Белорусский государственный университет информатики и радиоэлектроники

Основы конструирования программ

Преподаватель: **Меженная Марина Михайловна**

к.т.н., доцент,
доцент кафедры инженерной психологии и эргономики
а 609-2
mezhennaya@bsuir.by

Лекция 3: Технологии разработки программ

План лекции:

1.История развития представлений о разработке программ: от спагетти-кода к методологии структурного программирования.

2.Особенности методологии структурного программирования.

3.От структурного программирования к объектно-ориентированному.

4.Обобщение: особенности процесса проектирования в целом.

Неструктурные программы

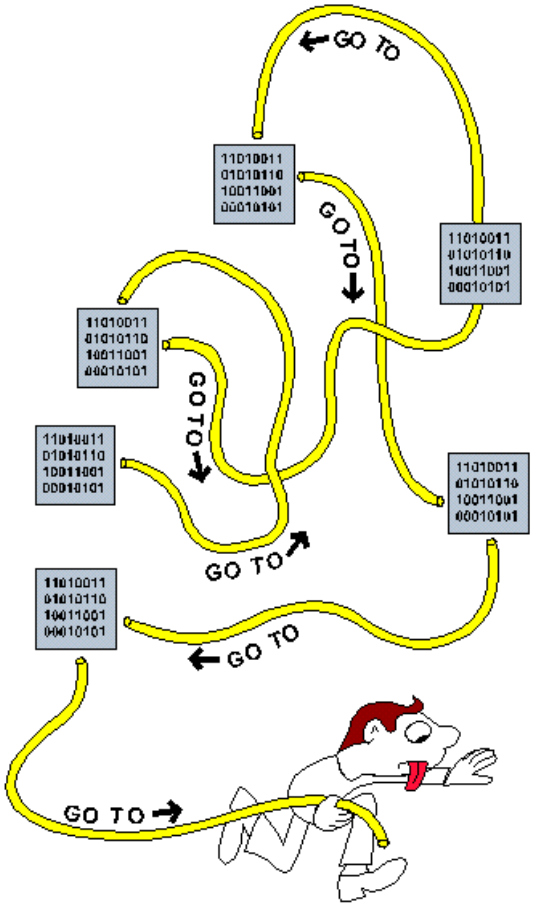
Изначально программы создавались так, чтобы задействовать минимум основной памяти и решить требуемую задачу за кратчайшее время.

Особенностью таких программ являлось множественное использование оператора безусловного перехода **goto** (аналог – **jump** в языках ассемблера).

Неструктурные программы

Подобные программы – слабо структурированные и трудные для понимания – получили название **спагетти-код**: ход выполнения такой программы похож на миску спагетти, то есть извилистый и запутанный.

Спагетти-код – самый распространённый антипаттерн программирования.



Пояснение: Оператор безусловного перехода goto

Оператор goto осуществляет переход к определённой точке программы, обозначенной меткой. Далее исполняются операторы программы, идущие в тексте непосредственно после метки.

```
goto label;

// Блок кода

label:

// Блок кода
```

Пояснение: Оператор безусловного перехода goto

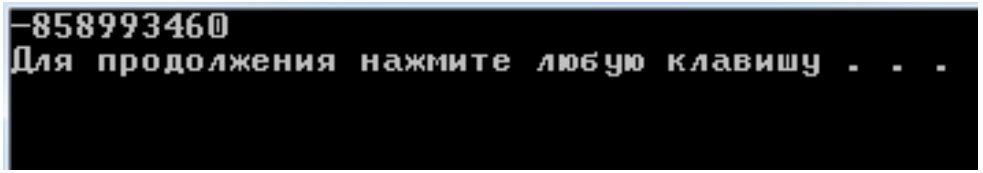
Некоторые способы применения goto могут создавать проблемы с логикой исполнения программы:

Переход в точку программы, расположенную после инициализации какой-либо переменной, приведёт к тому, что для этой переменной будет использовано некоторое случайное значение, которое находилось в памяти.

```
goto label;
// Блок кода

int number = 25;

label: int temp = number;
cout << temp << endl;
```



Пояснение: Оператор безусловного перехода goto

Некоторые способы применения goto могут создавать проблемы с логикой исполнения программы:

Передача управления внутрь тела цикла (процедуры или функции) приводит к пропуску кода инициализации цикла (выделение памяти под локальные переменные) и первоначальной проверки условия.

```
goto label;

for(int j=0; j<10; j++)
{
  label: cout << "j="<<j<<endl;
}
```

```
j=-858963082
j=-858963081
j=-858963080
j=-858963079
j=-858963078
j=-858963077
j=-858963076
j=-858963075
j=-858963074
j=-858963073
j=-858963072
j=-858963071
j=-858963070
j=-858963069
j=-858963068
j=-8589
```

Пояснение: Оператор безусловного перехода goto

Оператор goto в языках высокого уровня является объектом критики, поскольку чрезмерное его применение приводит к созданию нечитаемого «спагетти-кода».

