**SDLC**- Жизненный цикл разработки программного обеспечения (Software Development Life Cycle).

**STLC**. Жизненный цикл тестирования (Software Testing Life Cycle).

Методологии - это по сути названия разных способов разрабатывать по в зависимости какие у нас цели, ограничения, риски, сроки …

Методологии условно можно разделить на:

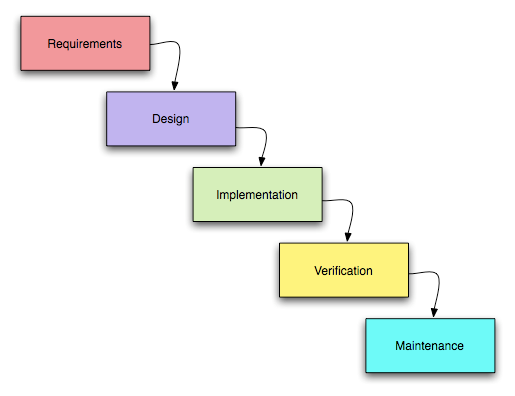
-Последовательные-водопад, v-модель

-Цикличные - итеративная, инкрементальная, гибкие методологии (agile)

Модель - в целом описывает как мы разрабатываем п.о.

Модель разработки ПО описывает какие стадии разработки жизненного цикла проходит ПО и, что происходит на каждой из них.

Методологии это конкретные практики

**1)- Waterfall** - **(Каскадная модель) Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к ПО. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка — внесение новой функциональности и устранение ошибок.**

**Минусы:**

**-До начала работы над проектом нужно утвердить все требования в случае необходимости внесение изменений в требования познее, мы не можем вернутся к первой стадии и нам придётся переделать заново проделанную до этого работ, из-за этого подвинутся также сроки, стоимость конечного продукта**

**-тесты в каскадной модели начинаются только после имплементации софта.**

**-Очень негибкая модель**

**-У Заказчика нет возможности ознакомиться с системой заранее**

**-У Пользователя нет возможности привыкать к продукту постепенно**

**-Все требования должны быть известны в начале жизненного цикла проекта**

**-Отсутствует возможность учесть переделку, весь проект делается за один раз**

**Плюсы:**

**Полное документирование каждого этапа (нам всегда доступны чёткие требования в виду этого мы всегда можем чётко спланировать наши сроки и затраты)**

**Высокая прозрачность разработки и фаз проекта**

**Чёткая последовательность**

**Стабильность требований**

**Строгий контроль менеджмента проекта**

**Облегчает работу по составлению плана проекта и сбора команды проекта**

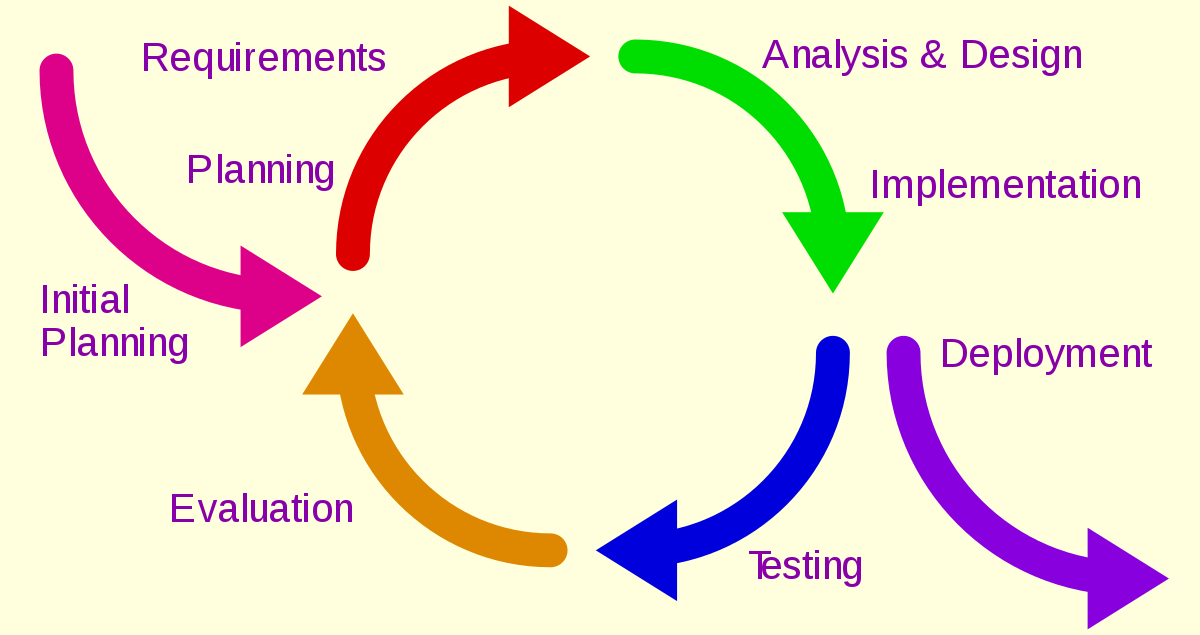
**Хорошо определяет процедуру контроля качества**

