Архитектура ПП

Это **совокупность базовых концепций его построения.**

Определяется:

сложностью решаемых задач;

степенью универсальности.

Различают следующие архитектуры:

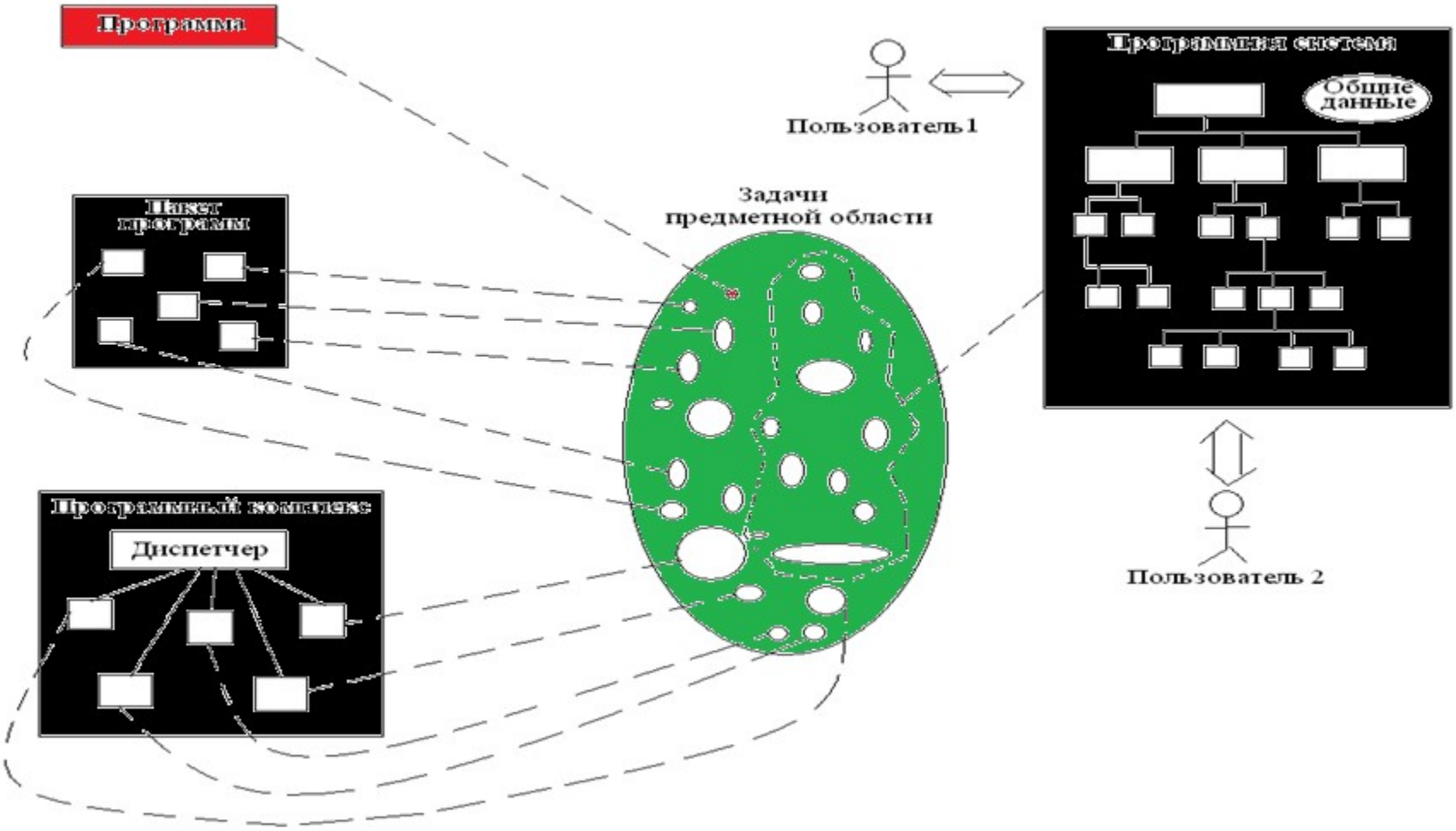
Программа

