11 Классические модели процесса разработки программного обеспечения - линейная, прототипирование, компонентная, эволюционная, инкрементальная, спиральная

линейная

1. планирование, сбор требований
2. анализ требований
3. проектирование
4. реализация, тестирование
5. внедрение, сопровождение

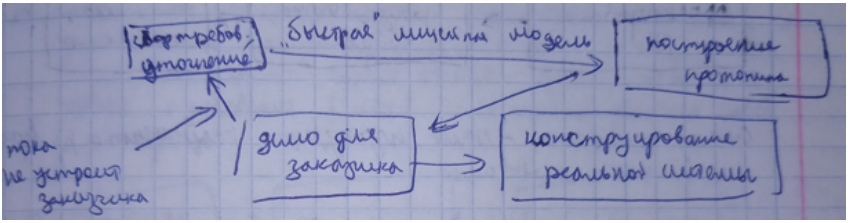
- базовые виды деятельности отделены друг от друга
- каждая стадия утверждается документально, результаты передаются на следующую стадию
- заказчик требуется только в начальных тапах и в конце
- не гибкая модель для заказчика и разработчика
- применяется лишь тогда, когда требования можно формализовать достаточно четко, полно и корректно
- удобно планировать сроки завершения работ

каскадная модель с возвратом (итеративная каскадная модель) позволяет возвращаться к предыдущему этапу для уточнения или пересмотра ранее принятых решений. имеет большую гибкость, время жизни каждого этапа растягивается на весь период разработки.

прототипирование

поэтапное уточнение требований заказчика - существенная часть спецификации требует составления прототипа. используется:

- все требования заказчик не может сформулировать → эволюционное прототипирование
- заранее не известно, можно ли создать такое ПО → экспериментальное прототипирование



плюсы:

- заказчик быстро получает представление о программе
- активная обратная связь с заказчиком
  минусы:
- заказчик и разработчик могут принять прототип за реальный продукт

компонентная

повторное использование программных модулей (компонент), наличие базы существующих компонент.

1. спецификация требований, требования заказчика
2. анализ компонент
3. моделирование требований
4. проектирование ПО
5. разработка, сборка

6. аттестация системы

анализ компонент - поиск подходящих компонент и проверка системных требований.
модификация требований - попытка по максимуму использовать возможности отобранных компонент, предложение альтернативных решений
проектирование - определение структуры ПО в соответствии с возможностями программных компонент, для недоступных компонент - проектирование нового ПО

плюсы:

- сокращение времени разработки
- уменьшение числа создаваемых программ
- снижение стоимости разработки
- увеличение надежности
  минусы:
- недостаточность компонент для нового ПО
- соответствие требованиям заказчика
- невозможность влияния на появление новых версий компонент

эволюционная

или итеративная.
на исходном этапе жизненного цикла ПО представлен весьма смутно

- требования известны лишь в общих чертах
- нет явного заказчика
- не определены ресурсы проекта

ситуация сходная с прототипированием. отличие: в конце любой итерации имеется законченный проект, а не

прототип. полученная версия эксплуатируется, в процессе выявляются новые требования, в соответствии с которыми делается новая итерация.

1. план
2. разработка
3. выпуск версии
4. эксплуатация
5. → 1.

плюсы:

- промежуточный продукт уже можно использовать
- привлечение персонала и средств по необходимости
- можно разрабатывать ПО без заказчика
  минусы:
- ПО может быть плохо структурированным - внутреннее устройство ПО с каждой итерацией становится все более громоздким и неэффективным
- отсутствие четкого понимания сроков и стоимости
- часто этапы плохо документируются

инкрементальная

пошаговая разработка. условия: когда требования можно разобрать на ядро и дополнительные функции. нужно знать количество итераций.

1. глобальное планирование всех итераций
2. пошаговая детализация требований
3. разработка архитектуры всей системы
4. переход к итерациям:
    1. планирование и разработка
    2. аттестация
    3. сборка

    4. аттестация собранной версии системы
5. конечное ПО

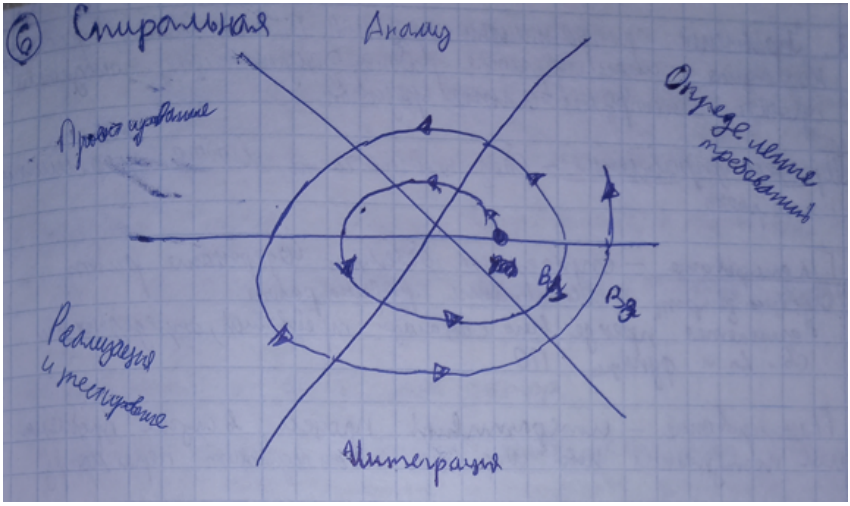
плюсы:

- каждая итерация продукта - готовая программа, которую можно использовать
- есть четкий план развития
- более важные функции проходят больше проверок

минусы:

- подходит только для подобного случая (есть ядро и доп. сервисы, которые можно разрабатывать постепенно)
- разработка конечного продукта будет продолжаться долгое время

спиральная



1. определение требований
2. анализ
3. проектирование
4. реализация и тестирование
5. интеграция

6. → 1.

подходит для новых долгосрочных проектов или для проектов с ожидаемым существенным изменением требований.

каждый виток спирали соответствует поэтапной модели создания фрагмента или версии ПО.
неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работ на текущем.
недостающую задачу можно выполнить на следующей итерации.
задача - как можно быстрее показать пользователю работоспособный продукт для уточнения и дополнения требований.

плюсы:

- ориентация на развитие и модификацию ПО в процессе его проектирования
  минусы:
- основная проблема - определение момента перехода на следующий этап. необходимо ввести временные ограничения на каждый этап жизненного цикла.
  переход осуществляется в соответствии с планом, даже если не вся работа закончена.