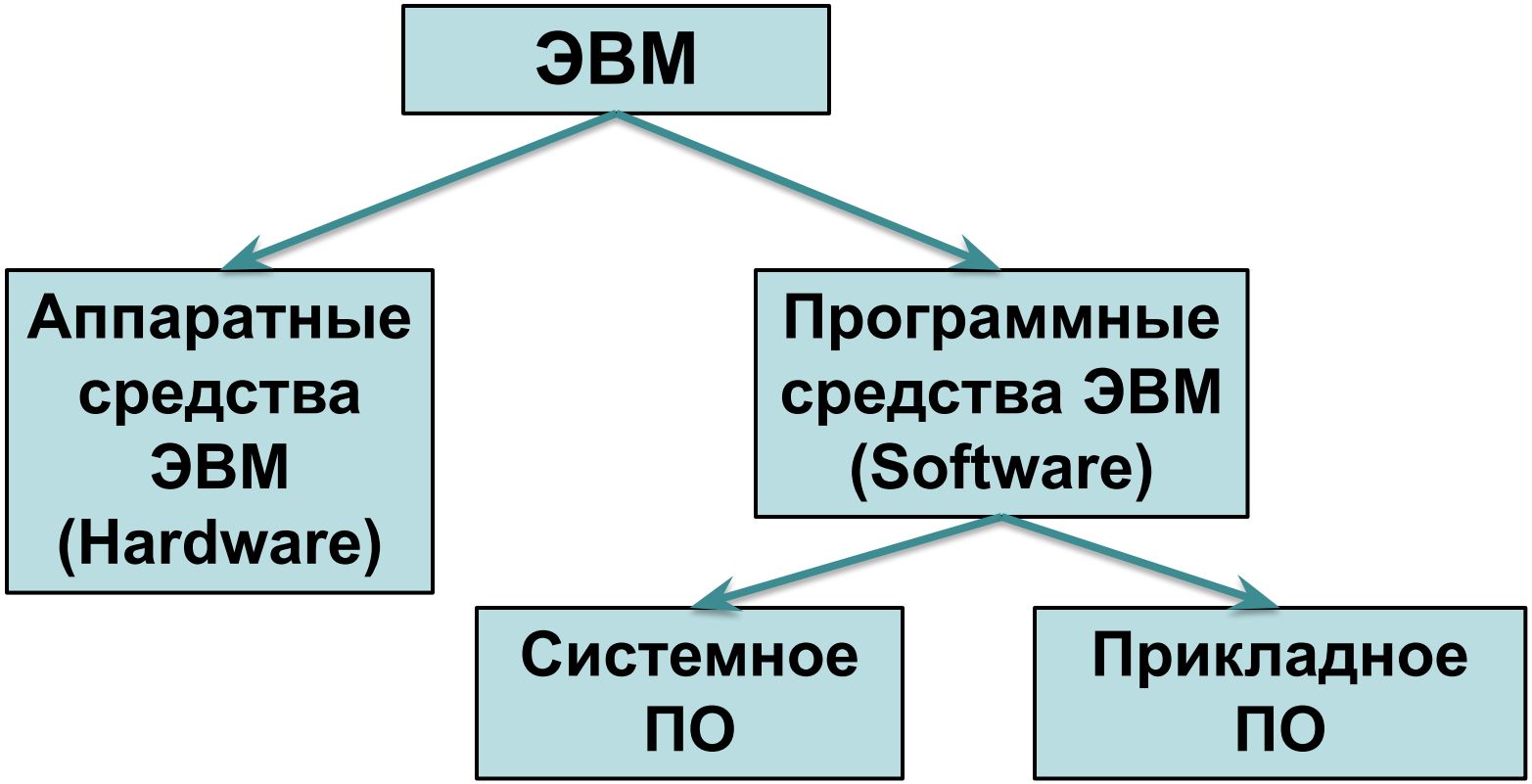
# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

## ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

- Системное программное обеспечение
- Прикладное программное обеспечение

- ОБЩИЕ ВОПРОСЫ РАЗРАБОТКИ ПО
- КЛАССИФИКАЦИЯ МЕТОДОВ ПРОЕКТИРОВАНИЯ ПРОГРАММНЫХ СРЕДСТВ

# ЭВМ как единство двух начал



**Системное ПО** предназначено для обеспечения работоспособности ЭВМ и разработки других программных средств

1. Общесистемное
2. Инструментальное
3. Диагностическое

# 1. Общесистемное ПО

1. Операционные системы (ОС)
2. Операционные оболочки – NC, VC
3. Операционные среды (Win 3.1) – надстройка над DOS
4. Драйверы
5. Утилиты

