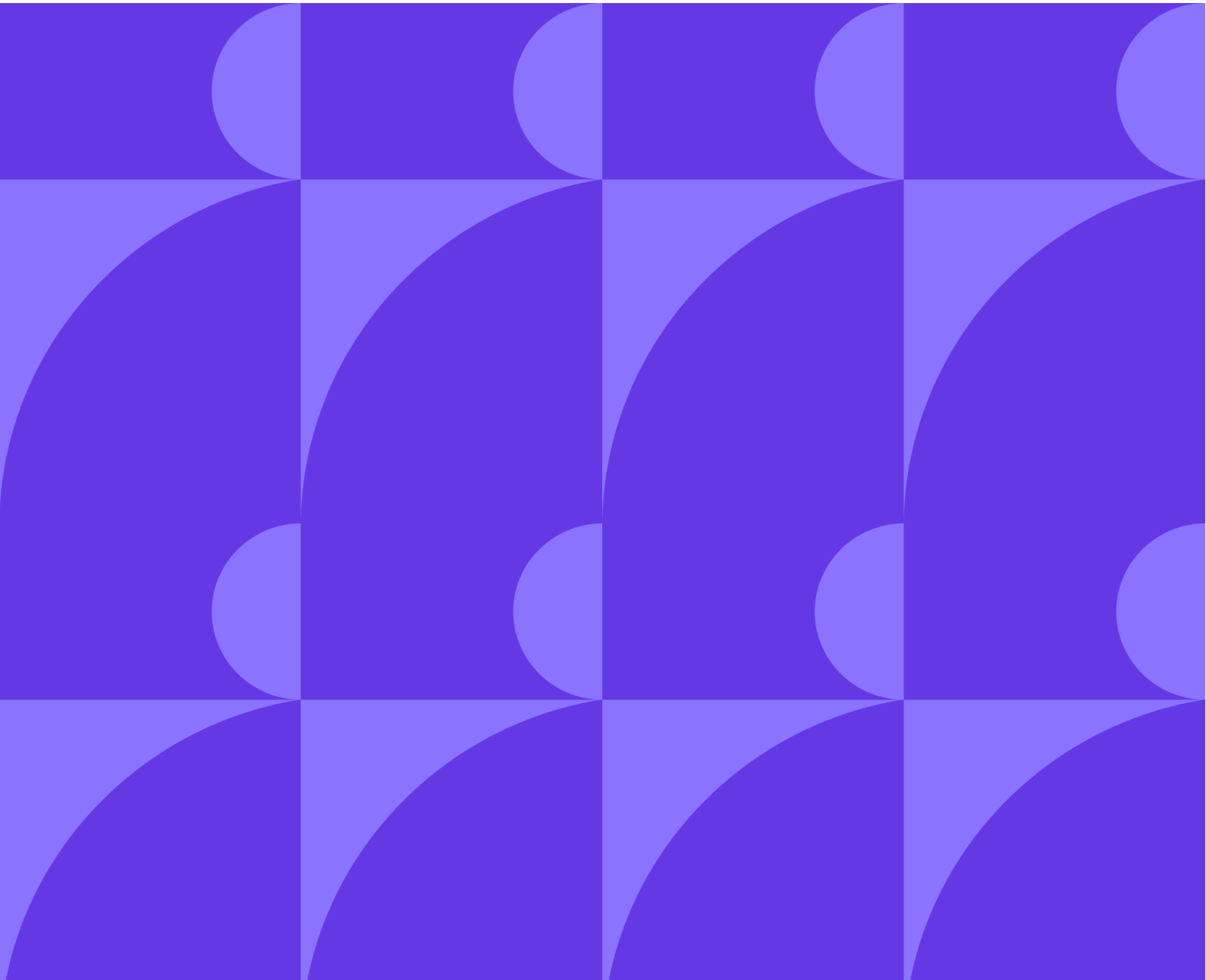
Гибкие методологии

Гибкие методологии. История возникновения



На этом уроке

1. Узнаем, как происходила эволюция методологий разработки ПО.
2. Поговорим про IT-продукт и его жизненный цикл.
3. Рассмотрим, какие бывают модели и методологии управления разработкой ПО.
4. Поговорим о плюсах и минусах традиционных и гибких методологий.
5. Обсудим отличия традиционных подходов к разработке ПО от гибких.

Оглавление

Глоссарий

IT (Information technology) – информационные технологии

ПО – программное обеспечение

Теория урока

1. IT-продукт и этапы его жизненного цикла
2. Эволюция возникновения моделей разработки продукта
    2.1. Первый кризис программного обеспечения
    2.2. Изменение взглядов на разработку софта
3. Классификация моделей разработки
    3.1. «Waterfall Model» (каскадная модель или «водопад»)
    3.2. V-образная модель (разработка через тестирование)
    3.3. Спиральная модель
    3.4. Инкрементная модель
    3.5. Итеративная или итерационная модель
    3.6. Agile
    3.7. Scrum
    3.8. Kanban
4. Отличия гибких методологий от традиционных (жёстких)
5. Выводы

IT-продукт и этапы его жизненного цикла

На протяжении всего обучения мы часто будем использовать слово

«IT-продукт» или просто **«Продукт».**

Давайте разберёмся, что это.