Формально доказано (теорема Бёма — Якопини), что **применение goto не является обязательным**, то есть не существует такой программы с goto, которую нельзя было бы переписать без него с полным сохранением функциональности (однако, возможно, с потерей производительности).

Пояснение: Оператор безусловного перехода goto

В практическом программировании применение goto считается допустимым в одном случае – для выхода из нескольких вложенных циклов сразу.

```
int matrix[n][m];
int value;
// Блок кода

for(int i=0; i<n; i++) {
    for (int j=0; j<m; j++) {
        if (matrix[i][j] == value) {
          goto stop;
        }
        // Блок кода
    }
}
stop:
// Блок кода
```

Пояснение: Оператор безусловного перехода goto

Альтернатива:
применение
вспомогательных
переменных-флагов и
условных операторов.

```
int matrix[n][m];
int value;
bool flag = false;
// Блок кода

for(int i=0; i<n; i++) {
  for (int j=0; j<m; j++) {
       if (matrix[i][j] == value) {
          flag = true;
          break;
       }
       // Блок кода
   }
   if (flag) break;
}
```

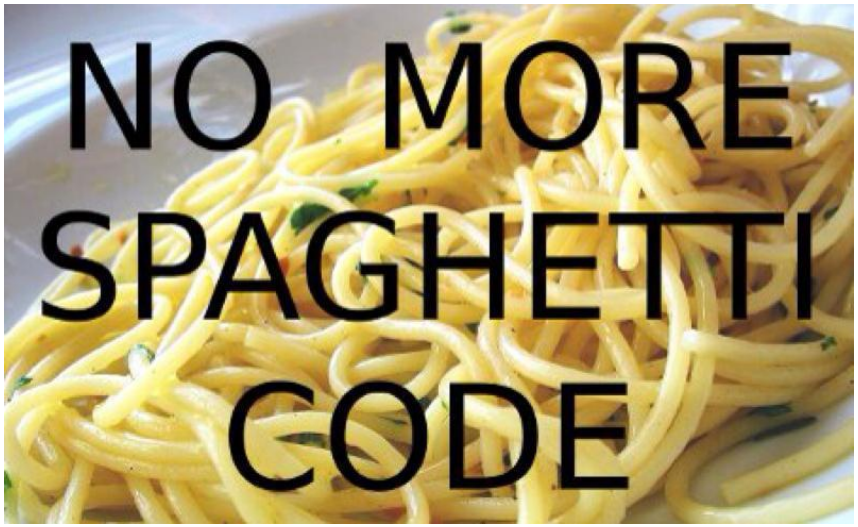
Пояснение: Оператор безусловного перехода goto

Спагетти-код может работать правильно и с высокой производительностью, но он крайне сложен в сопровождении и развитии. Доработка спагетти-кода для добавления новой функциональности иногда несет значительный потенциал внесения новых ошибок.

Методология структурного программирования

Исходя из проблем, связанных с программами спагетти-кода, программисты 70-х годов Э.Дейкстра и Н.Вирт разработали строгие правила разработки программ, которые получили название

Методология структурного программирования



Принципы структурного программирования

1.Программа (алгоритм) должна разделяться на независимые части, называемые **модулями**. Модуль – это последовательность логически связанных операций, оформленных как отдельная часть программы.

2.Модуль имеет **одну входную** и **одну выходную** точку (в отличие от программ с множественным использованием оператора goto).

3.В модуле используются только три базовые управляющие конструкции (**следование, ветвление, цикл**).
Теорема Бёма — Якопини (1965 год), или «Теорема о структурном программировании»: Любая программа, заданная в виде блок-схемы, может быть представлена с помощью трех управляющих структур: последовательность, ветвление, цикл.

Принципы структурного программирования

4.В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом. Никаких других средств управления последовательностью выполнения операций не предусматривается (т.е. **от использования оператора безусловного перехода goto следует отказаться**).

5.Повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). Таким же образом (в виде подпрограмм) можно оформить логически целостные фрагменты программы, **даже если они не повторяются**.

6.Разработка программы ведётся пошагово, методом «сверху вниз» (**нисходящее проектирование**).

Методология структурного программирования

Использование модулей имеет следующие преимущества:

- возможность создания программы несколькими программистами;
- простота проектирования и последующей модификаций программы;
- возможность использования готовых библиотек наиболее употребительных модулей;

Достоинства структурного программирования

- повышается надежность программ (благодаря хорошему структурированию при проектировании программа легко поддается тестированию и не создает проблем при отладке);
- уменьшается время и стоимость программной разработки;
- улучшается читабельность программ.