# Операционная система

*ОС* – программа, которая автоматически загружается в оперативную память и выполняет управление физическими и логическими ресурсами ЭВМ

-- Физические – память, процессор, внешние устройства

-- Логические – программы, файлы, события

(MS DOS, Windows, Windows NT Server, LUNIX)

***Операционная среда*** – надстройка над ОС с развитым пользовательским интерфейсом (Windows 3.1)

**Операционная оболочка** – это программа, которая позволяет более удобно выполнять команды ОС (Norton Commander, Total Commander).

**Драйвер** – набор инструкций или программа, расширяющая возможности ОС по управлению ЭВМ (например, раскладка клавиатуры, управление памятью)

**Утилита** – программа вспомогательного назначения, которая представляет пользователю возможность реализовать набор некоторых действий (обслуживание дисков, шифрование, архивация и пр.). Существуют отдельные утилиты и многофункциональные (Norton Utilities).

# 2. Инструментальное ПО

1. Системы программирования

(включают компилятор или интерпретатор, библиотеки подпрограмм, интегрированную оболочку для редактирования и отладки программ)
(Pascal, Delphi, C++, C#, Assembler, Microsoft Visual Studio).

2. СУБД – системы управления базами данных, которые обеспечивают операции по созданию больших информационных массивов, сортировки, поиску данных, выводу отчетов (Access, FoxPro, Oracle, MS SQL Server)

3. Case-системы – системы, поддерживающие разработку крупных программных средств на протяжении всего жизненного цикла, от моделирования бизнес-процессов до протоколирования всех этапов работы (CASE, Design/IDEF, Designer, BP Win).
   Case-системы являются инструментарием для системных аналитиков и разработчиков программных средств

4. Нетрадиционные средства разработки ПО – инструментальные средства с закрытой непубликуемой технологией (игры, мультимедиа) а также новые специализированные системы разработки игр, мультимедийных приложений, экспертных систем, средств организации WEB- серверов в сети Internet и пр.)

# 3. Диагностическое ПО

1. **Антивирусное программное обеспечение** – это ПО (например, Dr Web, Антивирус Касперского, Norton Antivirus, MS Antivirus и др.), предназначенное для ликвидации последствий или предотвращения заражения ПК специальными программами (вирусами), выполняющими на ПЭВМ нежелательные для пользователя действия

2. Средства тестирования аппаратных устройств – это программы, позволяющие проверить исправность отдельных узлов ПЭВМ, например,
   Vtest – визуальное тестирование качества изображения монитора,
   CoreTest – тестирование винчестера.
3. Диагностическое ПО - для поиска и определения характера неисправности в блоках ЭВМ, например, пакеты Check It, Win Check It.

4. Средства корректировки – это программы, позволяющие произвести настройку режимов работы отдельного узла ЭВМ, например, калибровка цветности монитора.
5. Вспомогательные программные средства – используется при ремонте узлов ЭВМ, например, Screen-Test генерирует тестовые сигналы в процессе ремонта монитора.

**ПРИКЛАДНОЕ ПО - предназначено для решения определенной задачи в конкретной предметной области**

1. **ПО общего назначения** (текстовые, табличные, графические процессоры, электронные секретари, эл. почта, игры)
2. **Специализированное ПО** (САПР, обучающие системы, математические системы, издательские системы, финансовые системы, системы управления проектами)
3. **Нетрадиционное ПО** (системы мультимедиа, интеллектуальные системы: экспертные системы, системы распознавания, перевод текста)

# ПАКЕТЫ ПРИКЛАДНЫХ ПРОГРАММ

(ППП) как средство организации прикладного ПО

1. Проблемно-ориентированные ППП
2. Интегрированные ППП
3. Пакеты ППП для решения научно-технических задач

# Классификация ППП

# 1.Проблемно-ориентированные ППП

Проблемно-ориентированные ППП включают следующие программные продукты:

- Текстовые процессоры
- Настольные издательские системы (НИС)
- Графические редакторы
- Пакеты для работы с векторной графикой
- Электронные таблицы
- Организаторы работ
- Системы управления базами данных (СУБД)
- Пакеты демонстрационной графики
- Пакеты программ мультимедиа
- Системы автоматизации проектирования
- Программы распознавания символов
- Финансовые, аналитико- статистические

# 2. Интегрированные ППП

Традиционные, или полно связанные, интегрированные комплексы представляют собой многофункциональный автономный пакет, в котором в одно целое соединены функции и возможности различных специализированных пакетов, родственных в смысле технологии обработки данных на отдельном рабочем месте

# 3. Пакеты ПП для решения научно-технических задач

- Пакет прикладных программ представляет собой набор подпрограмм, объединяемый управляющей программой и предназначенный для решения конкретных задач в какой-либо области знаний
- Обычно все подпрограммы делаются свободными от ввода-вывода и размер массивов указывается условный. Программы ввода-вывода выполняются в виде отдельного модуля
- В зависимости от структуры ППП модули могут быть различных структур:
  - простой
  - оверлейной
  - динамически последовательной
  - динамически параллельной

# Общие вопросы разработки программных средств

1. Жизненный цикл ПО
2. Этапы решения научно-технических задач

# Жизненный цикл ПО

Жизненный цикл программного обеспечения (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Этот цикл — процесс построения и развития ПО.

# Этапы решения научно-технических задач на ЭВМ

1. **Постановка задачи** (описывается цель решения задачи, проблема, подробное содержание характеристик, условия задачи, входные и выходные данные)
2. **Математическое описание** (все существующие соотношения между величинами выражаются посредством математических формул, формируется математическая модель задачи с определенной точностью и ограничениями и допущениями, математическая модель должна быть реалистичной и реализуемой)

3. **Выбор и обоснование метода решения** (одну и ту же задачу можно решать различными методами: процедурное программирование, объектно-ориентированное программирование, использование известного ПО)
4. **Проектирование** (создается общая структура программы, описывается взаимодействие между компонентами программы, блок-схема)
5. **Кодирование** (все конструкции, записанные на языке проектирования, переводятся на язык программирования высокого уровня)

6. **Тестирование** (всесторонняя проверка программы на *правильность*, *эффективность*, на *вычислительную сложность*- состоит в экспериментальном сравнении двух алгоритмов, решающих одну и ту же задачу)
7. **Составление рабочей документации** (требования ЕСПД: описание применения, руководство пользователя, руководство программисту)
8. **Сопровождение** (этапы эксплуатации программы: обучение пользователей, обновления программы, консультации)

# ОСНОВНЫЕ НАПРАВЛЕНИЯ В программировании

- **ПРОЦЕДУРНОЕ программирование**
- **МОДУЛЬНОЕ программирование**
- **ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ программирование**

# 1. Процедурное программирование

- В процедурном программировании основное внимание уделяется алгоритмам, т.е. некоторой последовательности действий, выполнение которых приводит к определенному результату.
- Языки программирования, которые поддерживают эту модель, называются процедурными, и главное внимание в них уделяется построению подпрограмм (процедур).

# Основные вопросы процедурного программирования:

- передача аргументов в процедуры
- получение вычисленных значений из процедур.

Первыми процедурными языками были FORTRAN, ALGOL 60, PASKAL, СИ.

При разработке большой программы коллективом разработчиков необходимо решать очень сложные проблемы (надо договариваться об используемых именах, об организации общих данных и способах доступа к ним).

# В процедурном программировании

Алгоритм записывается на выбранном языке программирования с помощью команд

- описания данных,
- вычисления значений,
- управления последовательностью выполнения программы.

**Используемые типы данных**

## 1. Переменные и константы

Реальные данные, с которыми работает программа, - это *числа*, *строки* и *логические величины*. Эти типы называются *базовыми*.