Пакеты программ

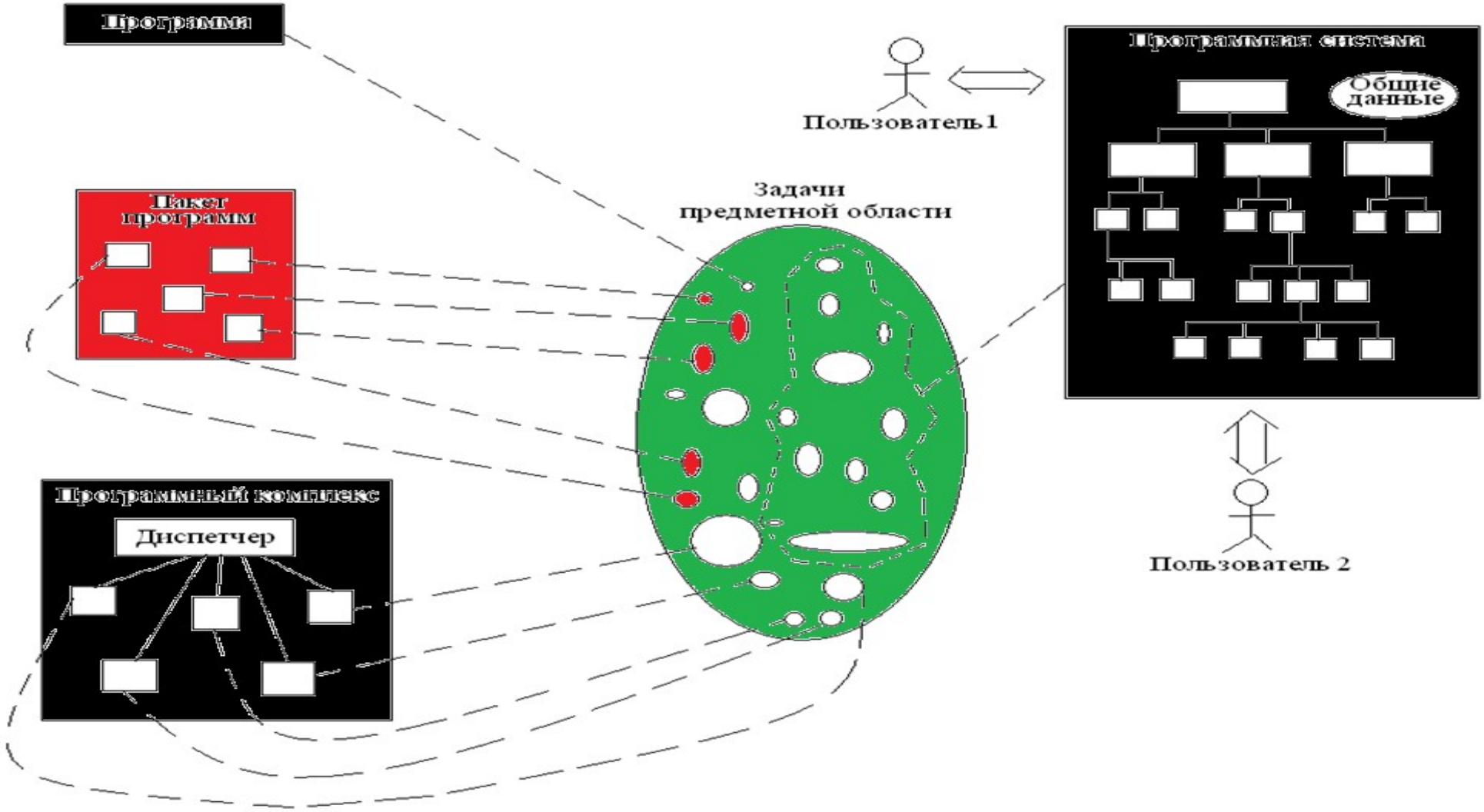
Программные комплексы

Программные системы

Программа - адресует компьютеру набор инструкций, точно описывающий последовательность действий *для решения конкретной задачи*.

