ИНФОРМАТИКА

Направление подготовки бакалавра 11.03.02 ИТСС

Беленький Павел Павлович
кандидат педагогических наук, доцент кафедры ОНП

ИНФОРМАТИКА

Модуль 6. Технологии программирования
Интегрированные среды программирования.
Этапы решения задач на компьютерах.
Структурное программирование. Модульный принцип программирования. Подпрограммы. Принципы проектирования программ сверху-вниз и снизу-вверх.
Объектно-ориентированное программирование.

Алгоритмизация и программирование. Языки программирования высокого уровня.
Алгоритм и его свойства. Способы записи алгоритма.
Линейная алгоритмическая структура. Разветвляющаяся алгоритмическая структура. Циклические алгоритмические структуры.
Основные операторы циклов и ветвления
Типовые алгоритмы. Рекурсивные алгоритмы.
Эволюция и классификация языков программирования.
Структуры и типы данных языка программирования.
Трансляция, компиляция и интерпретация.

ИНФОРМАТИКА

Технологии программирования

Технология появляется тогда, когда появляется производство.

Производство находится где-то между искусством и ремеслом.

Ремесло не обладает технологией в силу ограниченности решаемых задач, хотя имеет множество способов и приёмов.

Искусство не обладает технологией в силу уникальности каждой решаемой задачи, а потому набор способов и методов решения поставленной задачи часто точно также уникален и малоприменим, нередко совершенно не применим, при решении другой задачи.

В искусстве там, где оно смыкается с наукой, происходит появление и оттачивание новых технологий и в этом смысле искусство стоит на плечах технологии производства.

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения.

Технологии программирования

Краткая история технологии программирования

Определим программирование, как процесс разработки/создания программ.

Программа: также совершенно естественно определяется, как последовательность команд выполняемых центральным процессорным устройством (ЦПУ).

Попытка избавиться от рутинных процессов, а заодно с ними и от источника ошибок – программиста, прослеживается по всей истории программирования.

Появление трансляторов принесло качественные изменения в технологию программирования.

Транслятор это программа, переводящая один класс утверждений в другой.

Объект, получаемый транслятором в результате трансляции, называется объектный модуль.

Программа разрешающая все внешние ссылки объектных модулей называется редактор связей.

Редактор связей настраивал программу на конкретный адрес загрузки, получая на выходе загрузочный модуль (EXE – модуль).

Сам же загрузочный модуль каким-то образом должен был попасть в память машины и для того, чтобы он работал, ему необходимо было передать управление.

Загрузку загрузочного модуля в память и передачу ему управления выполняла программа, называемая загрузчик. При нормальном завершении исполняемой программы, управление вновь возвращалось к загрузчику и он был готов загрузить следующую программу.

Производительность труда программиста при работе на языке в десять раз выше, чем при работе в коде, что с лихвой окупало все появившиеся накладные расходы.

ИНФОРМАТИКА

Технологии программирования

Программа перестала быть чисто техническим объектом.

Программа в коде, написанная на листе бумаге, отдельно от машины представляла собой колонки никому не нужных цифр. Прочесть её сходу затруднялся даже автор.

Программа на алгоритмическом языке могла быть прочитана и относительно легко разобрана практически любым грамотным человеком, а не только программистом.

Программа обладает следующими свойствами:

- это технический объект;
- это совокупность некоторых математических утверждений;
- это авторское произведение;
- это, безусловно, товар.

Опираясь на то, что программа является техническим объектом, можно смело утверждать, что к программированию можно применить термин технология.

ИНФОРМАТИКА

Технологии программирования

С приходом системы IBM/360 программирование стало по настоящему промышленным производством программ. Система IBM вытеснила с рынка все ранее существовавшие ЭВМ и этим унифицировала рынок и косвенным образом стандартизовала программирование.

Операционная система ещё более отдалила программиста от машины. Произошло разделение на системных и проблемных программистов.

Грамотный программист обычно знал один – два алгоритмических языка, мог работать на ассемблере. Перейти с одного языка на другой для него не составляло проблемы.

Программисты научились писать большие программные комплексы. Не трансляторы или редакторы связей; тогда трудоёмкость этих программ измерялась цифрами порядка 10 человеко-лет. Речь идёт об операционных системах, трудоёмкость которых тогда оценивалась в несколько сотен человеко-лет.