## 2. Числовые данные

## 3. Арифметические операции

«+» (сложение)

«-» (вычитание)

«*» (умножение)

«/» (деление)

## 4. Арифметические выражения

**С помощью арифметических операций формируются арифметические выражения, которые состоят из операций и *операндов* (переменных и констант).**

## 5. Логические выражения

**При записи логических выражений используются операции сравнения и логические операции. Операции сравнения сличают значения правого и левого операндов. Результатом сравнения является *true*, если оно удачно, и *false* в противном случае.**

| Описание | Паскаль, Бейсик | Си ++ |
|---|---|---|
| Равно | = | = = |
| Не равно | < > | ! = |
| Меньше | < | < |
| Меньше или равно | < = | <= |
| Больше | > | > |
| Больше или равно | > = | > = |

| Логическая операция | Бейсик | Паскаль | Си ++ |
|---|---|---|---|
| И | AND | and | **&&** |
| ИЛИ | OR | or | **\|\|** |
| НЕ | NOT | not | **!** |

## 6. Строчные выражения

## 7. Указатели

*Указатели* – адреса физической памяти

## 8. Структуры

Современные языки программирования позволяют применять сложные типы данных, составляющиеся из базовых и определенных ранее сложных типов.

В результате удается организовать структуры данных произвольной сложности: списки, деревья и т.п. При этом структура объединяет группу разных данных под одним названием. Например, чтобы организовать обработку данных по студентам, в программе удобно не просто описать десяток различных переменных, а объединить их в структуру «студент», состоящую из полей разного типа «имя», «пол», «группа» и т. д.

## 9. Массивы

**Массив** – сложный тип данных, доступ к элементам которого происходит по их положению, по номеру или индексу. Например, можно описать массив, состоящий из тысячи элементов численного типа, и затем обратиться к десятому или сотому элементу по его номеру.

## 10. Операторы

## 11. Комментарии

**Комментарии** – часть исходных текстов, выделяемых с помощью специальных обозначений.

| | **Бейсик** | **Паскаль** | **Си ++** |
|---|---|---|---|
| Однострочные комментарии | REM или ‘ | // | // |
| Многострочные комментарии | нет | { } или (* *) | /* */ |

# Процедуры и функции

Подпрограммы бывают двух видов:

- Процедуры
- Функции

Отличаются они тем, что процедура просто выполняет группу операторов,

а функция вдобавок вычисляет некоторое значение и передает его обратно в главную программу (*возвращает* значение).

# Структура процедуры

| | Бейсик | Паскаль | Си ++ |
|---|---|---|---|
| **Заголовок процедуры** | SUB имя (список_параметров) | procedure имя (список_параметров) ; | Void имя (список_параметров) |
| **Тело** | Последовательность операторов | begin<br><br>Последовательность операторов<br><br>end; | {<br><br>Последователь-ность операторов<br><br>}; |
| **Завершение** | END SUB | нет | нет |

# Структура функции

| | **Бейсик** | **Паскаль** | **Си ++** |
|---|---|---|---|
| *Заголовок функции* | FUNCTION имя (список_параметров) | Function имя (список_параметров) : тип_функции; | тип_функции имя (список_параметров) |
| *Тело* | Последовательность операторов | begin<br><br>Последовательность операторов<br><br>end; | {<br><br>Последовательность операторов<br><br>}; |
| *Завершение* | END FUNCTION | нет | нет |

# 2. Модульное программирование

В модульном программирование основные акценты переносятся на построение модулей и их взаимодействию в программе.

**Модуль** – это множество взаимосвязанных подпрограмм (процедур) вместе с данными, которые эти процедуры обрабатывают.

Основная цель этого направления состоит в *скрытии данных в модулях*, что не позволяет обратиться к ним из-за пределов модуля.

# Организация данных, а не алгоритмов – это основная задача модульного программирования

- При создании ПО  необходимо определить все модули, которые будут использоваться, и разделить программу на модули так, чтобы ее данные были скрыты в этих модулях.
- Модуль – это самостоятельная часть программы, которая разрабатывается одним программистом, например.
- Поскольку доступ к данным из-за пределов модуля запрещен (скрыт), то соответственно, предотвращено их случайное изменение  (ошибки в программе).

- Язык MODULA2 был специально сконструирован для поддержки модульного программирования.
- Языки  C++, C#  не были специально для этого созданы, однако реализованная в них концепция **классов**, позволила работать с модулями.
- Эти языки содержат все необходимое для поддержки как процедурного, так и модульного программирования. Эти направления дополняют друг друга, а не исключают.

# 3. Объектно-ориентированное программирование

Программа представляется в виде набора **объектов**, взаимодействующих между собой посредством сообщений.