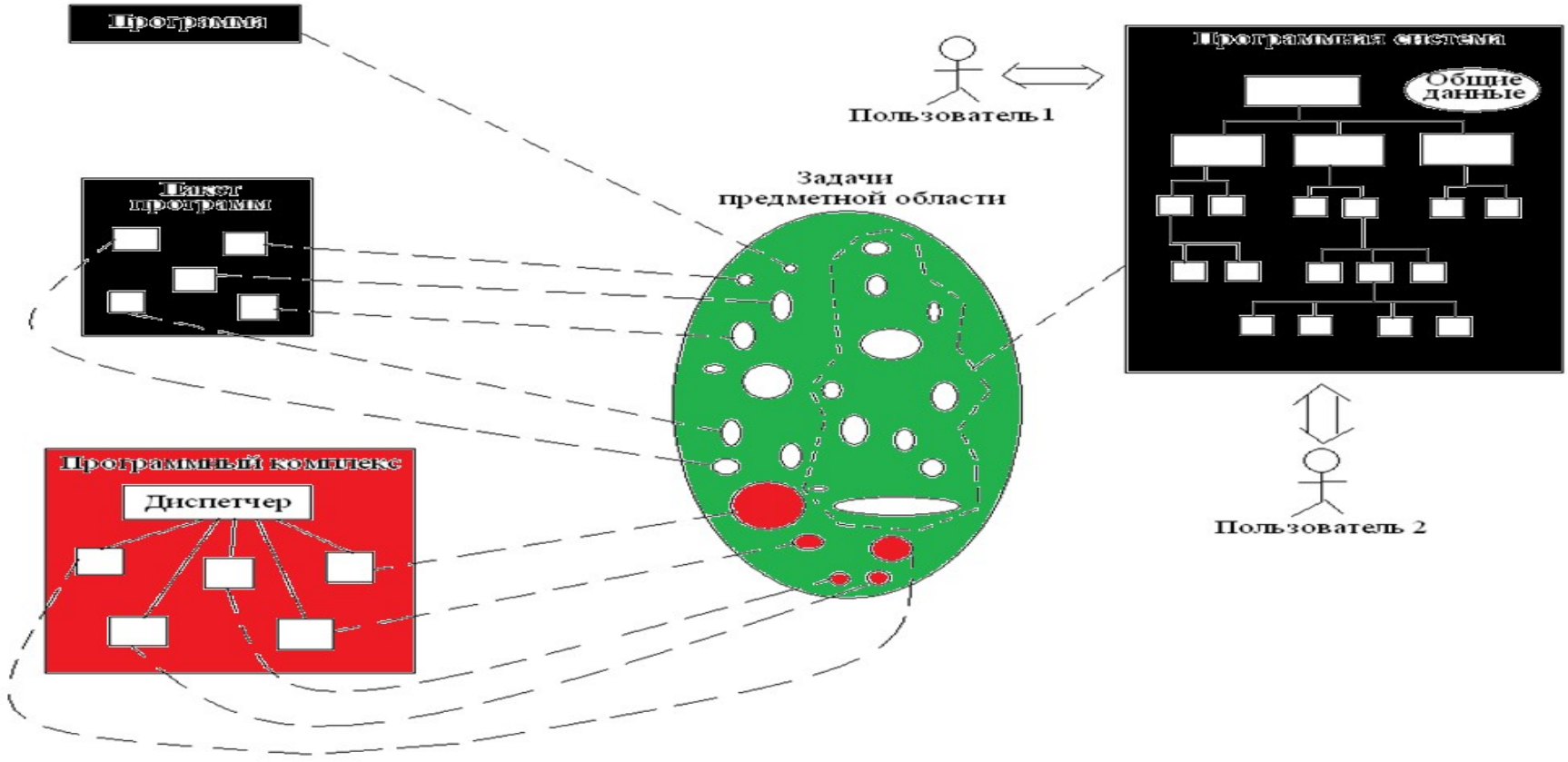


Пакеты программ - представляют собой *совокупность* программ, решающих *задачи некоторой прикладной области*.