ИНФОРМАТИКА

Технологии программирования

Следующий этап начался с широким приходом персональных электронных вычислительных машин (ПЭВМ).

С точки зрения технологии программирования этот этап характеризуется следующими явлениями:

Появлением, так называемых, «коробочных» программных продуктов.

Победным шествием языка С, а затем С++.

Внедрением ООП (объектно – ориентированное программирование) в широкую практику;

Появлением визуального программирования;

Внедрением сетевых технологий.

Появлением многокомпонентного программирования.

ИНФОРМАТИКА

Технологии программирования

Программирование стало более доступным и внешне совсем простым.

Появилось множество программных сред со средствами программирования.

Молчаливо предполагалось, что любой (!) сможет этими средствами реализовывать для себя специфические задачи, возникающие в процессе обработки данных, что делает, в свою, очередь ненужным наличие программиста.

Практика показала, что 90% пользователей не собираются программировать.

Оставшиеся 10% используют средства программирования на любительском уровне, что естественно, так как они не программисты.

Таким образом, реально спрос на программистов лишь увеличился. Тем не менее, появление сред со средствами программирования привело к появлению класса программистов – любителей.

ИНФОРМАТИКА

Технологии программирования

Технологические процессы и жизненный цикл программного обеспечения

В программировании выделяют следующие технологические процессы:

- постановка;
- разработка;
- эксплуатация.

В Соединённых Штатах делают более подробное деление:

- требования/спецификации;
- проектирование;
- реализация;
- отладка;
- Внедрение;
- сопровождение/экплуатация.

Процесс разработки и эксплуатации программы часто называют жизненным циклом программного обеспечения или программы.

Постановку определим, как процесс получения документа называемого «Задание на программирование». Иначе его могут называть «Техническое задание».

Стадия постановки чрезвычайно важна. Она определяет успех всего проекта.

Постановку выполняют чаще всего одни специалисты, а программируют другие.

В науке есть утверждение: решить задачу может каждый, а вот поставить – нет.

Что значит поставить задачу? Поставить задачу – значит описать, как её решить.

Таким образом, в постановке должно быть описано, как решить задачу, то есть постановка должна содержать с одной стороны алгоритм решения задачи, а с другой стороны конкретные спецификации задачи: входные/выходные документы, технологические допуски на данные; сами данные и их форматы и т.д.

ИНФОРМАТИКА

Технологии программирования

Цели программного обеспечения зависят от задач. Но необходимо сделать несколько кратких замечаний об общих правилах формулирования целей.

Цели должны быть чёткими, явными, разумными и измеримыми.

Цель, которую нельзя понять – бесполезна.

Цели должны быть известны всем разработчикам проекта.

Цели должны быть достижимы.

Все цели необходимо формулировать по возможности в количественных терминах.

Каждая цель должна быть сформулирована достаточно подробно.

Спецификации – хотя традиционно эту фазу относят к требованиям и целям, спецификации равно на столько принадлежат и процессу проектирования.

К спецификациям относят описание входных, выходных и внутренних обменных данных, а также некоторых состояний программной системы.

Детальные спецификации должны обязательно иметь:
.Описание входных данных.

2. Описание выходных данных.

3.Защита системы: Необходимо перечислить все уровни защиты системы. Кто к какой информации имеет доступ и какие функции может запускать.

4. Эффективность: Минимальные требования к скорости работы системы и потребляемым ресурсам.

Кроме того, в спецификации должен входить «контрольный пример». Контрольным примером называется один или более конкретных наборов входных данных с соответствующими им выходными данными.

ИНФОРМАТИКА

Технологии программирования

Разработка – собственно процесс написания и отладки программ.

Разработка архитектуры – это процесс разбиения большой системы на более мелкие части.

Процесс разработки архитектуры – шаг, необходимый при проектировании систем, но не обязательный при создании программы.

Если вы разрабатываете программную систему, то следующий шаг проектирования – разработка архитектуры, а за ним – проектирование структуры программы.

Проектирование сверху вниз.
Смысл проектирования сверху вниз состоит в том, что оно дает обозримое представление взаимосвязи всех составных частей проекта.