**Объект = данные + процедуры**

**Объект** – совокупность свойств (структур данных, характерных для этого объекта), методов их обработки (подпрограмм изменения свойств) и событий, на которые данный объект может реагировать и которые приводят, как правило, к изменению свойств объекта.

# Для описания объектов служат классы

*Класс* определяет свойства и методы объекта, принадлежащего этому классу. Соответственно, любой объект можно определить как *экземпляр класса*.

Итак, что же такое класс, и что такое объект, а главное — «в чем разница». В качестве примера рассмотрим лифт в доме.

Класс — это проект лифта. Именно проект. Ну, т.е., то, каким он будет. У него есть высота, ширина, скорость — это свойства. Он умеет ездить — вверх/вниз, это методы. То, что стоит у нас в домах — это экземпляры класса «Лифт», у которых есть свой номер (у каждого).

А интерфейс — это кнопки лифта. По проекту мы можем наштамповать этих лифтов столько, сколько угодно. Можем менять им в процессе цвет двери, например (паблик чтение/запись свойства), но не можем менять скорость, которая постоянна (паблик рид свойство). У него есть много внутренних свойств, которые нам неведомы, но благодаря им лифт работает (https://habr.com/users/Tomcat)

**Существует 4 важнейших механизма объектно-ориентированного программирования:**

1. **Наследование**
2. **Полиморфизм**
3. **Инкапсуляция**
4. **Абстракция**

# Наследование

Важнейшая характеристика класса – возможность создания на его основе новых классов с **наследованием** всех его свойств и методов и добавлением собственных.

Класс, не имеющий предшественника, называется *базовым*.

**Наследование** – это свойство системы, позволяющее описать новый класс на основе уже существующего, с частично или полностью заимствующейся функциональностью.

Класс, от которого производится наследование, называется базовым или родительским.

Новый класс называется потомком, наследником или производным классом.

Представим себя инженерами автомобильного завода.
Нашей задачей является разработка современного автомобиля. У нас уже есть предыдущая модель, которая отлично зарекомендовала себя в течение многолетнего использования. Поскольку времена и технологии меняются, то завод должен стремиться повышать комфорт продукции и соответствовать современным стандартам.
Нам необходимо выпустить целый модельный ряд автомобилей: седан, универсал и малолитражный хэтч-бэк.
Очевидно, что мы не собираемся проектировать новый автомобиль с нуля, а, взяв за основу предыдущее поколение, внесём ряд конструктивных изменений. Например, добавим гидроусилитель руля и уменьшим зазоры между крыльями и крышкой капота, поставим противотуманные фонари. Кроме того, в каждой модели будет изменена форма кузова.Очевидно, что все три модификации будут иметь большинство свойств прежней модели (двигатель, ходовая часть, коробка передач и т.д.).
При этом каждая из моделей будет реализовать некоторую новую функциональность или конструктивную особенность.
В данном случае, мы имеем дело с наследованием.
(Пример от  Михаила Пайсона https://habr.com/users/Tomcat/)

# Полиморфизм

Полиморфизм – возможность использования методов с одинаковыми именами для обработки данных разных типов.

Любое обучение вождению не имело бы смысла, если бы человек, научившийся водить, скажем, ВАЗ 2106 не мог потом водить ВАЗ 2110 или BMW X3. С другой стороны, трудно представить человека, который смог бы нормально управлять автомобилем, в котором педаль газа находится левее педали тормоза, а вместо руля – джойстик. Всё дело в том, что основные элементы управления автомобиля имеют одну и ту же конструкцию и принцип действия. Водитель точно знает, что для того, чтобы повернуть налево, он должен повернуть руль, независимо от того, есть там гидроусилитель или нет. Водитель может управлять автомобилем, выполняя одни и те же действия, независимо от того, какой именно тип автомобиля он использует. По сути, можно сказать, что все автомобили имеют один и тот же интерфейс, а водитель, абстрагируясь от сущности автомобиля, работает именно с этим интерфейсом. Независимо от того, каким образом будет реализовываться движение и внутреннее функционирование машины, интерфейс останется прежним.
**Полиморфизм** – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.
(Пример от Михаила Пайсона https://habr.com/users/Tomcat/)

# Инкапсуляция

Инкапсуляция - свойство языка программирования, позволяющее объединить и защитить данные и код в объект и скрыть реализацию объекта от пользователя (прикладного программиста) .