**Область применения:**

**-Проекты с чётко обозначенными требованиями для которых не предусматриваются их изменения в процессе разработки (гос. учреждения, банковские структуры).**

**-Проекты которые мигрируют с одной платформы на другую, при этом сохраняются все требования без изменений, меняется только некое системное окружение**

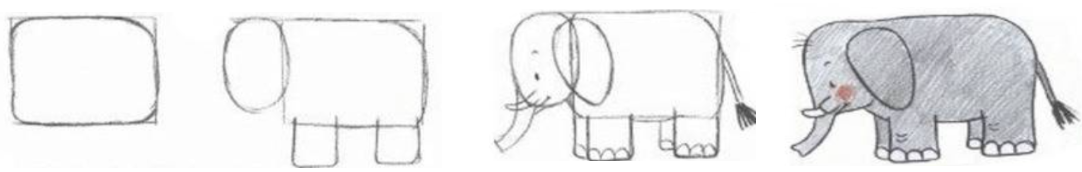
**Так как теперь эпоха изменяющихся и динамично развивающихся требований в процессе самой разработки ПО то данная модель малоиспользуемая.**

**2)- Iterative Model (итеративная модель)** вытекает из вотерфола

**Жизненный цикл разработки проекта при итерационной разработке разбит на последовательность итераций, каждая из которых, по сути, является проектом в миниатюре, то есть включает в себя все процессы разработки ПО (сбор и анализ требований, составление спецификаций, непосредственную реализацию, тестирование и запуск), но в рамках одной итерации разрабатывается не весь проект, а только его версия или отдельная часть.**

**Как правило, цель каждой итерации — это получение версии ПО, включающей в себя как новые или переработанные возможности, реализованные в ходе текущей итерации, так и функциональность всех предыдущих итераций. Результат же финальной итерации содержит всю требуемую функциональность продукта.**

**Итеративность (iteration, «повторение») в данном случае означает подход, основанный на выполнении задач в рамках «мини-проектов», инкрементность (increment «увеличение») означает последовательное добавление функционала к разрабатываемому продукту, а эволюционность (evolution, «развертывание») — процесс развития продукта, напоминающий естественное развитие биологических видов.**

**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:

-Заказчик решил, что хочет создать мессенджер.

-Разработчики сделали приложение, в котором можно добавить друга и запустить чат на двоих.

-Мессенджер «выкатили» в магазин приложений, пользователи начали его скачивать и активно использовать. Version 1.0

- Заказчик понял, что продукт пользуется популярностью, и решил его доработать.

-Программисты добавили в мессенджер возможность просмотра видео (version 1.1), загрузки фотографий (version 1.2), записи аудиосообщений (version 1.3). Они постепенно улучшают функциональность приложения, адаптируют его к требованиям рынка. Version 2.0

**Плюсы:**

**- 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.**

**-Проще протестировать продукт полученный в результате одной итерации, так как сама итерация не большая по сравнению если бы мы пилили проект полностью.**

**- Постоянное тестирование пользователями позволяет быстро обнаруживать и устранять ошибки.**

**Минусы:**

**- В итерации только одна фаза тестирования. Т.е. мы не тестируем требования, не тестируем дизайн, не тестируем код, но мы тестируем готовый результат всей итерации. Да это не большой проект, мы разрабатываем только версию но в случае нахождения багов надо будет откатываться на фазу требований или дизайна или кода, а то и всё по порядку. Т.е. такойже косяк как и в вотерфолле, за тем исключением что продукт разработан не весь а только какая то его часть. Похорошему надо тестировать каждую фазу в итерации.**

**- 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.**

Применяется:

-В крупных проектах, когда нам известны не все требования но какие-то ключевые аспекты - В проектах в которых нам известно, что требования будут менятся.

**3) Инкрементная модель (Incremental model)** похожа на итеративную модель (можно сказать её версия)

Заказчик точно знает что он хочет и какие требования должны быть соблюдены.

Это когда проект мы разбиваем по частям, а затем наченаем по этим частям разрабатывать. Т.е. каждая часть проходит одинаковый цикл состоящий из этапов:

планирование,

составление требований,

проектирование,

исполнение,

тестирование,

оценка и затем развёртывание выполненной версии.