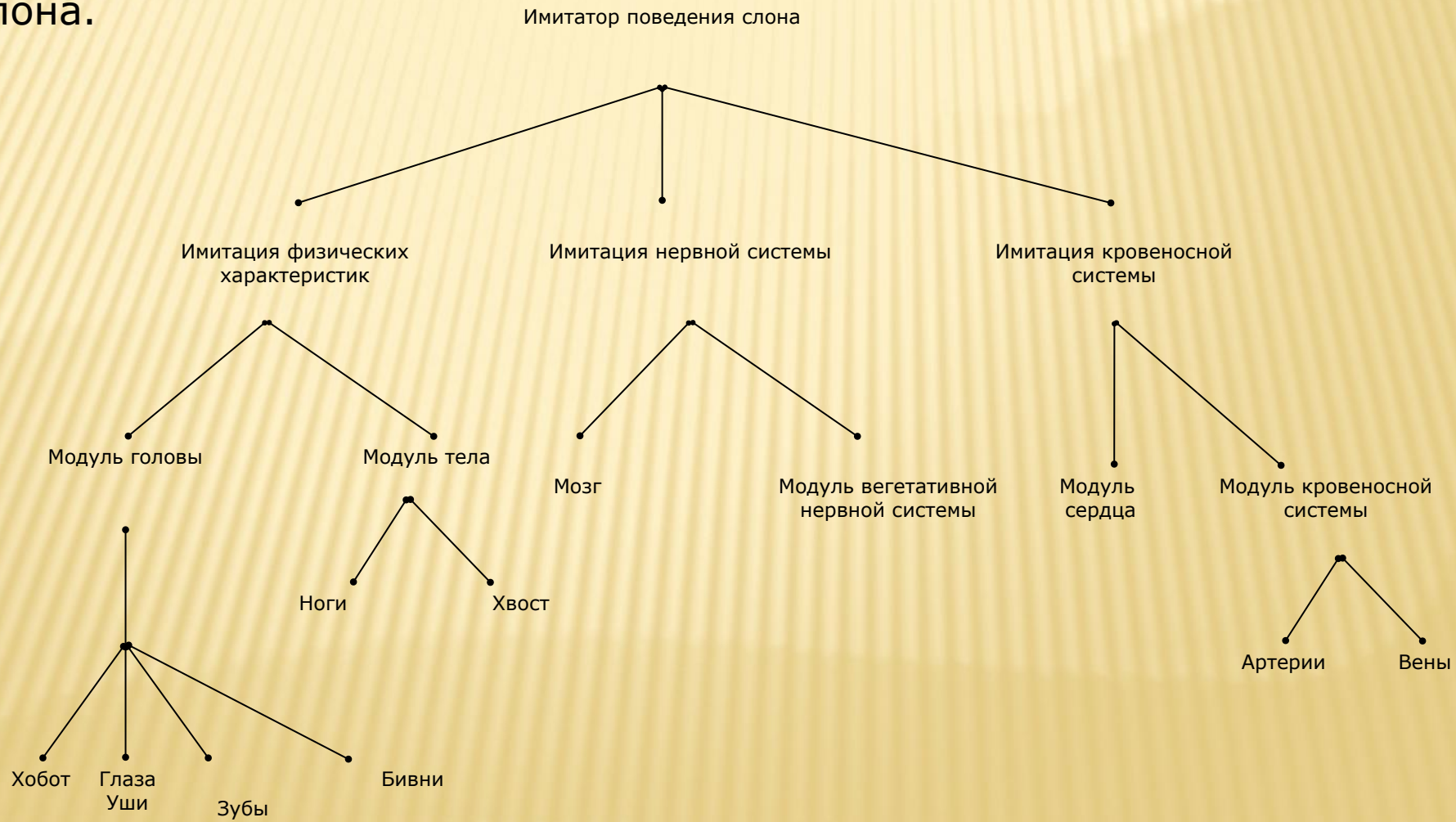
Такой подход позволяет своевременно замечать возникающие проблемы и не переходить к последующей детализации до тех пор, пока полностью не завершён предыдущий уровень. Рассматриваемый подход иногда называют «иерархической декомпозицией» из-за ступенчатой природы процесса.

ИНФОРМАТИКА

Технологии программирования

Допустим, вам предложили построить программу имитации слона.