IT-продукт – это совокупность программно-аппаратных средств (программное решение, основанное на технологической базе), которое предоставляет выполнение заложенных функций, определяющих его суть. Также IT-продукт имеет себестоимость и может быть продан (внедрён) или сдан в аренду на определённый срок конечному потребителю.

Если простыми словами:

IT-продукт – это любое программное обеспечение, сайт, мобильное приложение и другая IT-система, которая разрабатывается и внедряется для выполнения определённых функций.

Погрузиться сразу в процесс изучения гибких методологий невозможно — давайте делать это постепенно.

К примеру, у вас зародилась идея создать удобный сайт для своего интернет-магазина, чтобы любой желающий мог выбирать понравившиеся ему вещи, заказывать, получать доставку, оплачивать всё это прямо на сайте и вообще совершать кучу различных операций. Звучит круто, не правда ли? И даже больше: вы уже видите свой удобный и быстрый в работе крутой сайт, видите, как ваши будущие покупатели вещей легко пролистывают страницы, как летят заказы, логисты отгружают товар, а курьеры бегают с заказами по домам довольных покупателей. НО это всё лишь вершина айсберга, а под водой находится куча потраченного времени, сложностей, провалов, неудач и много чего ещё интересного. В общем, сам процесс создания любого IT-продукта, будь то интернет-магазин, доработка сайта, какое-то мобильное приложение банка или ещё что-то — скрыт от наших глаз.

Именно поэтому для начала хотелось бы вас погрузить в процесс создания и использования IT-продукта. Это называется **жизненный цикл продукта.**

Жизненный цикл продукта (программного обеспечения) — этапы, через которые IT-продукт проходит от начала создания до конца разработки и внедрения в бизнес-среду. Кратко можно выделить следующие основные этапы:

1. Подготовка
2. Проектирование
3. Создание
4. Поддержка

⚠️ **Внимание!**

Этапы могут называться по-другому и дробиться («декомпозироваться») на более мелкие стадии.

Рассмотрим этапы жизненного цикла IT-продукта на примере проекта по созданию интернет-магазина.

1. **Подготовка** Вы занимаетесь производством одежды или у вас есть оптовые поставщики и Вы хотите начать зарабатывать и, конечно же, хотите делать всё это быстро и эффективно. Первое, что вам приходит на ум — запустить

интернет-магазин одежды. И это только небольшое словосочетание — «интернет-магазин». Первым делом вы начинаете анализировать конкурентов: на рынке полно интернет-магазинов одежды, сайты одних очень круты и многофункциональны, других — наоборот, пестрят багами и кучей недоработок. Естественно, вы хотите сделать лучший! Вы собираете информацию о трафике сайтов-конкурентов, их ассортименте представленной одежды, изучаете их функциональность.

2. **Проектирование** Вы выбираете компанию, которая разработает архитектуру и дизайн вашего интернет-магазина, согласовывает все условия, и компания приступает к разработке. Вам всё нравится, вы получаете хорошо проработанную архитектуру сайта и интересный завлекающий дизайн.

3. **Создание** Вы начинаете искать компанию-разработчика, которая всё это претворит в жизнь и разработает сам сайт будущего интернет-магазина. Вы заключаете договор с компанией-разработчиком и ждёте результат. Компания пишет код, отрисовывает дизайн, составляет необходимую документацию и, наконец-то, вы получаете готовый продукт!

4. **Поддержка** Вы размещаете свой сайт на сервере, запускаете интернет-магазин в работу и понимаете, что это ещё далеко не всё. Появляются новые способы оплаты, новые карточки товаров, новые условия для фильтров, какие-то баги, о которых вам пишут пользователи. И всё это значит лишь одно: сайт с интернет-магазином необходимо поддерживать. Вы берёте к себе в штат программистов или заключаете договор со специальной компанией на техническую поддержку своего IT-продукта.

⚠️ **Внимание!**

Чтобы у вас всё получилось и был разработан интернет-магазин, нужно выбрать модель и методологию разработки и управления этим проектом по созданию IT-продукта.

- **Модель** разработки IT-продукта описывает, какие стадии жизненного цикла продукт проходит и что происходит на каждой из них.

- **Методология** включает в себя набор методов по управлению разработкой: это правила, техники и принципы, которые делают её более эффективной и приводят к получению качественного продукта в результате.

Эволюция возникновения моделей разработки продукта

Модели разработки продукта берут своё начало ещё с 1930-х годов и связаны с появлением новых методов подготовки, анализа и исполнения крупных проектов в США: в фирмах «US Air Corporation» (авиационные проекты) и «Exxon» (нефтегазовые). В это же время в СССР начинает появляться теория и практика потоковой организации работ на масштабных строительных проектах. В конце 60-х – начале 70-х годов появляются первые предпосылки внедрения принципов управления проектами в процессы разработки ПО.