Программные комплексы - представляют собой совокупность программ, *совместно* обеспечивающих решение *небольшого класса сложных задач одной прикладной области.*

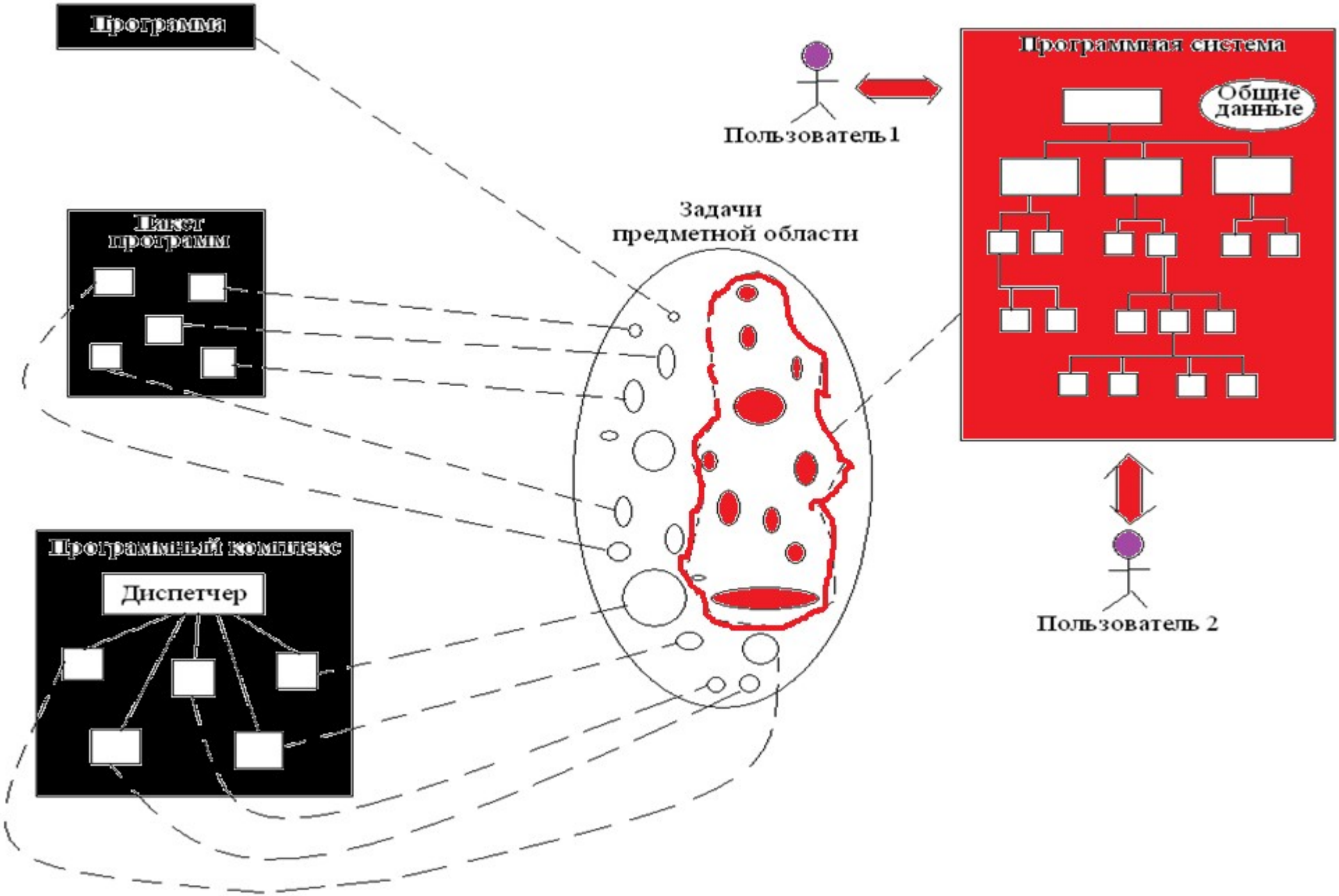
(Вызов программ в программном комплексе осуществляется специальной программой – диспетчером, который обеспечивает несложный интерфейс с пользователем.)



