником с телефона. Они начали действовать по модели «водопад»: выслушали идею, провели анализ предложений на рынке, обсудили с заказчиком архитектуру системы, решили, как будут её реализовывать, разработали, протестировали и «выкатили» конечный продукт.
2. Заказчик оценил результат и риски: насколько нужна пользователям следующая версия продукта — уже с управлением телевизором. Рассчитал сроки, бюджет и заказал разработку. Программисты действовали по каскадной модели и представили заказчику более сложный продукт, разработанный на базе первого.
3. Заказчик подумал, что пора создать функциональность для управления холодильником с телефона. Но, анализируя риски, понял, что в холодильник сложно встроить Wi-Fi-модуль, да и производители не заинтересованы в сотрудничестве по этому вопросу. Следовательно, риски превышают потенциальную выгоду. На основе полученных данных заказчик решил прекратить разработку и совершенствовать имеющуюся функциональность, чтобы со временем понять, как развивать систему «Умный дом».

Спиральная модель похожа на инкрементную, но здесь гораздо больше времени уделяется оценке рисков. С каждым новым витком спирали процесс усложняется. Эта модель часто используется в *исследовательских проектах и там, где высоки риски.*

**Преимущества спиральной модели**

* *Большое внимание уделяется проработке рисков.*

**Недостатки спиральной модели**

* *Есть риск застрять на начальном этапе*— бесконечно совершенствовать первую версию продукта и не продвинуться к следующим.
* *Разработка длится долго и стоит дорого.*

На основе итеративной модели была создана Agile — не модель и не методология, а скорее подход к разработке.

**Что такое Agile?**

Agile («эджайл») переводится с английского как «гибкий». Включает в себя практики, подходы и методологии, которые помогают создавать продукт более эффективно:

* экстремальное программирование (Extreme Programming, XP);
* бережливую разработку программного обеспечения (Lean);
* фреймворк для управления проектами Scrum;
* разработку, управляемую функциональностью (Feature-driven development, FDD);
* разработку через тестирование (Test-driven development, TDD);
* методологию «чистой комнаты» (Cleanroom Software Engineering);
* итеративно-инкрементальный метод разработки (OpenUP);
* методологию разработки Microsoft Solutions Framework (MSF);
* метод разработки динамических систем (Dynamic Systems Development Method, DSDM);
* метод управления разработкой Kanban.

Различия между Agile и традиционным подходом к разработке мы свели в таблице:



Не всё перечисленное в списке — методологии. Например, Scrum чаще называют не методологией, а фреймворком. В чём разница? Фреймворк — это более сформированная методология со строгими правилами. В скраме все роли и процессы чётко прописаны. Помимо Scrum, часто используют Kanban.

**Kanban**

Сегодня это одна из наиболее популярных методологий разработки ПО. Команда ведёт работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания.



В отличие от скрама, в канбане можно взять срочные задачи в разработку сразу, не дожидаясь начала следующего спринта. Канбан удобно использовать не только в работе, но и в личных целях — распределять собственные планы или задачи семьи на выходные, наглядно отслеживать прогресс.

