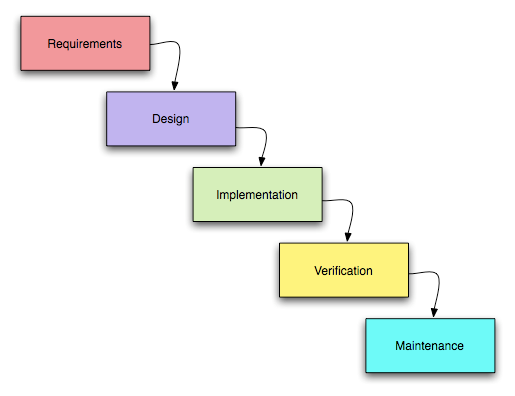
**SDLC**- Жизненный цикл разработки программного обеспечения (Software Development Life Cycle).

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

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

**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**

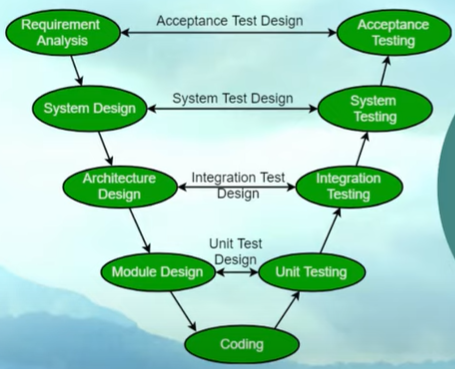
**Хорошо определяет процедуру контроля качества**

**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)-V-модель-(разработка через тестирование) Это модель тестирования, в которой фаза тестирования происходит параллельно с соответствующей фазой написания кода, это улучшенная версия классической каскадной модели. Здесь на каждом этапе происходит контроль текущего процесса, для того чтобы убедится в возможности перехода на следующий уровень. В этой модели тестирование начинается еще со стадии написания требований, причем для каждого последующего этапа предусмотрен свой уровень тестового покрытия.** 

**Процесс разработки представлен нисходящей последовательностью в левой части условной буквы V, а стадии тестирования – на ее правом ребре, 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:**

**-Строгие этапы**

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

**-Раннее тестирование**

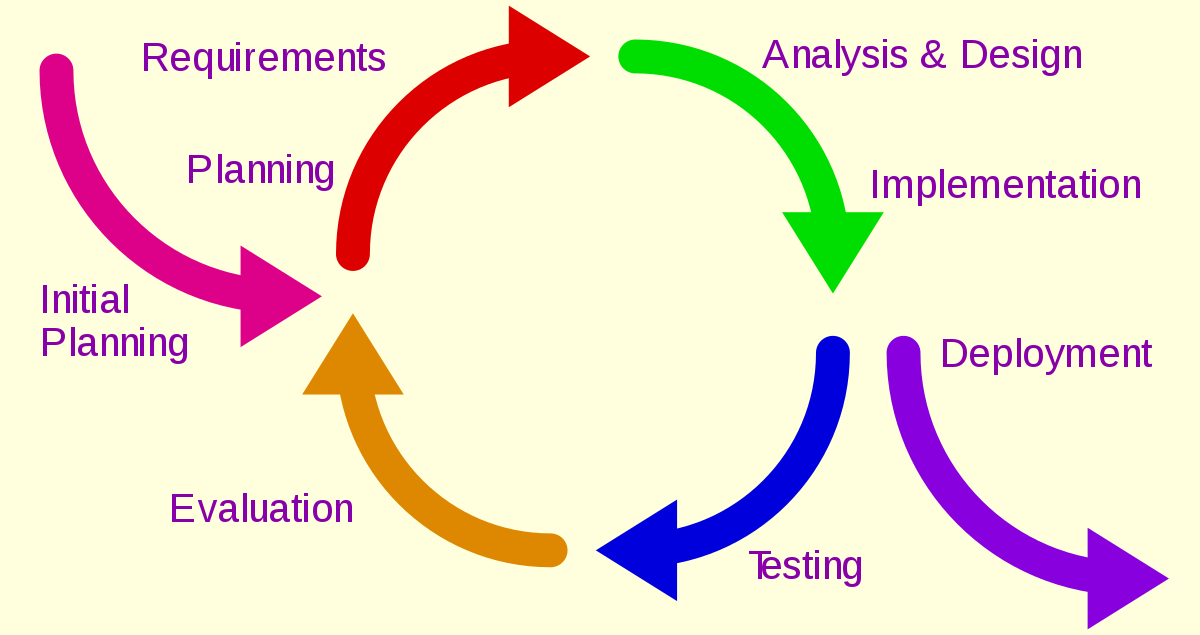
**-Промежуточное тестирование каждого этапа**

**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»)**

**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.***

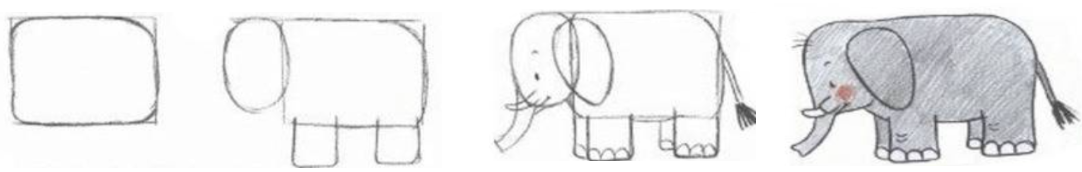
3)- **Iterative Model (итеративная модель)**

**Жизненный цикл 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.**

Итеративная, итерационная, инкрементная и эволюционная разработка — фактически, это синонимы.

**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:**

**- 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.