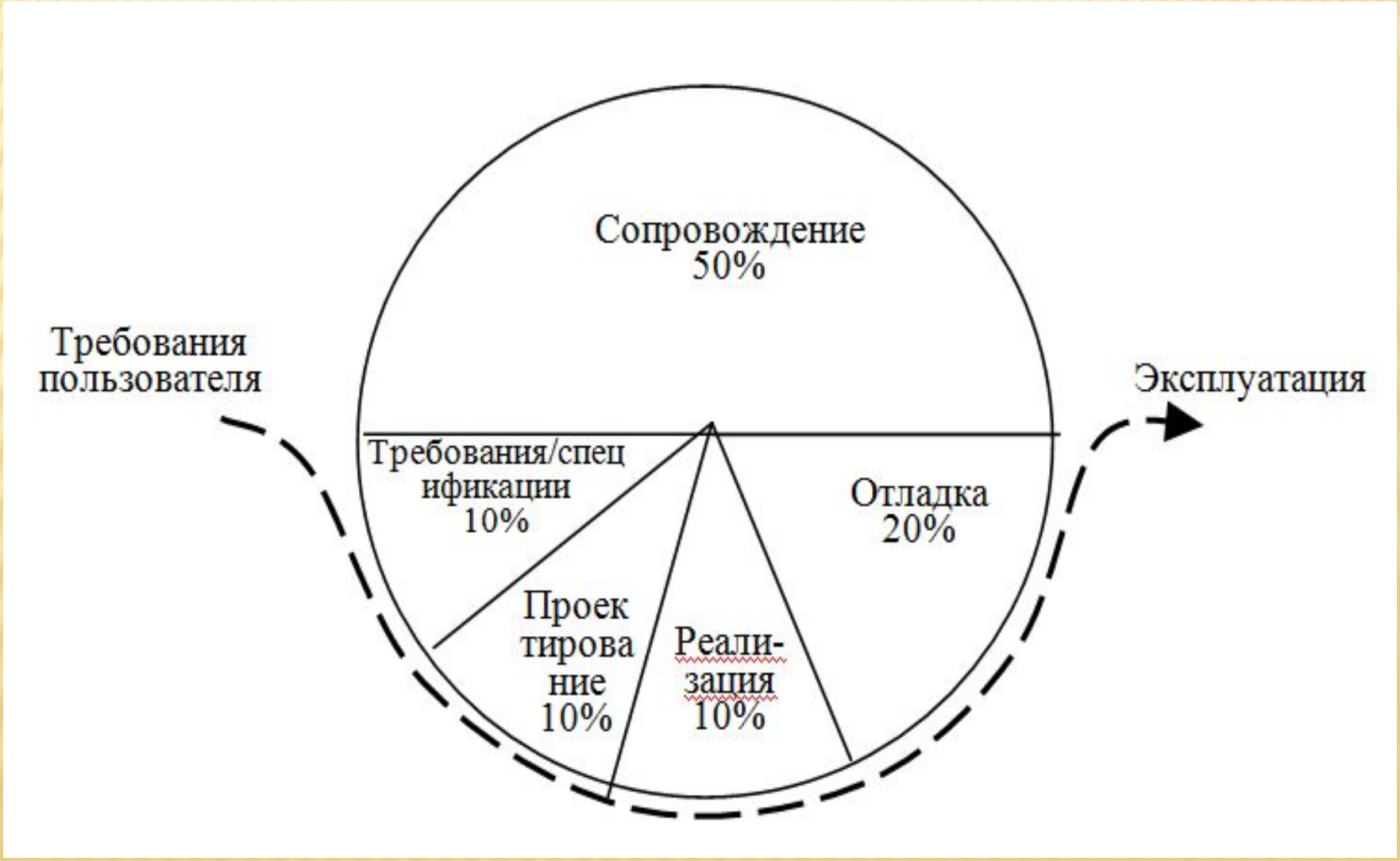
Вы делаете иерархическую декомпозицию отдельных частей имитатора слона.



ИНФОРМАТИКА

Технологии программирования

Рассмотрим, сколько занимает каждая стадия в «жизни программы».



ИНФОРМАТИКА

Технологии программирования

При проектировании возникает до 61% - 64% ошибок и лишь остальные при реализации.

Ошибки, допущенные при проектировании, влияют на все последующие стадии, а их ликвидация обходится гораздо дороже.

Иной раз ошибки, пропущенные на этой стадии, приводят к пересмотру всей схемы реализации программы.

Ошибки делаемые программистом в большинстве случаев менее существенны, чем концептуальные ошибки, получаемые при проектировании.