Происходит это по следующим причинам:

1. Первый кризис программного обеспечения.
2. Изменение взглядов на разработку софта.

Первый кризис программного обеспечения

На Конференции НАТО «Инженерия программного обеспечения» в 1968 году Фридрих Л. Бауэр констатировал увеличение темпов роста вычислительных мощностей компьютеров и значительное увеличения сложности проблем разработки и проектирования ПО.

Были выделены следующие проблемы:

- Стоимость разработки программ приблизилась к стоимости аппаратного обеспечения (серверов).
- Стоимость проектов всё чаще превышала бюджет.
- Программное обеспечение имело чрезвычайно низкое качество.
- Разработанные программы не соответствовали заявленным требованиям.
- Возникали трудности с поддержкой кода.

Изменение взглядов на разработку софта

Под влиянием перечисленных выше факторов промышленное программирование и программная инженерия выделились в отдельные области научного знания. Начались многочисленные исследования, как повысить качество и скорость разработки, отказоустойчивость кода, эффективность процессов тестирования и прочих технологических показателей.

Таким образом, появляются **стандарты и регламенты, методы и «лучшие практики»,** которые входят в программную инженерию как область знания.

Совокупности практик, применяемых на разных стадиях жизненного цикла программного обеспечения и объединенных общим философским подходом, принято называть **«Методологиями разработки программного обеспечения»**.

Классификация моделей разработки

Есть следующие модели управления разработкой продукта:

1. Waterfall Model (каскадная модель или «Водопад»).
2. V-образная модель (разработка через тестирование).
3. Спиральная модель.
4. Инкрементная модель.
5. Итеративная или итерационная модель.
6. Гибкие модели, методологии и подходы (Agile, Scrum, Kanban и другие).

Давайте рассмотрим каждую, с её плюсами и минусами, и посмотрим, как люди пришли к гибким методологиям.

Waterfall Model — каскадная модель или «Водопад»

Модель впервые описана в 1970 году в статье американского учёного Уинстона Ройса (одного из пионеров разработки ПО) и является традиционной и старейшей моделью управления разработкой продукта. Предполагает последовательное выполнение всех фаз проекта. Итоговый продукт получают после завершения всех фаз проекта.

Модель «Водопад» долго рассматривалась как основной способ регулярной разработки ПО. В 70-х–80-х годах XX века она была принята Министерством обороны США как стандарт.

Казалось бы, всё прекрасно, но почему каскадная модель не «панацея»?

Графически «Водопад» можно изобразить следующим образом:

Waterfall предполагает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом разработки ПО. Благодаря её жесткости, разработка проходит быстро, стоимость и срок заранее определены. *Но* это палка о двух концах по следующим причинам:

1. Каскадная модель даёт отличный результат только в проектах с чётко и заранее определенными требованиями и способами их реализации.
2. Нет возможности сделать шаг назад: тестирование начинается только после того, как разработка завершена или почти завершена.
3. Продукты, разработанные по данной модели без обоснованного её выбора, могут иметь недочёты (список требований нельзя скорректировать в любой момент), которые «выплывают» лишь в конце работы — из-за строгой последовательности действий.
4. Стоимость внесения изменений высока: для её инициализации приходится ждать завершения всего проекта.

Когда использовать Waterfall

Только при соблюдении следующих условий:

- Требования известны, понятны и зафиксированы.
- Нет противоречивых требований к функциональности продукта.
- Нет проблем с доступностью программистов нужной квалификации.

- В относительно небольших проектах.

> ⚠️ **Внимание!**
>
> Waterfall обладает как преимуществами, так и недостатками.

Преимущества:

- Разработку просто контролировать. Заказчик всегда знает, чем сейчас заняты программисты, и 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:

- Тестирование начинается на последних этапах разработки. Если в требованиях к продукту была допущена ошибка, её исправление дорого обойдётся. 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 работу. Чем обширнее документация проекта, тем больше изменений нужно вносить и дольше их согласовывать.

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.

Waterfall 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.

В 70-80 годы IT-продукты были достаточно простыми и в большинстве создавались для оборонной промышленности или масштабных проектов. В таких проектах не допускались существенные изменения по ходу работы над ними, поэтому водопадная модель разработки отлично выполняла свои задачи. Кроме того, водопадная модель была привычна всем: использовалась во многих проектах и казалась естественной моделью для успешной разработки ПО.

По мере развития IT-сферы и усложнения процессов разработки ПО водопадная модель продемонстрировала ряд своих существенных недостатков:

1. Сильный рост стоимости незапланированного изменения требований к продукту на каждом следующем этапе работ.
2. Количество проблем в процессе проектирования растёт вместе с проектом.