Программные системы - представляют собой ***организованную*** совокупность программ (*подсистем*), позволяющую решать *широкий класс задач из некоторой прикладной области*.

В отличие от программных комплексов:

- программы *взаимодействуют через общие данные*;
- имеют развитый пользовательский интерфейс;
- имеют развитый внутренний интерфейс.


Программа
Пакет программ
Программный комплекс
Диспетчер
Задачи предметной области
Пользователь 1
Программная система
Общие данные
Пользователь 2

В соответствии с ГОСТ 19.101-90 программы подразделяются на следующие виды:

- **Компонент.** Это программа, рассматриваемая как единое целое и выполняющая законченную функцию. Может применяться самостоятельно или в составе комплекса.

- **Комплекс.** Это программный продукт, состоящий из двух или более компонентов и (или) комплексов, которые выполняют взаимосвязанные функции. Может применяться самостоятельно или в составе другого комплекса.

Модульный подход

Модуль - *это отдельная часть программы, допускающая сепаратную компиляцию и сборку с другими блоками.*

В частности, под модулем понимают отдельно компилируемую библиотеку ресурсов.

С другой стороны модуль одновременно может быть и функциональной компонентой программы.

Преимущества использования модулей:

- Возможность одновременной работы нескольких человек;
- Возможность создания библиотек стандартных модулей;
- Возможность подзагрузки модулей в оперативную память;
- большое количество естественных точек для тестирования;
- Упрощают проектирование и модификацию программы.

Недостатки:

- Увеличение времени выполнения за счет организации межмодульных связей;
- Увеличение времени компиляции и загрузки;
- Код программы и тексты занимают больше памяти.

Чтобы определить насколько удачно была проведена разбивка системы на модули, можно оценить степень независимости модулей (как подпрограмм, так и библиотек).

Для этого используют критерии: сцепление и связность.

Сцепление

Характеризует качество отделения модулей.

Различают пять типов сцепления модулей:

- **по данным** (модули обмениваются данными, представленными скалярными значениями).

Например: Function Min(a, b: integer):integer;

- **по образцу** (модули обмениваются данными, объединенными в структуры).

Например:

Function Max(a:array of integer):integer;
где A – открытый массив);

- **по управлению** (Например, в качестве параметров передаются управляющие данные (Флаги), от которых зависит алгоритм работы модуля или режим);

- **по общей области данных** (не рекомендуется использовать, т.к. усложняется локализация ошибок, уменьшается гибкость пространства имен);

- **по содержимому** (Один модуль содержит обращения к внутренним компонентам другого).

  Языки высокого уровня такое сцепление не поддерживают, но языки низкого уровня, например, Ассемблер позволяют.