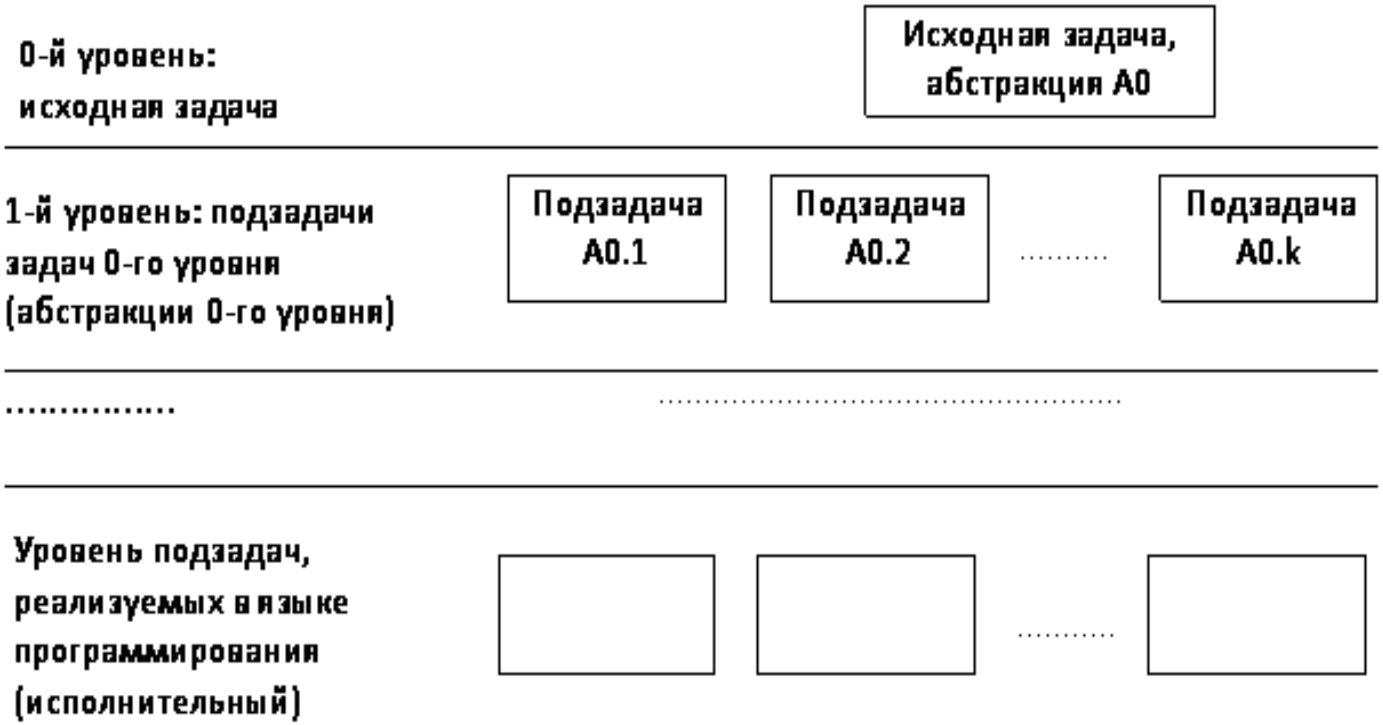
Нисходящее проектирование

В рамках методологии структурного программирования была разработана специальная технология – **нисходящее проектирование**, которая состоит в **пошаговой детализации (декомпозиции, разложении)** задачи на подзадачи.

Пошаговая детализация представляет собой процесс дробления задачи на подзадачи, установления логических связей между ними.

После этого переходят к уточнению выделенных подзадач.
Этот процесс детализации продолжается до уровня, позволяющего достаточно легко реализовать подзадачу на выбранном языке программирования.