Тем не менее традиционная модель внесла существенный вклад в понимание процессов разработки ПО благодаря следующим утверждениям:

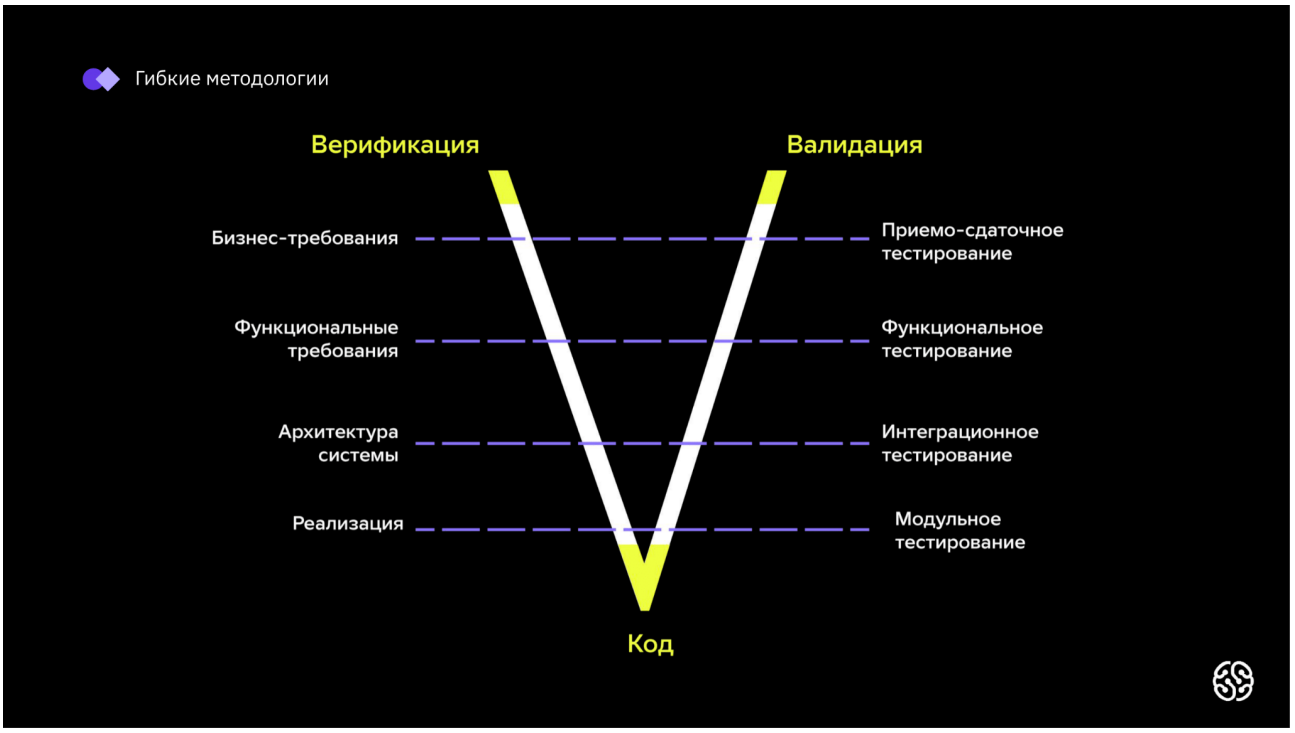
1. Старт реализации проекта должен быть отложен до выяснения всех целей и задач проекта и сбора полных требований к нему.
2. Разработка продукта должна быть хорошо скоординированной, подчиняться разумному планированию и управлению.

Прогресс не стоит на месте. Технологии развивались, рынок менялся, и эта модель уже не могла отрабатывать все сценарии, а недостатки стали перевешивать плюсы. В результате появилась V-образная модель.

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.

Графически её можно представить следующим образом:



V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и тестирование продукта, находящегося на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.

Преимущество V-образной модели в том, что количество ошибок в архитектуре ПО сводится к минимуму. А недостаток, как и у модели Waterfall, — в дороговизне исправления ошибок, допущенных при разработке архитектуры.

Когда использовать V-модель

- Если требуется тщательное тестирование продукта. V-модель оправдает заложенную в себя идею *validation and verification*.
- Для малых и средних проектов, где требования чётко определены и фиксированы.
- В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

Мы видим, что это лишь ответвление Waterfall-модели, созданное в ответ на требование рынка тщательно тестировать ПО во избежание ошибок в коде. Полученная модель управления разработкой ПО всё ещё не идеальна, а значит, нужны и другие подходы.