Пример: заказчик хочет соц сеть. Разработчики принимают решение создать соцсеть с минимальным функционалом, только имя профиля + возможность посылать друг-другу сообщения. Эта часть проходит все этапы описанные выше и на выходе мы получаем хоть и мало функциональный но готовый продукт. Затем пользователи пользуются, заказчик анализирует фид бэк. Затем заказчик решает выпустить новую версию V2 этой соц сети. Эта версия также проходит все этапы и на выходе получаем V2, с добавленной опцией например пересылать фото. Опять же пользователи используют, заказчик анализирует фид бэк. и т.д.



Используется когда есть чёткие, понятные требования к системе и продукт должен быстро выйти на рынок, заказчик отчётливо знает что он хочет.

**Плюсы**:

-Получаем рабочее приложение за короткий период времени. Первая версия.

-Сразу можно проанализировать пользуется приложение успехом или нет, стоит его дальше развивать или нет.

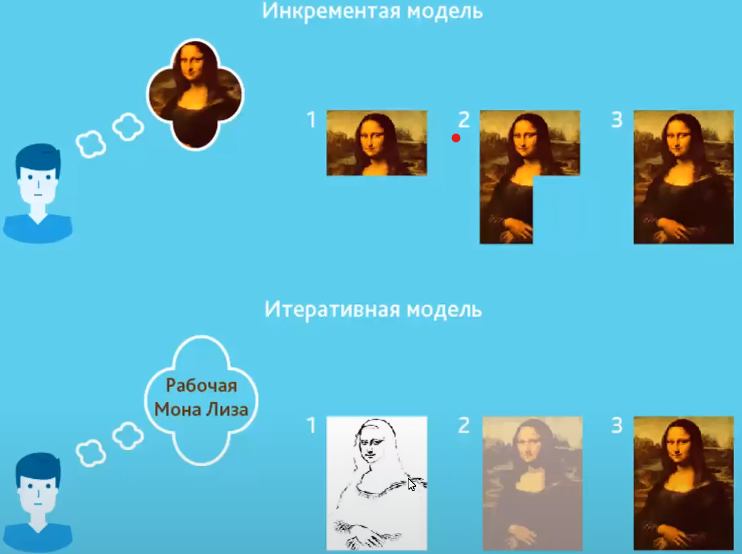
-Не нужно ждать и тем более тратить большое количество денег на проект который может не пойти.

-При разработке нового модуля можно учесть фидбэк от юзеров по действующей версии, тем самым улучшать систему именно под юзеров а не под требования.

-если возникнут какие то ошибки то это будет дешевле.

Минусы:

-Сложно разбить на части, не ко всем приложениям это применимо

**отличия между инкрементной и итеративной:**

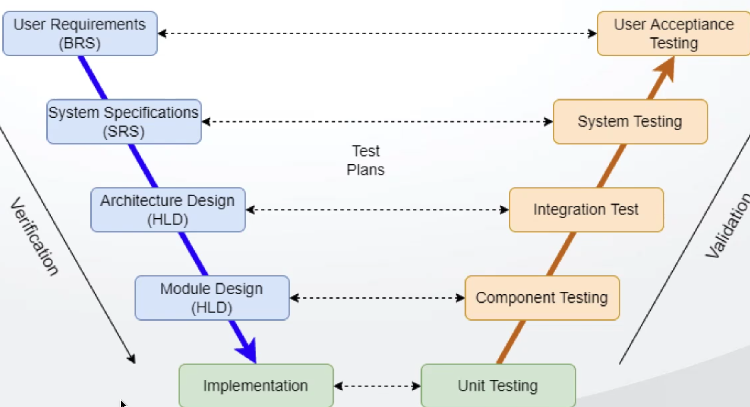
Разница в том что в инкрементной модели **заказчик знает** как выглядит конечный продукт но чтобы убедится что продукт будет востребован, его разбивают на части и разрабатывают сначала только базовый функционал, если всё ок то разрабатывают следующую часть.

В Итерационной модели **заказчик не знает** как будет выглядеть законченный продукт, но понимает какой то самый базовый функционал который должен присутствовать. С него и начинается разработка продукта. Если пользователи довольны они оставляют свои комментарии и пожелания отсюда рождается следующая версия продукта.

**4)-V-модель-(разработка через тестирование) Это модель, в которой фаза тестирования происходит параллельно с соответствующей фазой создания приложения.**

**It is based on the association of a testing phase for each corresponding development stage. The development of each step is directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e., for each development activity, there is a testing activity corresponding to it.**

**!!! Внимание** - здесь есть 2 большие раздела verification и Validation они идут **по очереди**, они не идут параллельно как может показаться с картинки. Сначала выполняются все фазы по синей стрелке и для каждой фазы выполняются тесты, затем выполняются все фазы по оранжевой стрелке, для каждой фазы свои тесты.