Нисходящее проектирование для выделения уровней проекта



Общая схема нисходящей разработки

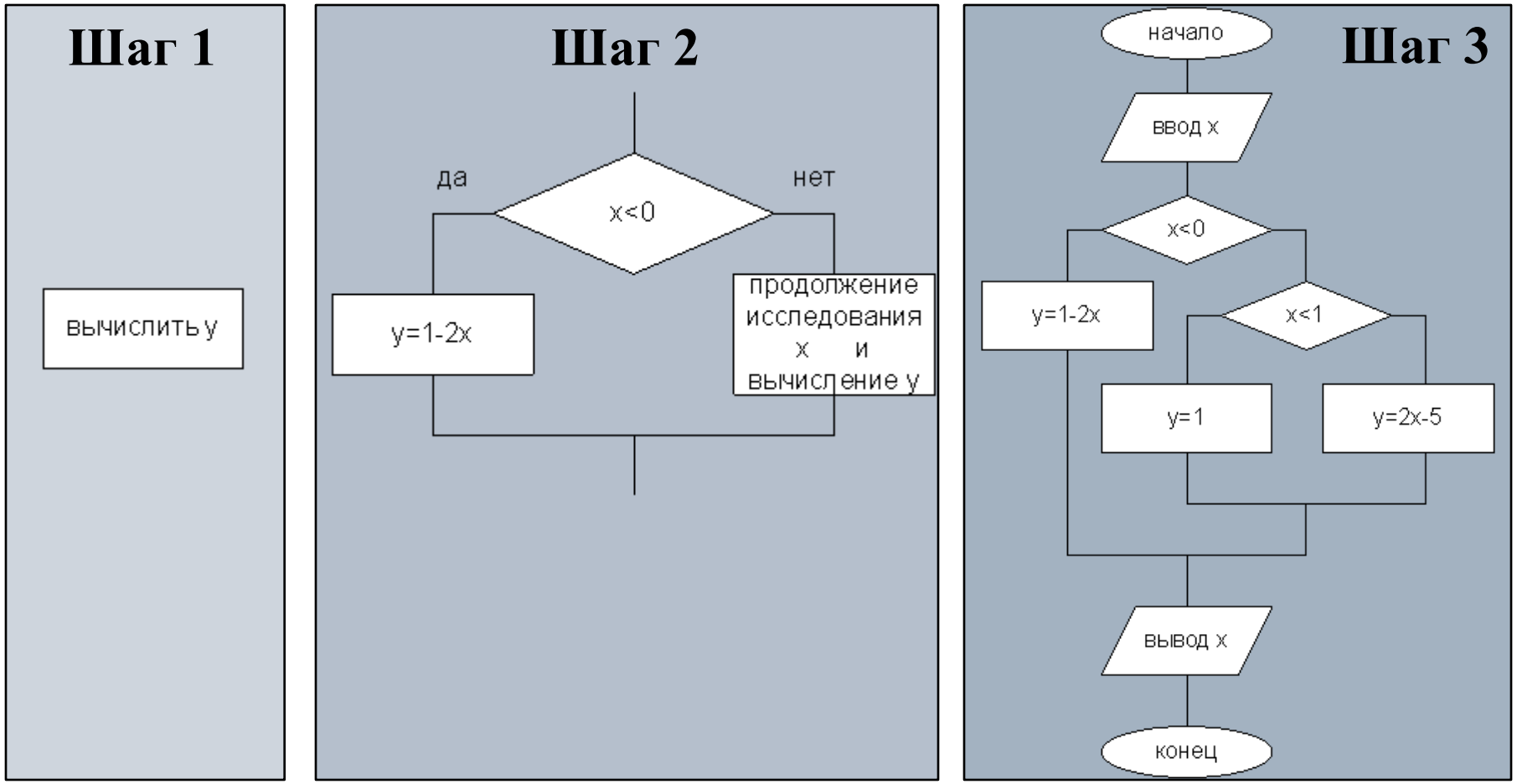
Нисходящее проектирование при разработке частных решений

Для разработки алгоритма конкретной задачи также можно использовать нисходящее проектирование:

при этом пошаговая детализация строится на **принципе замещения**, который состоит в замене любого функционального блока (прямоугольника) блок-схемы некоторой базовой структурой (следование, ветвление, повторение).

Пример. Построить структурированную блок-схему алгоритма для вычисления функции:

$$y=\begin{cases} 1-2x, & \text{если } x<0; \\ 1, & \text{если } 0<=x<1; \\ 2x-5, & \text{если } x>=1. \end{cases}$$



Разработка и отладка программы, созданной в соответствии с принципами структурного программирования

Можно разработать тест основной программы таким образом, чтобы вместо каждого связного логического фрагмента вставлялся вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих подпрограмм, в программу вставляются фиктивные части — **заглушки**, которые, говоря упрощенно, ничего не делают.

Затем заглушки заменяются или дорабатываются до настоящих полнофункциональных фрагментов (модулей).

Разработка заканчивается тогда, когда не останется ни одной заглушки. Полученная программа проверяется и отлаживается.