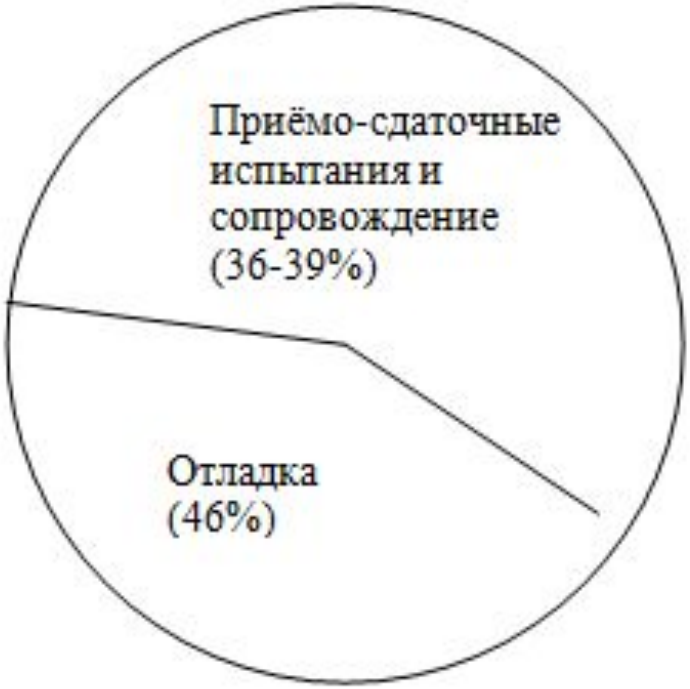
ИНФОРМАТИКА

Технологии программирования

Жизненный цикл программного обеспечения: распределение допущенных ошибок по фазам

ИНФОРМАТИКА

Технологии программирования



Жизненный цикл программного обеспечения: распределение выявленных ошибок по фазам.

ИНФОРМАТИКА

Технологии программирования

Жизненный цикл программного обеспечения:
распределение стоимости устранения ошибок по фазам

ИНФОРМАТИКА

Технологии программирования

Блок-схемы и алгоритмы, программы и подпрограммы

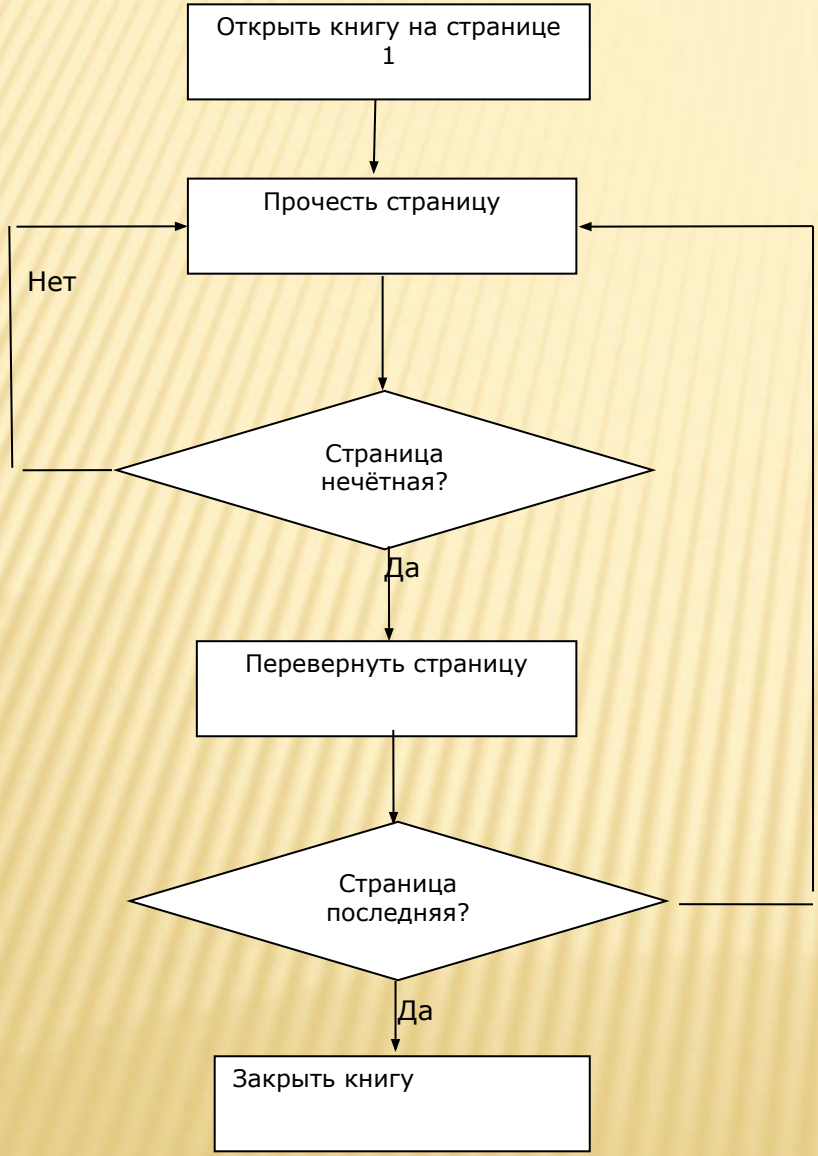
Наиболее известная форма представления проекта – блок-схема.