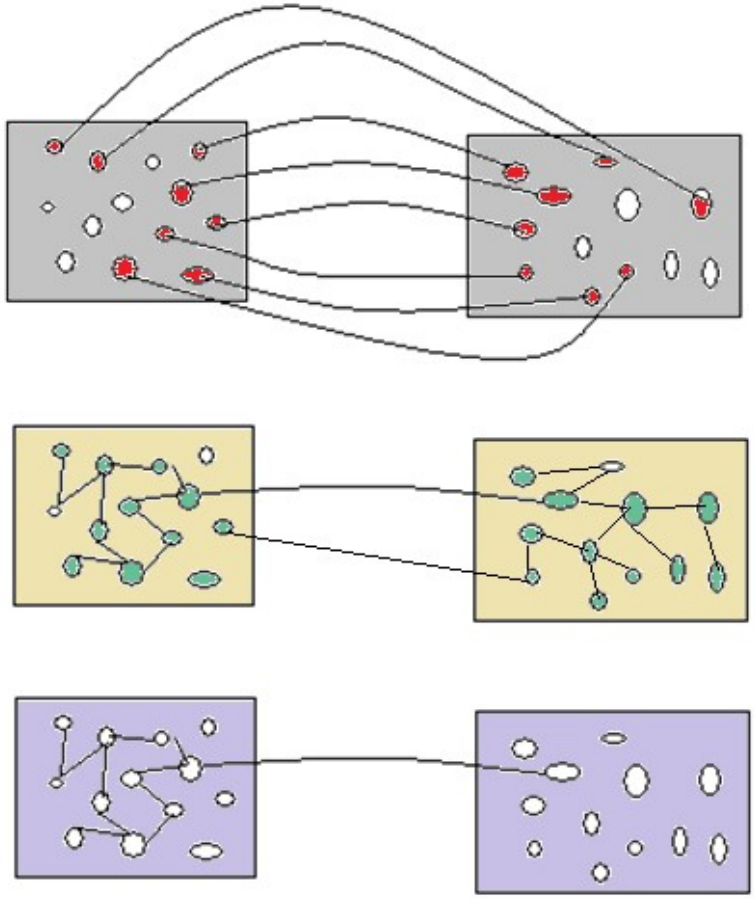
Связность

Характеризует степень взаимосвязи элементов, реализуемых одним модулем.

Размещение сильно связанных элементов в одном модуле уменьшает межмодульные связи.

И наоборот, размещение сильно связанных элементов в разные модули усиливает межмодульные связи и усложняет понимание их взаимодействия.

Объединение слабо связанных элементов также **уменьшает** технологичность модулей, так как такими элементами сложнее мысленно манипулировать.



Различают следующие **виды связности**

(в порядке убывания уровня):

- **функциональную** (компоненты модуля предназначены для выполнения одной функции);
- **последовательную** (выход одной функции служит исходными данными для другой функции);
- **информационную** (компоненты обрабатывают одни и те же данные);
- **процедурную** (функции или данные являются частями одного процесса);

- **временную** (функции выполняются параллельно или в течение некоторого периода времени, а данные используются в некотором временном интервале);
- **логическую** (модуль с логической связностью функций часто реализует альтернативные варианты одной операции);
- **случайную** (связь между элементами практически отсутствует).