Разработка алгоритма: восходящее проектирование

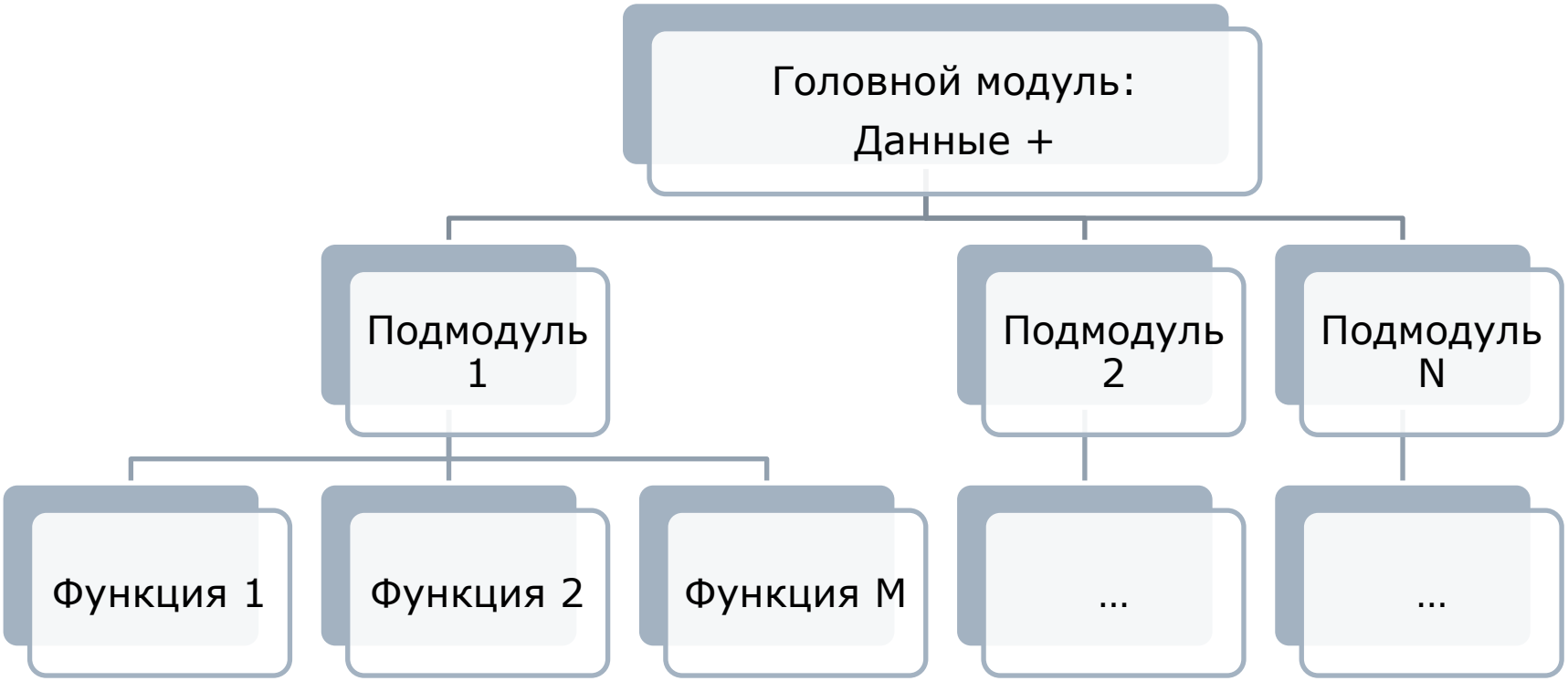
Восходящее проектирование – методика разработки программ, при которой крупные блоки собираются из ранее созданных мелких блоков.

Случаи восходящего проектирования:

1.Хорошо известны решения на нижних уровнях.

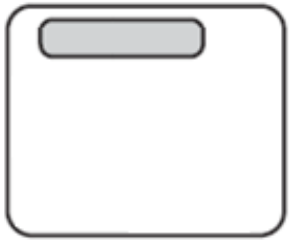
2.Разработка не поддается детальному планированию, она ведется методом проб и ошибок.

Структура программы в процедурном программировании

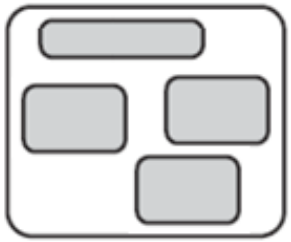


Этапы проектирования в процедурном программировании

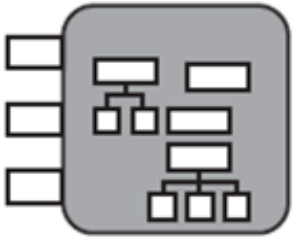
1 Программная система



2 Выбор способов описания входных данных



3 Разделение системы на модули/файлы



4 Выделение функций в рамках каждого модуля

5 Проектирование функций

От структурной методологии к объектно-ориентированной

Недостатки процедурного программирования:

- сложность создания больших программ;

- типы данных и функции для их обработки не объединены в структуре программы, поэтому, отсутствует механизм управления доступом к этим данным и функциям; любая функция может обработать эти данные;