Блок-схема – графическое представление проекта решения задачи. Действия описаны в прямоугольниках, условные выборы в ромбиках, последовательность действий передаётся стрелками, соединяющих прямоугольники или ромбики.

Проект может выглядеть, например, следующим образом:

ИНФОРМАТИКА

Технологии программирования



Блок-схемы отображают некий обобщённый взгляд на действия, без достаточной детализации.

Блок-схемы, прежде всего, отображают **алгоритм** решения задачи.

ИНФОРМАТИКА

Технологии программирования

Алгоритм – последовательность «шагов» или действий решающих поставленную задачу. В этом смысле, понятие алгоритм очень близко к понятию «рецепт», «процесс», метод, способ и т.д.

Для того чтобы способ или метод стал алгоритмом он должен обладать, по крайней мере, пяти свойствами.

.Конечность (финитность). Алгоритм всегда должен заканчиваться после конечного числа шагов.

.Определённость. Каждый шаг алгоритма должен быть точно определён.

.Ввод. Алгоритм имеет некоторое, возможно равное нулю, число входных данных. Эти величины заданы до начала работы.

.Вывод. Алгоритм имеет одну или несколько выходных величин, определённым образом связанных с входными данными.

.Эффективность. Все операции, которые необходимо выполнить должны быть достаточно простыми; чтобы их в принципе можно было выполнить карандашом на бумаге за конечный отрезок времени.

Алгоритмы обладают множеством других свойств. Кратко рассмотрим некоторые из них.

Назовём проверку особых ситуаций алгоритма **особенностью** алгоритма (например, алгоритм вычисления тангенса проверяет угол на близость к 90 градусам, так как при этом тангенс не существует, но промежуточные точки отрезка существуют).

Алгоритм без особенностей всегда предпочтительней алгоритма с особенностями.

Свойство: **мощность** алгоритма. Использование алгоритма для целей, для решения которых он, вообще говоря, не предназначался можно назвать мощностью алгоритма.

Свойство: как **красота** алгоритма.

Алгоритмы бывают красивые и некрасивые.

Если у вас имеется дерево, на вершине которого висит плод, а вам необходимо достать этот плод, то вы можете сделать это, по крайней мере, тремя способами:
а) спилить дерево;
б) долго и нудно насыпать пирамиду из камней или земли, так чтобы потом, взобравшись по пирамиде, просто сорвать плод;
в) метким выстрелом перебить черенок плода.

Последний алгоритм доступа к плоду красивее двух предыдущих:
- он требует гораздо меньше ресурсов для достижения поставленной цели;
- цель достигается гораздо быстрее;
- достижение цели не разрушает окружение и через какое-то время можно будет снова получать плоды.

Если придётся выбирать из нескольких алгоритмов, то лучше выбрать из них наиболее красивый.

ИНФОРМАТИКА

Технологии программирования

Подпрограмма – программа, вызываемая другой программой или подпрограммой.

Подпрограмма не может быть запущена самостоятельно! Она должна быть вызвана другой подпрограммой или главной программой. При этом вызывающая программа обязана обеспечить среду функционирования подпрограммы.

Если подпрограмма могла быть вызвана повторно, до окончания её работы при предыдущем вызове или, если одна копия подпрограммы в памяти могла быть одновременно использована несколькими подпрограммами, то такая подпрограмма называлась **повторновходовой** или **реентерабельной**.

Если подпрограмма по ходу выполнения могла прямо или косвенно обратиться к самой себе, то такая программа называлась **рекурсивной**, а подобное обращение к подпрограмме рекурсивным вызовом.