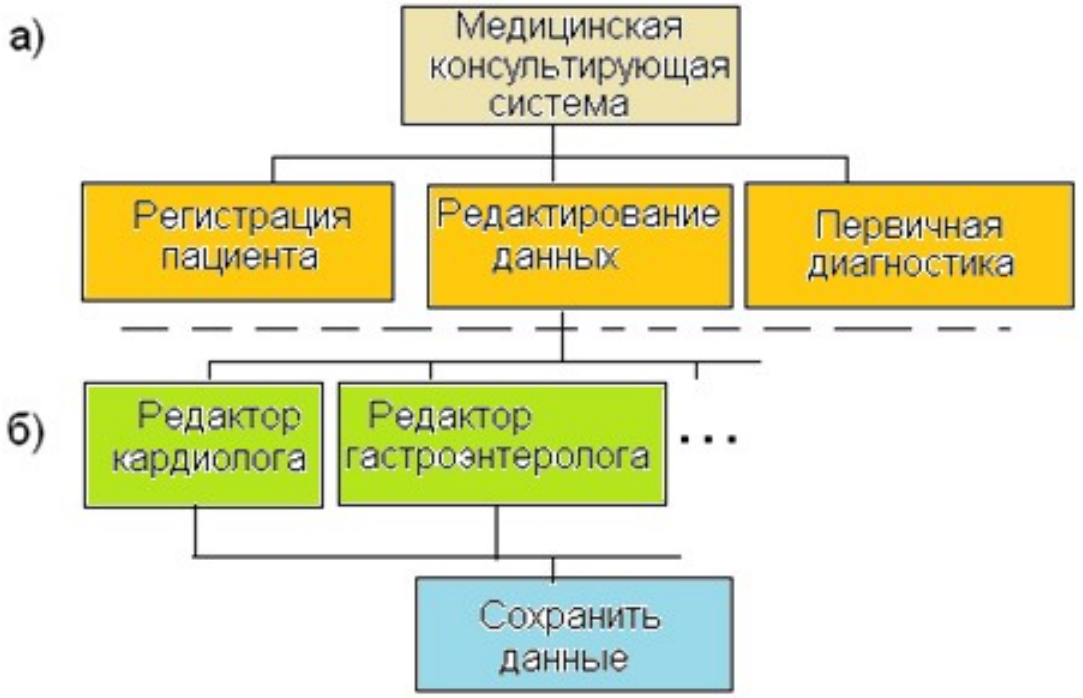
Как правило, **модули верхних уровней иерархии** имеют функциональную или последовательную связность функций и данных.

Принцип вертикального управления

При проектировании системы строится:

а) схема иерархии отдельных программ;

б) схема иерархии отдельных модулей
(проектирование следует начинать с вершины)



Передача управления происходит лишь по вертикальным линиям, соединяющих модули в схеме иерархии. Вертикальное управление происходит по следующим правилам

(Свойства модулей):

- Должен возвращать управление модулю, который его вызвал;
- Может вызывать другие модули уровнем ниже, но не может вызывать модуль своего уровня или выше (кроме случаев рекурсии);

- Должен иметь одну точку входа и одну точку выхода;
- Должен быть сравнительно невелик (50-60 операторов);
- Один модуль должен выполнять по возможности одну функцию определенного уровня;
- Модуль должен быть независим от истории вызовов

Преимущества вертикального управления:

- Логика программы более понятна;
- При чтении головного модуля проявляется логика всей программы;
- Программу проще изменять и дополнять;
- Позволяет быстро обнаружить логические ошибки.

Технологичность

Под ***технологичностью*** понимают **качество** проекта ПП, от которого зависят трудовые и материальные затраты на его реализацию и последующую модификацию.

Хороший проект сравнительно быстро и легко кодируется, тестируется, отлаживается и модифицируется.

Технологичность ПП определяется:

- ***Проработанностью модели***

Хорошая проработка означает, что:

- четко определены подзадачи;
- четко определены структуры данных.

Уменьшает количество ошибок,
из-за которых потребуется существенно изменять программу.

- ***Уровнем независимости модулей*.**

Практика показала, что чем выше независимость, тем:

- легче понять работу, как отдельного модуля, так и всей программы;
- легче наращивать функциональность программной системы;
- меньше вероятность появления «волнового» эффекта;
- проще организовать разработку группой программистов и легче его сопровождать.

***Стилем программирования*.**

Это набор правил, которым следует программист.

Хороший стиль предполагает:

- использование комментариев;
- использование со смыслом имен переменных, процедур и функций и др.;
- использование отступов и пустых строк.

От стиля зависят: количество ошибок при наборе текста; читаемость программы; процессы отладки и внесения изменений.

- ***Степенью повторного использования кодов.***

Повторное использование — **основная методология**, которая применяется **для** сокращения трудозатрат при разработке сложных систем.

Увеличение степени предполагает:

- использование ранее разработанных библиотек;
- унификацию кодов текущей разработки.