**Жизненный цикл программного обеспечения** — ряд событий, происходящих с системой в процессе ее создания и дальнейшего использования. Говоря другими словами, это время от начального момента создания какого-либо программного продукта, до конца его разработки и внедрения. Жизненный цикл программного обеспечения можно представить в виде моделей.

**Модель жизненного цикла программного обеспечения** — структура, содержащая процессы действия и задачи, которые осуществляются в ходе разработки, использования и сопровождения программного продукта.  
Эти модели можно разделить на 3 основных группы:

1. Инженерный подход
2. С учетом специфики задачи
3. Современные технологии быстрой разработки

Теперь рассмотрим непосредственно существующие модели (подклассы) и оценим их преимущества и недостатки.

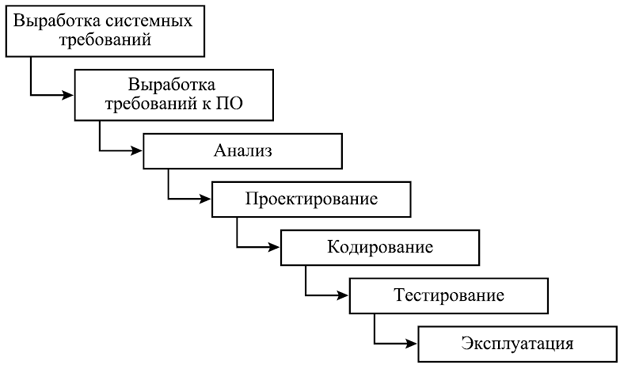
**Модель кодирования и устранения ошибок**

Совершенно простая модель, характерная для студентов ВУЗов. Именно по этой модели большинство студентов разрабатывают, ну скажем лабораторные работы.  
Данная модель имеет следующий алгоритм:

1. Постановка задачи
2. Выполнение
3. Проверка результата
4. При необходимости переход к первому пункту

Модель также *ужасно* устаревшая. Характерна для 1960-1970 гг., по-этому преимуществ перед следующими моделями в нашем обзоре практически не имеет, а недостатки на лицо. Относится к первой группе моделей.

**Каскадная модель жизненного цикла программного обеспечения (водопад)**

Алгоритм данного метода, который я привожу на схеме, имеет ряд преимуществ перед алгоритмом предыдущей модели, но также имеет и ряд **весомых** недостатков.  
  
  
Преимущества:

* Последовательное выполнение этапов проекта в строгом фиксированном порядке
* Позволяет оценивать качество продукта на каждом этапе

Недостатки:

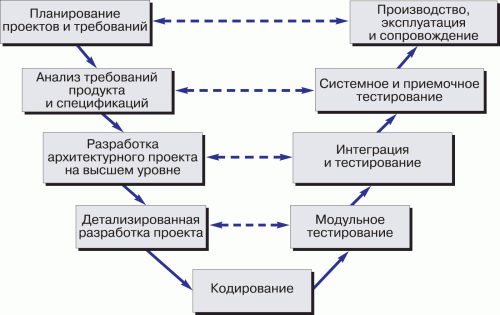
* Отсутствие обратных связей между этапами
* Не соответствует реальным условиям разработки программного продукта

Относится к первой группе моделей.

**Каскадная модель с промежуточным контролем (водоворот)**

Данная модель является почти эквивалентной по алгоритму предыдущей модели, однако при этом имеет обратные связи с каждым этапом жизненного цикла, при этом порождает очень весомый недостаток: *10-ти кратное увеличение затрат на разработку*. Относится к первой группе моделей.

**V модель (разработка через тестирование)**

Данная модель имеет более приближенный к современным методам алгоритм, однако все еще имеет ряд недостатков. Является одной из основных практик экстремального программирования.  
  


**Модель на основе разработки прототипа**

Данная модель основывается на разработки прототипов и прототипирования продукта.  
**Прототипирование** используется на ранних стадиях жизненного цикла программного обеспечения:

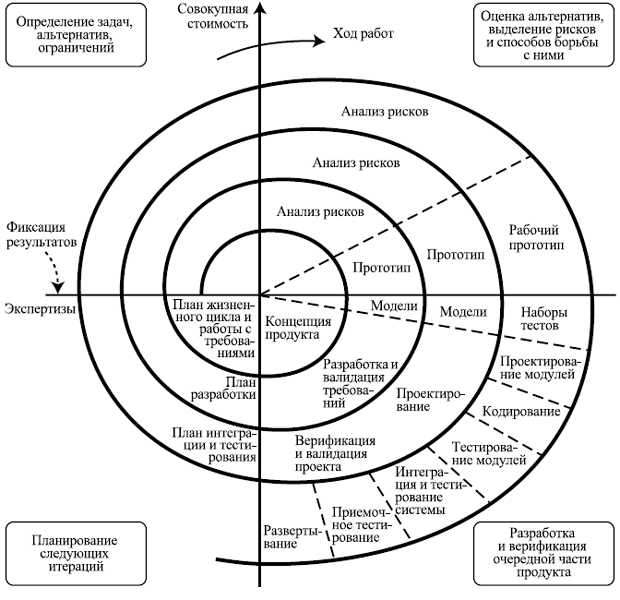
1. Прояснить не ясные требования (прототип UI)
2. Выбрать одно из ряда концептуальных решений (реализация сцинариев)
3. Проанализировать осуществимость проекта