ИНФОРМАТИКА

Технологии программирования

Для того, чтобы работать с подпрограммой необходимо знать как ей подать на вход переменные.

Использование подпрограмм экономит память, программа в целом становится короче, но работает такая программа чуть-чуть медленнее, так как тратится время на вызов подпрограммы.

С другой стороны, при разбиении программы на подпрограммы лучше просматривается структура больших и сложных программ.

Подпрограммы обеспечивают логическую сегментацию программы, облегчают отладку, снижают общее время отладки. Но самая главная ценность грамотно написанных подпрограмм в том, что их могут использовать другие программисты в других программах!

ИНФОРМАТИКА

Технологии программирования

Разбиение программы на подпрограммы иногда называют процедурной абстракцией.

Процесс разбиения программы на подпрограммы сильно напоминает процесс алгоритмизации. В некотором смысле это одно и тоже. Структура программы, будучи определённым образом отражена, похожа на структуру алгоритма.

Программы, выполняющиеся таким образом, называются последовательными программами, как программные единицы, выполняющие последовательность действий.

Последовательность действий заранее определена. Во всех до Windows – системах работают именно такие программы.

ИНФОРМАТИКА

Технологии программирования

Каким образом разбивать программу на составные части?

Здесь нет общих правил, хотя можно указать, что может повлиять на разбиение программы:

.Прежде всего, сама задача.

Задача может быть:

- исследовательской или технической;
- формализованной – не формализованной;
- эпизодической, требующей однократной реализации, или регулярно «считаемой»;
- простой или сложной;
- разрабатываемой для использования самим программистом или разрабатываемой на продажу и т.д.

ИНФОРМАТИКА

Технологии программирования

Кроме того, все задачи в программировании условно делятся на следующие категории: вычислительные, “асушные”, системные, управляющие и смешанные.

Вычислительные, обычно, характеризуются большим объёмом вычислений и незначительным объёмом ввода/вывода. Программисту специализирующемуся на вычислительных задачах (методах) просто необходимы знания математики в объёмах превышающих общеинженерные курсы математики.

В “асушных” задачах относительно незначительные объёмы простых вычислений, но со значительным объёмом ввода/вывода. Основные используемые технологии: работа с базами данных. В большинстве случаев, это преобразование одного множества таблиц в другое множество таблиц с тем или иным объёмом ввода данных и вывода. Задачи, почти всегда простые, но критичные ко времени выполнения.

ИНФОРМАТИКА

Технологии программирования

Системные: реализация модулей операционной системы, трансляторов, программ СУБД (системы управления базами данных), программы взаимодействия в сети и т.д. Как правило, требуются серьёзные знания вычислительной техники и особенностей её функционирования, а также операционных сред.

Управляющие: программы управляющие работой технических устройств или технологией протекания технических процессов. Почти всегда работают в режиме реального времени, чаще всего реализуются на Assemblere (в последнее время используется и Си++). Высокие требования по надёжности функционирования.

Смешанные: задачи, обладающие характеристиками нескольких ранее выделенных групп, например – реализация серьёзной графической среды требует значительных объёмов непростых вычислений и значительного объёма операций по вводу/выводу или реализация операционной системы реального времени.

Далее может повлиять на разбиение программы:

2. Программист. Программисты бывают разные и соответственно они по-разному сделают выбор алгоритма и разбиение задачи. Очень важно учитывать тип задачи и склонность программиста к решению именно этого типа задач.

3. Объём данных и соответственно выбираемый алгоритм.

4. Техническая и программная среда решения.

Структура программы будет существенно разной в последовательной системе или в системе управляемой событиями (Unix или Windows).

Вы можете работать в многопроцессорной среде с параллельными вычислениями или в однопроцессорной системе; в сети, используемой только для передачи данных или в существенно распределённой среде. Выбранный алгоритм может допускать параллельную обработку или нет.

5. Условиями конфиденциальности.

ИНФОРМАТИКА

Технологии программирования

Данные.

Любую программу можно рассмотреть, как некий набор операций, который применяется к некоторым данным в некоторой последовательности.

Во время выполнения программы существуют, условно, две группы данных:

- данные, определяемые программистом;
- данные, определяемые системой.

Данные, определяемые программистом, состоят из элементов, которые программист явно определяет в своей программе и которыми он манипулирует сам – числа, массивы, файлы и т.п.