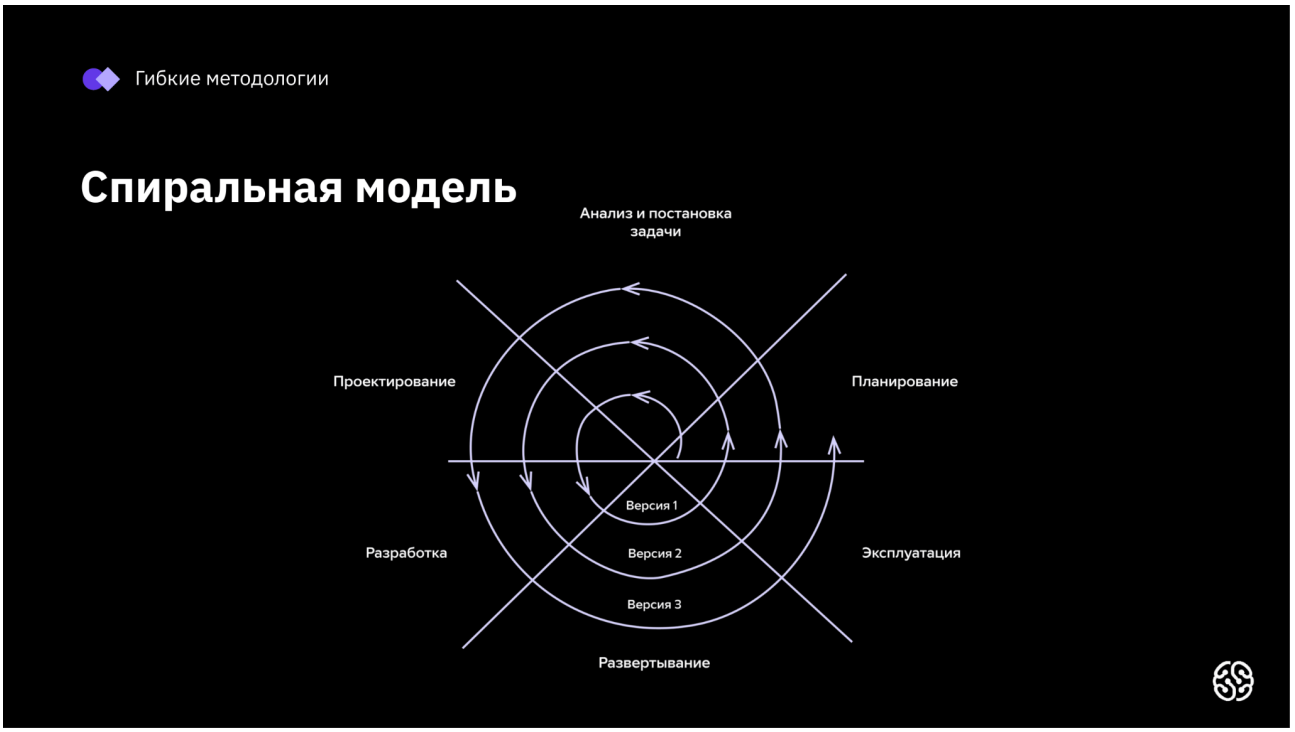
Спиральная модель

В спиральной модели работа над проектом представлена как цикл (спираль), на каждом витке которого — водопадная модель. Цикл начинается со сбора требований к предполагаемым изменениям, вносимым на данном витке, и завершается реализацией прототипа. Это решает основную проблему традиционных моделей: невозможность изменения требований к продукту.

Модель была впервые описана американским инженером-программистом Барри Боемом в статье “A Spiral Model of Software Development and Enhancement”, опубликованной в 1988 году.

С помощью этой модели заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Каждая следующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект.

Графически модель можно представить следующим образом:



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

1. Заказчик хочет создать глобальную систему, но начинает её с маленькой части и решает сначала сделать управление чайником с телефона. Программисты приступают к работе по модели «Водопад»: выслушивают идею, анализируют предложения на рынке, обсуждают с заказчиком архитектуру системы, решают, как

будут её реализовывать, разрабатывают, тестируют и «выкатывают» конечный продукт.

2. Заказчик оценивает результат и риски: насколько нужна пользователям следующая версия продукта — уже с возможностью управления телевизором. Таким образом, наращивая функциональность, заказчик всё больше приближается к своей глобальной идее о создании умного дома. Заказчик рассчитывает сроки, бюджет, решает дополнить свой продукт новой технологией и заказывает разработку. Программисты действуют по каскадной модели и представляют заказчику более сложный продукт, разработанный на базе первого.

3. Заказчик думает, что пора создать функциональность для управления холодильником с телефона. Но, анализируя риски, понимает, что в холодильник сложно встроить модуль Wi-Fi, да и производители не заинтересованы в сотрудничестве по этому вопросу. Следовательно, риски превышают потенциальную выгоду. На основе полученных данных заказчик решает прекратить разработку и совершенствовать имеющуюся функциональность, чтобы со временем понять, как развивать систему «Умный дом».

⚠️ **Внимание!**

Преимуществом спиральной модели является то, что большое внимание в ней уделено проработке рисков.

При этом есть и недостатки:

- Риск застрять на начальном этапе, бесконечно совершенствовать первую версию продукта и не продвинуться к следующим.

Давайте обратимся к предыдущему примеру с системой «Умный дом»: мы сидим и год допиливаем приложение по управлению чайником, дотачиваем его, меняем визуал, расширяем количество поддерживаемых чайников, добавляем отложенный старт и т.д., но не переходим к следующим этапам разработки по умному дому.