При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Пользователь может взаимодействовать с объектом только через этот интерфейс.

А в двух словах — это понятие предельно просто *in-capsulo*, «в капсуле». Внутренняя логика скрыта в классе «как в капсуле» и не видна снаружи. Виден только интерфейс, т.е. набор методов, который действительно хотели показать.

Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

Представим, что мы оказались в конце позапрошлого века, когда Генри Форд ещё не придумал конвейер для автомобилей. Представим, что для управления первым паровым автомобилем необходимо было знать, как устроен паровой котёл, постоянно подбрасывать уголь, следить за температурой, уровнем воды. При этом для поворота колёс использовать два рычага, каждый из которых поворачивает одно колесо в отдельности. Тут можно согласиться с тем, что вождение автомобиля того времени было весьма неудобным и трудным занятием.

Теперь вернёмся в сегодняшний день к современным чудесам автопрома с коробкой-автоматом. На самом деле, по сути, ничего не изменилось. Бензонасос всё так же поставляет бензин в двигатель, дифференциалы обеспечивают поворот колёс на различающиеся углы, коленвал превращает поступательное движение поршня во вращательное движение колёс.

Прогресс в другом. Сейчас все эти действия скрыты от пользователя и позволяют ему крутить руль и нажимать на педаль газа, не задумываясь, что в это время происходит с инжектором, дроссельной заслонкой и распредвалом. Именно сокрытие внутренних процессов, происходящих в автомобиле, позволяет эффективно его использовать даже тем, кто не является профессионалом-автомехаником с двадцатилетним стажем. Это сокрытие внутренних процессов в ООП носит название инкапсуляции.

(Пример от  Михаила Пайсона https://habr.com/users/Tomcat/)

## Абстракция

в объектно-ориентированном программировании – это придание объекту характеристик, которые чётко определяют его концептуальные границы, отличая от всех других объектов, при этом особенность выбранных характеристик такова, что при работе с объектами не потребуется вникания в особенности реализации объектов

Абстракция - пример

Представьте, что водитель едет в автомобиле по оживлённому участку движения. Понятно, что в этот момент он не будет задумываться о химическом составе краски автомобиля, особенностях взаимодействия шестерён в коробке передач или влияния формы кузова на скорость (разве что, автомобиль стоит в глухой пробке и водителю абсолютно нечем заняться). Однако, руль, педали, указатель поворота (ну и, возможно, пепельницу) он будет использовать регулярно.

**Абстрагирование** – это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, **абстракция** – это набор всех таких характеристик.

(Пример от  Михаила Пайсона https://habr.com/users/Tomcat/)

# Виды ошибок в программировании

- Существуют различные типы программных ошибок. Поскольку машины все чаще используются в автоматическом режиме, с бортовыми встраиваемыми системами или компьютерами, контролирующими их функционирование, программная ошибка может иметь серьезные последствия.
- Известны случаи, когда космические челноки и самолеты, разбивались из-за ошибки в программном обеспечении во встраиваемом компьютерном оборудовании. Одна лазейка, оставленная в коде операционной системы, может обеспечить точку входа для хакеров, которые могут использовать эту уязвимость. К этим ошибкам нужно относиться очень и серьезно, так как мы все больше и больше полагаемся на компьютеры. Original: http://juice-health.ru/programming/102

- Логическая ошибка

Это, пожалуй, наиболее серьезная из всех ошибок. Если написанная программа на любом языке компилируется и работает правильно, но выдает неправильный вывод, недостаток заключается в логике основного программирования. Это ошибка, возможно, была унаследована от недостатка в базовом алгоритме. Поэтому сама логика, на которой базируется вся программа, является ущербной. Чтобы найти решение такой ошибки нужно фундаментальное изменение алгоритма.

- Original: http://juice-health.ru/programming/102

- Синтаксическая ошибка

Каждый компьютерный язык, такой как C, Java, Perl и Python имеет специфический синтаксис, в котором будет написан код. Если программист не придерживается "грамматики" спецификаций компьютерного языка, возникнет ошибка синтаксиса.

Такого рода ошибки легко устраняются на этапе компиляции.