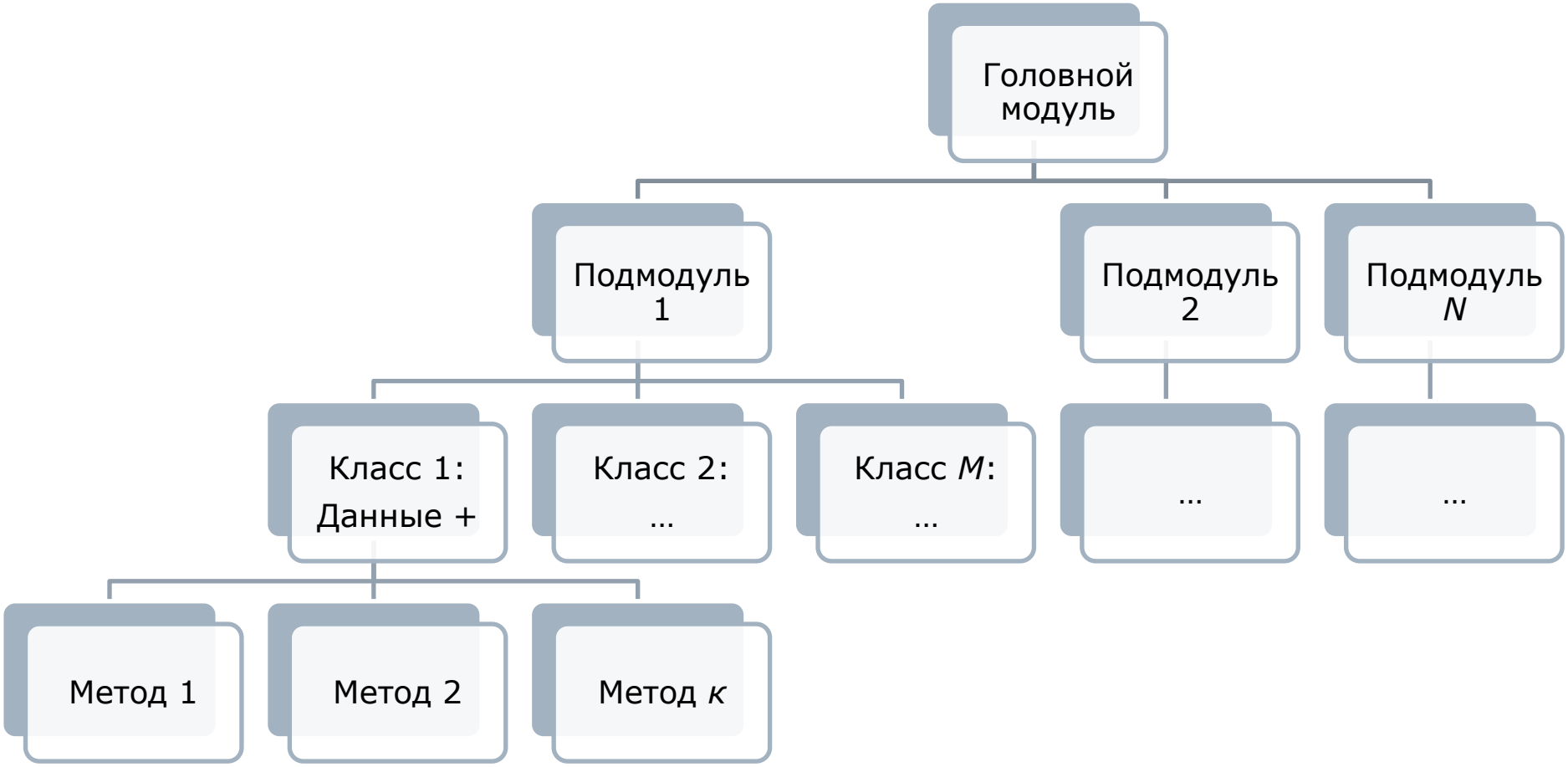
- для множества типов данных, определяемых пользователем, необходимо программировать множество комплексов похожих функций для обработки этих типов данных;

От структурной методологии к объектно-ориентированной

Процедурное программирование не обеспечивает в достаточной степени абстракции данных. Потребовался основной принцип абстрактного типа данных: никакие другие функции, кроме специальных не должны иметь доступа к этим данным. Для реализации абстрактного типа данных был предложен новый тип - класс, как совокупность данных и функций.

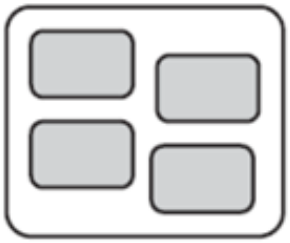
Методология программирования 90-х годов – объектно-ориентированное программирование; основывается на построении программы как иерархии классов.

Структура программы в объектно-ориентированном программировании

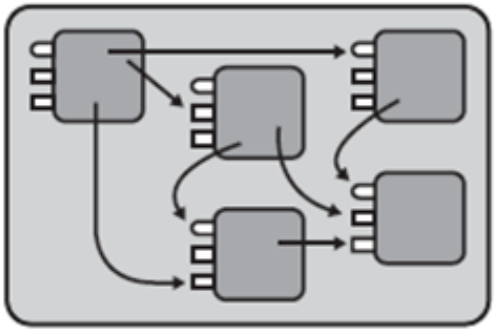


Этапы проектирования в объектно-ориентированном программировании

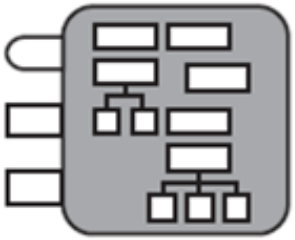
1 Программная система



2 Разделение системы на подсистемы/пакеты/слои



3 Разделение подсистемы на классы



4 Разделение классов на данные и методы

5 Проектирование методов

Особенности процесса проектирования программ

Проектирование — недетерминированный процесс

Если вы попросите трех человек спроектировать одну и ту же программу, они вполне могут разработать три совершенно разных, но вполне приемлемых проекта. Как правило, спроектировать компьютерную программу можно десятками разных способов.