- Разработка длится долго и стоит дорого.

На этой модели прогресс тоже не остановился.

Инкрементная модель

В инкрементной модели полные требования к системе делятся на разные сборки.

В этой модели несколько циклов разработки составляют жизненный цикл «Мультиводопад». Цикл разделён на мелкие легкосоздаваемые модули. Каждый модуль проходит фазы определения требований, проектирования, кодирования, внедрения и

тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не завершился создание полной системы.

⚠️ **Внимание!**

Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы.

Рассмотрим эту модель 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.

И ещё один пример – инкрементная модель при создании системы «Умный дом»:

1. Заказчик решил, что хочет создать систему «Умный дом», и написал подробное техническое задание с развёрнутой функциональностью по управлению всеми домашними приборами с телефона. Программисты предложили реализовать основные функции: создать приложение с минимальной функциональностью, к примеру, по включению чайника, а затем протестировать на пользователях, «взлетит» или нет.

2. Команда разработки показывает продукт заказчику и выпускает его на рынок. Если и заказчику, и пользователям новая система нравится, работа над ней продолжается, но уже по частям.

3. Программисты параллельно создают функциональность для управления телевизором, холодильником, кондиционером и другими бытовыми приборами, согласовывая всё это с заказчиком и постепенно выпуская обновлённые версии продукта на рынок, тем самым тестируя это на пользователях. Инкремент за инкрементом они совершенствуют продукт, приближаясь к описанной в техническом задании полноценной системе «Умный дом».

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.

Инкрементная модель очень похожа на спиральную, но и здесь не все проблемы решены – нужно думать дальше.

Итеративная или итерационная модель

Модель предполагает разработку ПО как последовательность итераций, каждая из которых сама по себе является небольшим проектом в рамках общей задачи и предполагает измеримый прирост ценности продукта по завершении итерации.

Можно сказать, что эта модель даёт начало гибким методологиям!

Итерационные (гибкие) методологии, в отличие от спиральной**,** фокус смещают внимание с обеспечения полноты требований к продукту на формирование процессов слаженной работы команды.

Зачатки модели итеративной разработки появляются ещё в 30-х годах в статьях специалиста по проблемам качества продукции Уолтера Шеварта из компании Bell Labs. Показательным примером эффективности гибких методологий стал реализованный в 50-е годы проект сверхзвукового самолета X-15. Согласно мнению участников проекта, итерационный подход стал одним из ключевых факторов успешной реализации проекта.