Классификация протопипов:

1. Горизонтальные и вертикальные
2. Одноразовые и эволюционные
3. бумажные и раскадровки

**Горизонтальные** прототипы — моделирует исключительно UI не затрагивая логику обработки и базу данных.  
**Вертикальные** прототипы — проверка архитектурных решений.  
**Одноразовые** прототипы — для быстрой разработки.  
**Эволюционные** прототипы — первое приближение эволюционной системы.  
Модель принадлежит второй группе.

**Спиральная модель жизненного цикла программного обеспечения**

Спиральная модель представляет собой процесс разработки программного обеспечения, сочетающий в себе как проектирование, так и постадийное прототипирование с целью сочетания преимуществ восходящей и нисходящей концепции.  
  
  
  
Преимущества:

* Быстрое получение результата
* Повышение конкурентоспособности
* Изменяющиеся требования — не проблема

Недостатки:

* Отсутствие регламентации стадий

**Этапы равития пррограммирования**

1)Стихийное программирование: с 50 до сер 60х(имели прост структуру на машинном языке, появляются ассемблеры что позволило использовать символьные имена данных (фортран,алгон),   
Появляются средства управления программами и подпрограммамив рез созд: библиотеки, подпрограммы. Изначально подпрограммы работали только с глобальными данными, но в итоге смогли оперировать со всеми.)

2)Структурное программирование 60е по 70е (представляет собой структурный подход, совокупность рекомендуемых технических приёмов. Структурный код требует представление задачи, в виде иерархии подзадач простейших структур. Проектирование подразумевает реализацию общей идеи обеспечивая обработку интерфейсов подпрограмм при этом одновременно вводиться огр на констр алгор, рек опр мод опис, исп метод пошаговой детализации для проэктирования алгоритма. Прцед подход подерживет языки програмирования. Локализ и влечения обл данных. Начинает появляться технологи модульного програм в кот заклад возм огр дост к глоб данным, возможность выявления пользов типов данных. Модуль программ предполагает выделение групп подпрограмм использующие одни и те же данные в разных областях.   
Технология для созд сл програм обеспечения в виде совакупности обьектов каждый из кот является экз опр типа, а классы обр иерархи со св. Взаимод прогр обьектов произв посред передач сообщ.   
Более успешная поз програм обеспеч. И интегрирование их с подпрограммами   
Основанная на механизмах наслед, полиморфизма   
Появление возможности создания библиотек классов для различного применения, появляется возможн проект некоторую часть например интерфейс внутреннего продукта, с применением визуальных средств.   
Явл заг программ.

4)Комбинированный подход, Case технологии сер 90х до наст вр   
Предпологает построение програм обеспеч из отдел компонентов физически существующих частей программного обеспечения, которые взаимодействуют с собой через интерфейс,

Это позволяет программистам создавать продукты частично состоящие из повторных частей

Спецификация — это точное формализованное описание функций и ограничений разрабатываемо ПО. Для получения спецификаций выполняют:

* 1. Анализ требований ТЗ
  2. Формулирование содержания
  3. Выбирают мат аппарат формализации
  4. Строят модель предметной области
  5. Определяют подзадачи, выбирают и разрабатывают методы их решения
  6. Проектирование

**Разница языков с++ и с#**

1. C# - компонентно-ориентированный язык
2. Статичные классы
3. Реализован сборщик мусора
4. Новые типы данных – var и dynamic

**Модули и свойства сцеплений**

* 1. Сцепление по данным – предполагает, что модули обмениваются скалярными данными
  2. Сцепление по образцу – предполагает, что модули обмениваются данными объединенными в структуры
  3. Сцепление по управлению – один модуль посылает другому некий флаг предназначенный для управления внутренней логики модулей
  4. Сцепление по общей области данных – предполагает, что модули работают с общей областью данных и считается что этот тип сцеплений недопустимый
  5. Сцепление по содержимому – когда один модуль содержит обращения к внутренним компонентам другого

**Виды связанности**

* 1. Функциональная связанность – когда все объекты предназначены для одной функции
  2. Последовательная связанность
  3. Информационная связанность
  4. Временная связность
* *int* a, b;  
  a = Convert.ToInt32(Console.ReadLine());  
  b = Convert.ToInt32(Console.ReadLine());  
  *string* result = a > b ? "больше" : "меньше";  
  Console.WriteLine(result);

**Перечисления**

Enum type1 : type2 (default int) {t1:t2, t1:t2}