Особенности процесса проектирования программ

Проектирование — эвристический процесс

Методы проектирования носят эвристический характер, т. е. представляют собой «практические советы», которые могут сработать, а могут и не сработать.
Универсальных методик проектирования не существует.
НО: существует наработанный практический опыт в виде **шаблонов проектирования.**

Особенности процесса проектирования программ

Проектирование — постепенный, итеративный процесс

Проекты приложений не возникают в умах разработчиков сразу в готовом виде. Они развиваются и улучшаются в ходе обзоров, неформальных обсуждений, написания кода и выполнения его ревизий.

К чему нужно стремиться при проектировании *или* характеристики качественного проекта

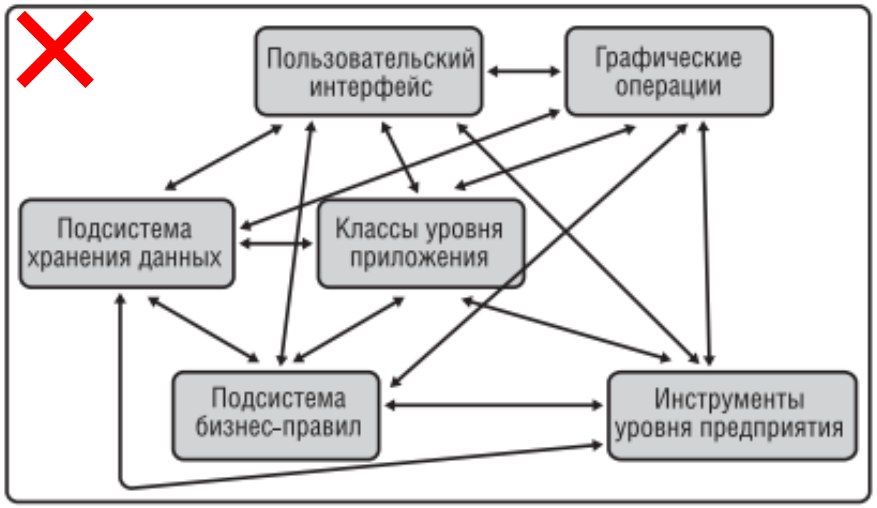
Минимальная сложность. Главная цель проектирования – управление сложностью.

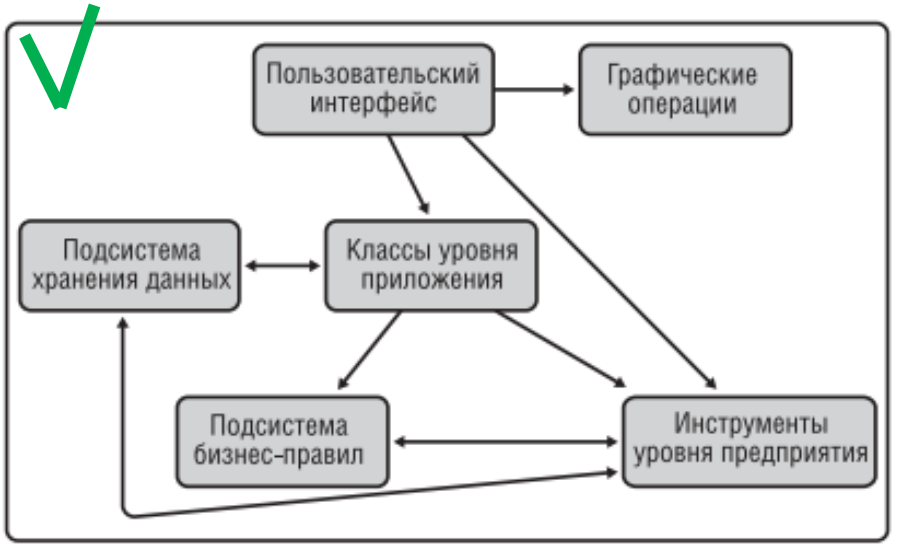
Простота сопровождения. Проектируя приложение, не забывайте о программистах, которые будут его сопровождать.

Слабое сопряжение предполагает сведение к минимуму числа соединений между разными частями программы. Это позволит максимально облегчить интеграцию, тестирование и сопровождение программы.

К чему нужно стремиться при проектировании *или* характеристики качественного проекта

Про слабое сопряжение:





К чему нужно стремиться при проектировании *или* характеристики качественного проекта

Расширяемость. Расширяемостью системы называют свойство, позволяющее улучшать систему, не нарушая ее основной структуры. Изменение одного фрагмента системы не должно влиять на ее другие фрагменты. Внесение наиболее вероятных изменений должно требовать наименьших усилий.

Возможность повторного использования. Проектируйте систему так, чтобы ее фрагменты можно было повторно использовать в других системах.

К чему нужно стремиться при проектировании *или* характеристики качественного проекта

Сокрытие информации — один из основных принципов и структурного, и объектно-ориентированного проектирования. В первом случае сокрытие информации лежит в основе идеи «черных ящиков».
Во втором оно дает начало концепциям инкапсуляции, модульности, абстракции.