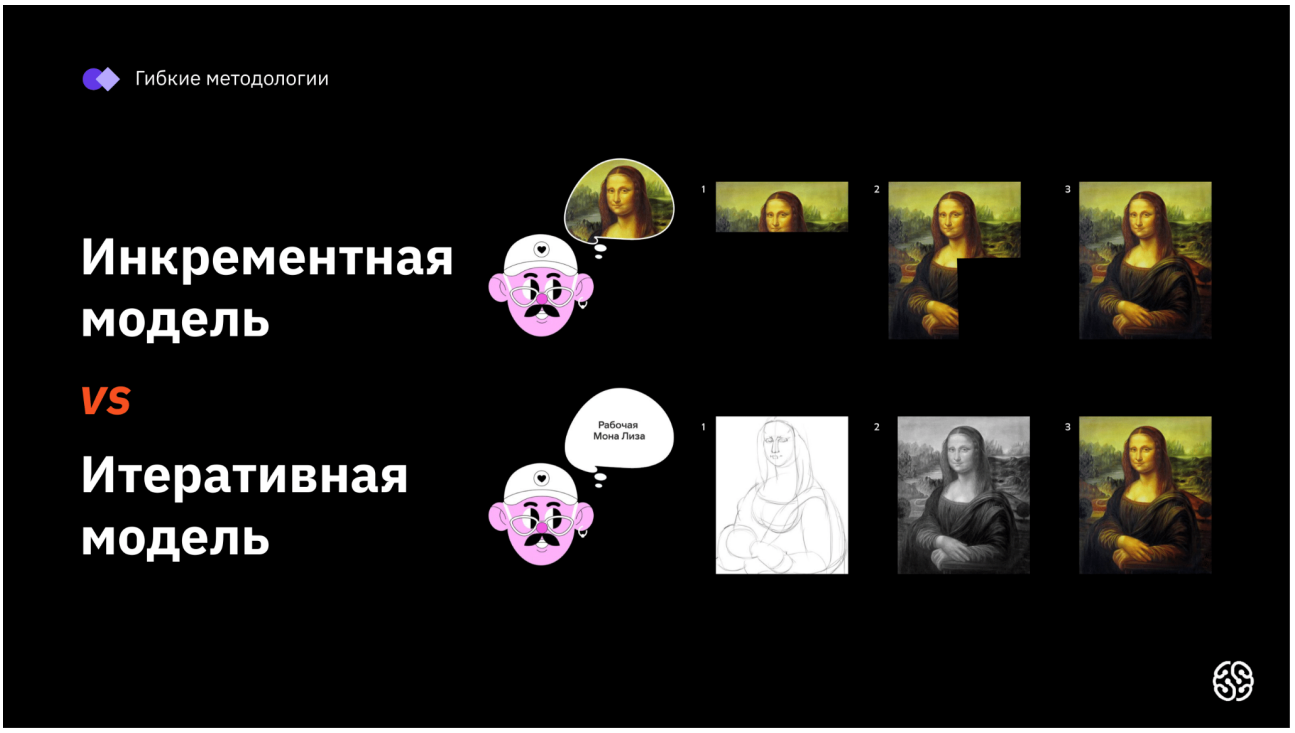
⚠️ **Внимание!**

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части возможностей, которые служат базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.

Именно это и выделяет итеративную модель среди всех остальных:

- Она гибкая.
- Не нужно в самом начале чётко просчитывать все риски, технические требования и другие показатели IT-продукта. Зачастую это и невозможно сделать, когда вы создаёте что-то уникальное.
- Можно разделить IT-продукт на несколько итераций, реализовывать их и затем тестировать: это помогает создавать работоспособную версию продукта на каждой итерации. Ошибки можно выявлять на ранних этапах и сразу их исправлять.

Давайте посмотрим на классический сравнительный пример создания Мона Лизы с помощью итеративной (гибкой) и инкрементной (жёсткой) моделей разработки:



На иллюстрации показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй появляются цвета, а третья добавляет деталей, насыщенности и завершает процесс.

В инкрементной же модели функциональность продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, здесь каждый кусочек представляет собой целостный элемент.

Примером итерационной разработки может служить распознавание голоса. Первые исследования и подготовка научного аппарата начались давно, сначала в мыслях, затем — на бумаге. С каждой новой итерацией качество распознавания улучшалось. Тем не менее, идеальное распознавание ещё не достигнуто, следовательно, задача ещё не решена полностью.

Когда оптимально использовать итеративную модель

- Проект большой или очень большой.
- Основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

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 позволяет оперативно получать обратную связь от заказчика и пользователей. А значит, фокусироваться на самых 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.

⚠️ **Внимание!**

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

Итак, мы плавно подошли к гибким методологиям. С этого момента начинается их эволюция.

Методологии разработки ПО как область знаний в программной инженерии развивались эволюционно наряду с развитием самой отрасли промышленного программирования.

⚠️ **Внимание!**

Замена традиционных подходов итерационными была неизбежна ввиду быстрорастущей сложности технологий и роста количества практических проблем в процессах разработки продуктов.

Так, в 1995 году появляется методология Scrum, в 1996 — экстремальное программирование (XP). Разработанный в 2001м году Agile-манифест провозгласил эру итерационной разработки и задал направление развития гибких методологий в будущем.

Agile

Давайте поговорим про наиболее распространённые гибкие модели и методологии, которые чаще всего применяются в IT и будут рассмотрены в нашем курсе.

На основе итеративной модели был создан Agile. Это не модель и не методология, а скорее подход к разработке.

Что же за зверь такой — Agile?

Agile («эджайл») переводится с английского как «гибкий». Подход включает в себя следующие практики и методологии для эффективной работы над продуктом:

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

Из всех перечисленных выше гибких подходов наиболее часто в IT применяют Scrum и Kanban.

Scrum

- Scrum обычно называют не методологией, а фреймворком.
    - Фреймворк — это более сформированная методология со строгими правилами.
- В Scrum все роли и процессы чётко прописаны.

Kanban

Сегодня Kanban является одной из наиболее популярных методологий разработки ПО. Если коротко, принцип Kanban состоит в следующем:

- 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 Scrum, при использовании Kanban-подхода можно взять срочные задачи в разработку сразу, не дожидаясь начала следующей итерации.

Отличия гибких методологий от традиционных (жёстких)

Давайте посмотрим на основные различия между гибким и традиционным подходом к разработке:

Характеристики проекта	Традиционный подход (модели и методологии)	Гибкие методологии
Подход	Прогнозирующий	Адаптивный
Критерии успеха	Следование плану	Ценность для бизнеса
Риски	Риски определены	Риски не определены
Контроль	Легко контролировать	Зависит от профессионального уровня специалистов
Заказчики	Низкая вовлечённость	Высокая вовлечённость
Документация	Детальная с самого начала проекта	Доработка по мере развития проекта
Требования	Известны заранее, стабильны	Не всегда известны заранее, легко изменяемы
Команда проекта	Включение новых специалистов на любом этапе	Опытные специалисты, стабильный состав
Рефакторинг (изменение кода)	Дорого	Недорого