Original: http://juice-health.ru/programming/102

- Ошибка компиляции

Компиляция это процесс, в котором программа, написанная на языке высокого уровня, преобразуется в машиночитаемую форму. Многие виды ошибок могут происходить именно на этом этапе, в том числе и синтаксические ошибки. Иногда, синтаксис исходного кода может быть безупречным, но ошибка компиляции все же может произойти. Это может быть связано с проблемами в самом компиляторе. Эти ошибки исправляются на стадии разработки.

- Original: http://juice-health.ru/programming/102

# Ошибки среды выполнения (RunTime)

Программный код успешно скомпилирован, и исполняемый файл был успешно создан. Далее запускается  исполняемый файл,, чтобы проверить работу программы.

Ошибки при выполнении программы могут возникнуть в результате аварии или недостатка ресурсов носителя. Разработчик должен был предвидеть реальные условия развертывания программы.

Возникшие ошибки среды выполнения можно исправить, вернувшись к стадии кодирования.

Original: http://juice-health.ru/programming/102

- Арифметическая ошибка

Многие программы используют числовые переменные или константы, и алгоритм может включать несколько математических вычислений. Арифметические ошибки возникают, когда компьютер не может справиться с проблемами, такими как «деление на ноль", или ошибками, ведущими к бесконечному результату. Это снова приводит к логической ошибке, которая может быть исправлена  только путем изменения алгоритма.

Original: http://juice-health.ru/programming/102

- Ошибка ресурса

Ошибка ресурса возникает, когда значение переменной переполняет максимально допустимое значение.

Переполнение буфера, использование неинициализированной переменной, нарушение прав доступа и переполнение стека - примеры некоторых распространенных ошибок.

Original: http://juice-health.ru/programming/102

# Ошибка взаимодействия

- Ошибка может возникнуть  в связи с несоответствием программного обеспечения аппаратному  интерфейсу или интерфейсу прикладного программирования.
- В случае веб-приложений, ошибка интерфейса может быть результатом неправильного использования веб-протокола.

Original: http://juice-health.ru/programming/102

# Вывод по типам ошибок ПО

- Интенсивное тестирование и фаза отладки неотъемлемая часть цикла разработки программного обеспечения, которое может помочь пресечь эти ошибки в зародыше, прежде чем произойдет полномасштабное развертывание программного обеспечения.
- Много ошибок можно избежать с помощью предварительного планирования во время стадии кодирования. Большинство ошибок можно исправить в процессе разработки программного обеспечения через практику и строгие процедуры отладки. Ошибки являются частью обучения, и их никогда нельзя полностью избежать.
  Original: http://juice-health.ru/programming/102

# Главная уязвимость — человеческий фактор.

- Практика не устает доказывать, что в любом ПО самая главная уязвимость — человеческий фактор. И неважно, появился ли баг из-за нехватки квалифицированности специалиста, выжил ли после долгих часов дебага в поисках ошибки, или считался хитро замаскировавшейся фичей. Среди разработчиков даже укрепилось мнение, что баги в принципе существуют всегда и везде. В идеальной ситуации баги исправляют все и сразу. Но в жизни всегда есть задачи, отодвигающие полный и бесповоротный багфикс (новый функционал, срочные хотфиксы, расставленные приоритеты при исправлении багов). Это значит, что в первую очередь находятся и исправляются очевидные проблемы. Остальные тихо ждут своего часа, превращаясь в бомбы замедленного действия. Иногда ошибки вызывают настоящие катастрофы.

# Известные примеры программных ошибок

- **США, 1962 год.** Гибель несущего аппарата “Маринер-1”. Причина – ошибка в одном символе программы
  - DO 100 I = 1, 10
  - DO100I = 1.10
- **США, 1987 год.** Ускоритель Therac-25. Переоблучение пациентов онкоклиник. Причина – ошибка «race condition»
- **США, 1991 год.** Комплекс Patriot. Погибло 28 чел. Причина – ошибка округления
- **Европа, 1996 год.** Ракета Ариан-5. Ущерб 7 млрд. $. Причина – использование унаследованного кода

Катастрофические последствия программных ошибок. Блог компании Mailru

https://habr.com/ru/company/mailru/blog/370153/

История вобрала в себя большинство классических проблем тестирования.

На эту тему есть хорошая статья в wiki, с нее можно начать чтение на эту тему, а затем можно ознакомиться с большой статьей

«Мифы о безопасном ПО: уроки знаменитых катастроф».