****

**Verification -** means whether we are building the right project or not. Focus on Static testing documentation.

**Validation -** means whether we are building the product right. Takes place **after** verification are completed. Focus on **dynamic** testing software.

**Это улучшенная версия классической каскадной модели (waterFall). Здесь на каждом этапе (фазе) происходит контроль текущего процесса, для того чтобы убедится в готовности (отсутствие дефектов) данной фазы и соответственно возможности перехода на следующую фазу.**

В этой модели, тестирование начинается сразу с фазы написания требований (user requirements), и для каждой последующей фазы предусмотрены **свои специфические** тесты.

**Процесс разработки представлен нисходящей последовательностью в левой части условной буквы V, а стадии тестирования – на ее правой части, соответствие этапов разработки и тестирования показано горизонтальными линиями.**

**в V-модели получается, что 2 жизненных цикла идут параллельно друг другу SDLC и STLC, когда со стороны разработчиков идет этап анализа требований, qa в это время уже пишут тест-план/тест-кейсы/чек-листы для будущего системного тестирования; аналогично на следующих этапах sdlc. На каждом этапе происходит контроль текущего процесса.**

**Плюсы:**

**-Планирование тестирования и верификация самой системы производится на ранних этапах**

**-Раннее тестирование**

**- тестирование каждой фазы**

**Минусы:**

**-Нету возможности внесения динамических изменений (Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде»)**

**- Больше документации.**

**Область применения:**

**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.***

**! Внимание** - Самое важное что это как водопад но на каждой стадии мы проводим тестирование. ВСЁ

**Гибкая модель**

Гибкая модель представляет собой совокупность различных подходов к разработке ПО и базируется на т.н. «agile-манифесте».

Люди и взаимодействие важнее процессов и инструментов.

Работающий продукт важнее исчерпывающей документации.

Сотрудничество с заказчиком важнее согласования условий контракта.

Готовность к изменениям важнее следования первоначальному плану.

Стоит понимать, что левая часть утверждения не отменяет наличие правой. Речь лишь про то, что является более приоритетным для данной модели. (примечание автора курса)

Положенные в основу гибкой модели подходы являются логическим развитием и продолжением всего того, что было за десятилетия создано и опробовано в водопадной, v-образной, итерационной инкрементальной, спиральной и иных моделях.

Главным недостатком гибкой модели считается сложность ее применения к крупным проектам, а также частое ошибочное внедрение ее подходов, вызванное недопониманием фундаментальных принципов модели. Тем не менее можно утверждать, что всё больше и больше проектов начинают использовать гибкую модель разработки.

Спиральная модель представляет собой частный случай итерационной инкрементальной модели, в котором особое внимание уделяется управлению рисками, в особенности влияющими на организацию процесса разработки проекта и контрольные точки

На стадии написания требований, (софт ещё не написан) тестирование фокусируется на 2-х аспектах: **Correctness, completeness**. Надо помнить что документация относительно статичный документ поэтому и тестирование этих документов будет называтся static testing. Дак а как проверить Correctness, completeness это делается вместе с командой и надо использовать static testing techniques :

**1) Review,** - это личное ознакомление с документацией на предмет корректно ли описана документация (соответствует ли она функционалу планируемого продукта). Достаточно ли полно описана эта функциональность.

**2) Walkthrough** - несколько людей задействованы. Один читает другие сверяют, задают вопросы, проясняют какието непонятные моменты. Идёт обсуждение.

**3) Inspection** это когда митинг происходит с модератором, кто то читает остальные анализируют, задают вопросы, уточняют и при необходимости вносятся правки в документацию

Actual software will be ready during the implementation phase. Once the software ready we will start testing software it is called - **dynamic testing**

Static testing - testing project related documentations.

Dynamic testing - testing software.

**Поочерёдность тестирования согласно V - модели.**

Dynamic testing starts with unit testing and it is done by developers. Then component testing, (component consists of units) it tests components separately and similar to unit testing it is done by developers. Then comes the integration testing (when several modules/components are ready they need to be checked on how they work/communicate together) it is done by developers.

After that the entire software will be combined as one single unit/final software project. On that final project we will do **system** testing and this system testing is done by **testers**, the other phases are done by developers. Then we will deliver software to the customer and the software will be tested in the customer environment - user acceptance testing (UAT).

Смотри папку фукциональное тестирование