Выводы

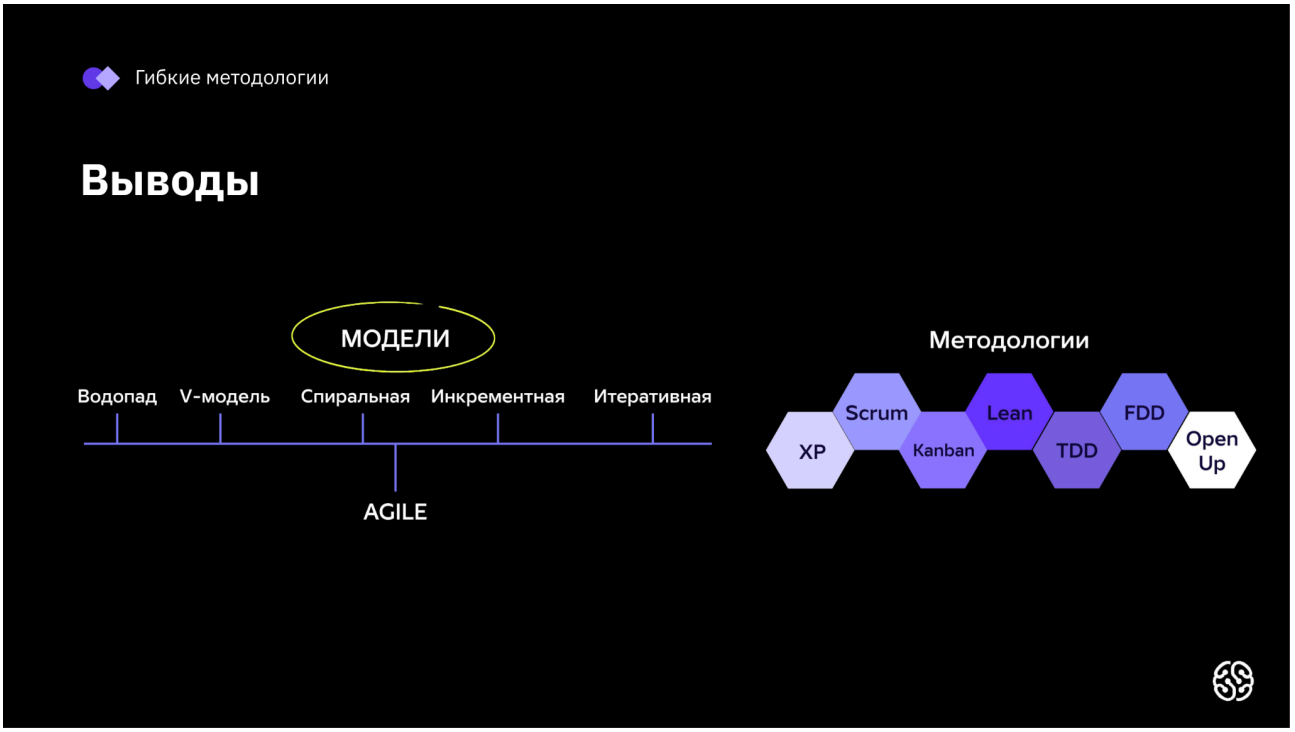
В современной практике модели разработки IT-продуктов многовариантны. Ни одна из низ не является единственно верной для всех проектов, стартовых условий и моделей оплаты.

Даже гибкие методологии не дают универсального решения всех проблем: не все заказчики готовы максимально вовлекаться в разработку и не всегда возможно гибкое финансирование проекта.

⚠️ Внимание!

Опытный руководитель проекта, разработки или другой работы, связанной с IT-продуктом, должен умело найти ту самую единственную верную модель и методологию или совместить несколько при работе над продуктом. Конечно, нет универсального шаблона правил, следуя которым вы точно попадёте в точку. Нет, всё это вы научитесь делать только на практике, а мы с вами разберём разные ситуации и увидим, где и при каких обстоятельствах максимально применима та или иная гибкая методология разработки.

Также хотелось бы графически продемонстрировать традиционные и гибкие методологии разработки:



Как видно из схемы выше, сначала люди придерживались традиционных (жёстких) подходов с детальной проработкой всех рисков и технических требований на старте. Но, по мере развития информационных технологий и открытия новых, порой недоступных человеческому уму, возможностей IT-продуктов? пришло понимание, что нельзя всё просчитать на старте и чётко следовать плану. В этот момент люди задумались, не лучше ли разрабатывать уникальные большие продукты постепенно, итерациями, смотреть, тестировать их, исправлять ошибки и делать что-то качественное и интересное. Ведь ключевая цель при создании любого IT-продукта – обеспечить удобство его использования.

Иначе жёсткое следование плану приведёт лишь к соблюдению условий плана, а не удовлетворению конечных пользователей и успеху создаваемого IT-продукта.

Используемые источники

1. Project Management Body of Knowledge (PMBOK).
2. Эндрю Стеллман, Дженнифер Грин «Постигая Agile. Ценности, принципы, методологии».
3. Стивен Деннинг «Эпоха Agile».
4. https://blog.ganttpro.com/ru/metodologiya-agile-methodology/
5. https://rb.ru/opinion/agile-practices-russia/
6. https://infostart.ru/1c/articles/1056335/