Обобщение подходов к проектированию

1 Используйте итерацию (делайте попытки проектирования снова и снова, пока не получите «полную картину» дальнейших действий)

2 Реализуйте принцип «Разделяй и властвуй» (разделите программу на части и проектируйте каждую часть отдельно; возникли проблемы? – пересмотрите свое решение или… см. пункт 1)

3 Берите на вооружение нисходящий и восходящий подходы (декомпозиция vs композиция)

4 Рисуйте

5 Используйте шаблоны проектирования

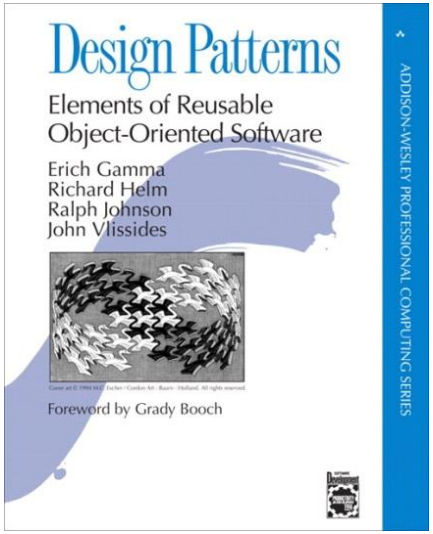
Пояснение: Паттерны проектирования

Паттернами (шаблонами) проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения.

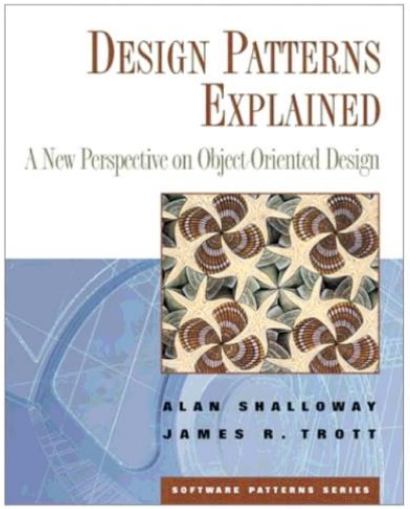
Паттерны проектирования не являются готовыми решениями, которые можно трансформировать непосредственно в код, а представляют общее описание решения проблемы, которое можно использовать в различных ситуациях.

Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем.

Паттерны проектирования



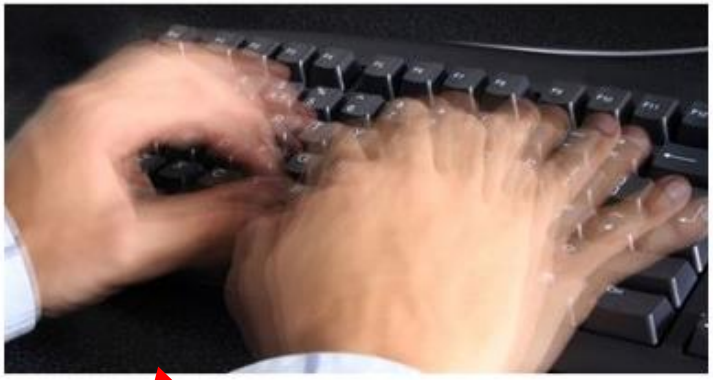
Gamma, Erich, Johnson, Vlissideset.
Design Patterns.
Книга о шаблонах проектирования, написанная «Бандой четырех».



Shalloway, Trott.
Design Patterns Explained.

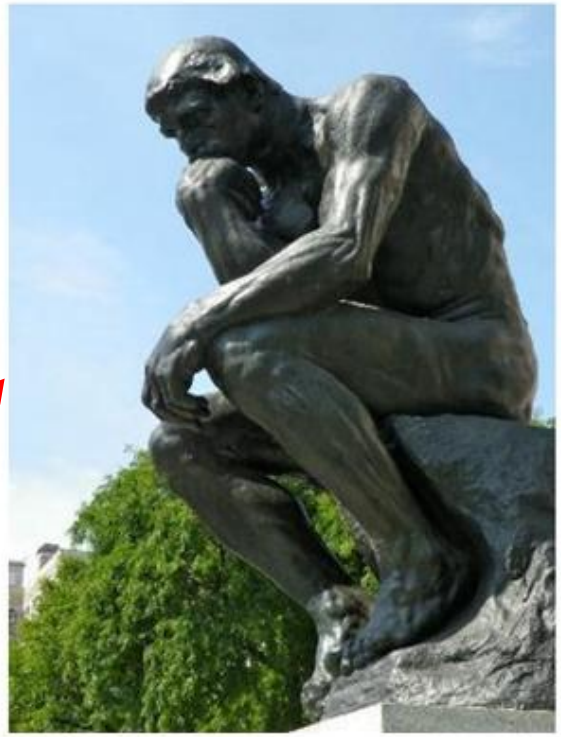
Что в результате?

Люди думают, что программирование выглядит вот так:



2 собственно кодирование

Но на самом деле оно выглядит вот так:



1 проектирование программы