Данные, определяемые системой, формируются средой функционирования программы и зависят, как от языка реализации, так и от операционной системы и техники, на которой функционирует программа. Например, ЭВМ может иметь аппаратную реализацию виртуальных адресов, а может и не иметь, что, конечно, влияет на наличие служебных данных.

ИНФОРМАТИКА

Технологии программирования

ООП – объектно – ориентированное программирование.

Процедурное программирование или алгоритмизация по сути своей представляет собой выделение в задаче действий и сосредотачивает внимание программиста на действиях.

Как ранее указывалось, происходит некоторое абстрагирование от данных. Однако восприятие окружающего мира мы делаем несколько не так, мы, скорее, сначала воспринимаем данные, а потом уже действия: если лес, то сначала какой лес и где, а уж потом, что там можно делать или что можно делать с этим лесом. Принципиально, мы воспринимаем мир объектами, а затем манипуляциями с этими объектами. При этом сам объект естественным образом воспринимается как чёрный ящик, мы редко задумываемся о действительном устройстве воспринимаемых объектов.

Эта технология восприятия, рано или поздно должна была найти отражение в технологии программирования.

С другой стороны, чем фактически занимается программист - постоянно изобретает собственные типы данных. Число машинных типов данных изумительно мало: int, float и char.

Уже тип string является агрегатом данных. При внимательном изучении мы должны будем заметить, что int, float и char всего лишь разные способы интерпретации одного и того же состояния слова. Отметим ещё раз, что сами данные не изменились: поменялись действия по получению конечного результата! Другими словами, по своей сути на нижнем уровне данные и действия не отделимы друг от друга.

Мы всегда изучаем данные и операции, выполняемые над данными, при этом всё это рассматривается без отрыва друг от друга, что очень естественно.

ИНФОРМАТИКА

Технологии программирования

Основные идеи ООП: инкапсуляция, полиморфизм, наследование.

Инкапсуляция (Encapsulation) – механизм, объединяющий данные и код, работающий с этими данными, в единое целое, который позволяет манипулировать и данными и кодом как единым целым и позволяющим защищать и данные и код от внешнего вмешательства. Когда коды и данные объединяются вместе, создаётся некоторая новая общность, которую мы называем **объект.** Но тогда в языке необходимы средства, позволяющие объявлять объекты и манипулировать им. Кроме того, надо чтобы мы могли объявлять переменные типа «объект»

Полиморфизм: (Polymorphism) – свойство, позволяющее одно и то же имя использовать для решения нескольких задач. Перегрузка функций один из примеров полиморфизма. На самом деле понятие полиморфизма гораздо шире вышеприведённого определения. Полиморфизм, следует понимать как применение одних и тех же действий/операций к разным объектам. В реальном мире большинство операций полиморфны. Примером типичной полиморфной операций может служить почти любое действие: переместить, например, можно стол, книгу, столицу и мн. др.

Другими словами: полиморфизм такое определение операций и функций, которое не зависит от типа данных.

Демонстрацией реально сложившейся ситуации может служить следующее заявление Страуструпа [8]: «настоящей объектно-ориентированной программой может называться только та, при трансляции которой транслятор не в состоянии определить типы объектов: реальные типы объектов появляются только при конкретном выполнении программы».

Наследование (Inheriance) – средство, посредством которого один объект может использовать свойства другого. Более точно, именно использование буквального значения слова наследовать. То есть, каким образом свойства одного объекта могут рассматриваться другим объектом как свои собственные. Таким образом, объект может иметь свои собственные свойства и наследуемые свойства. Такая система позволяет строить иерархии объектов: живое существо - животное - млекопитающее - хищное - кошачьи - тигры - уссурийский тигр.

Именно идея наследования упрощает разработку и отладку сложной иерархии объектов. Более того, в большинстве визуальных сред можно явно просмотреть, что от чего наследует. Самое интересное, что наследуемое свойство или операцию можно переопределить конкретно для каждого объекта.

Итак: инкапсуляция, полиморфизм и наследование - три кита, на которых держится объектно-ориентированное программирование (ООП). Но каждый принцип сам по себе в отдельности не создаёт тех преимуществ, которые даёт нам ООП. Видно, что некоторые принципы (полиморфизм) являются в основном преодолением возведённых в принцип некоторых проектных решений. Однако, применяемые все три вместе эти концепции действительно дают нечто новое, то самое, что и называют ООП.