



Software Design Lecture Outline



- 1) Этап проектирования ПО
- 2) Типы архитектур ПО
- 3) Паттерны управления
- 4) Модульная декомпозиция
- 5) Документирование
(SDD)
- 6) Фундаментальные паттерны

Design, Coding, Testing



Design Process

Предварительное проектирование

- Структурирование системы
- Моделирование управления
- Модульная декомпозиция

Детальное проектирование

- Проектирование модулей
- Проектирование данных
- Проектирование процедур

Architecture Patterns

Database-centric Architecture

Client–Server Architecture

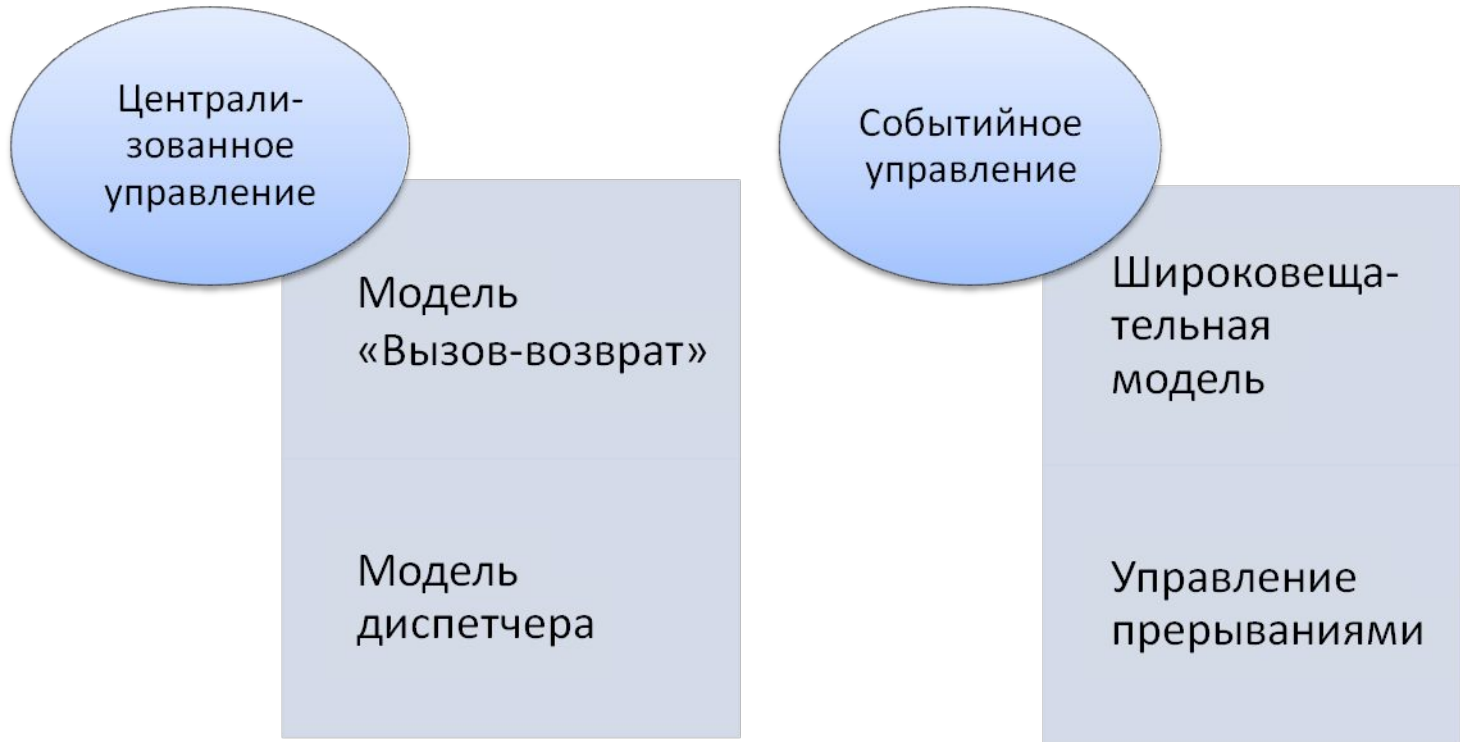
Three-Tier Architecture

Front-End and Back-End

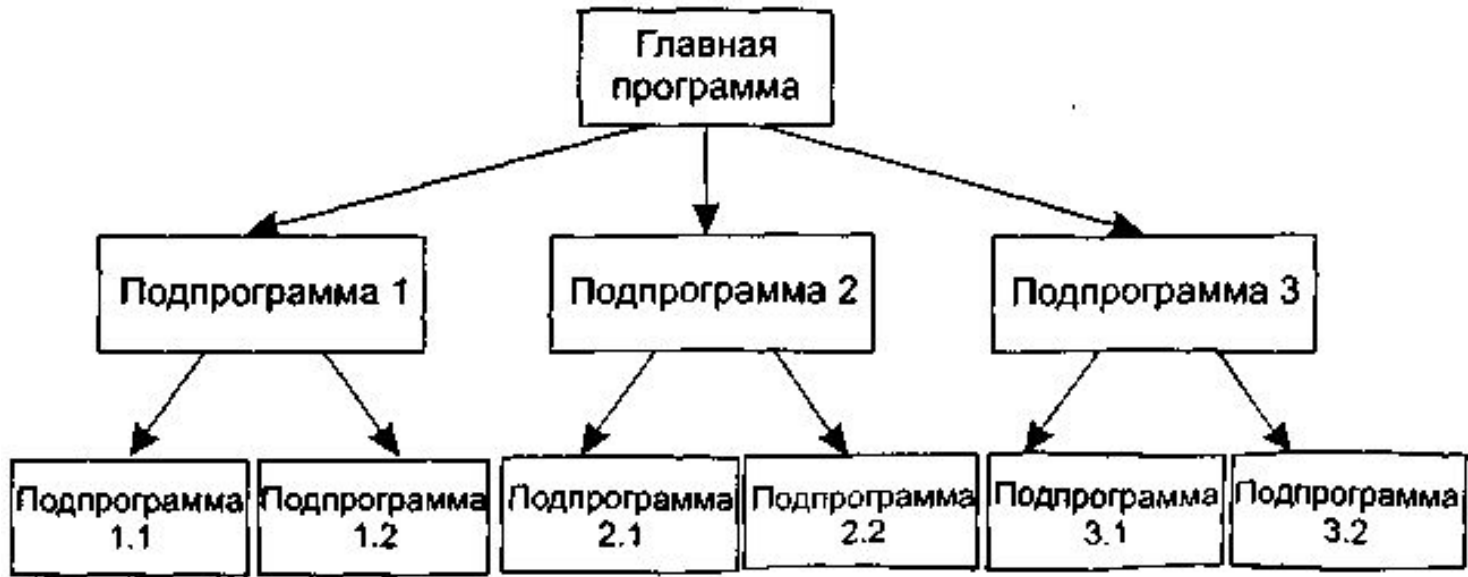
Web Application

Data Flow Architecture

Control Patterns

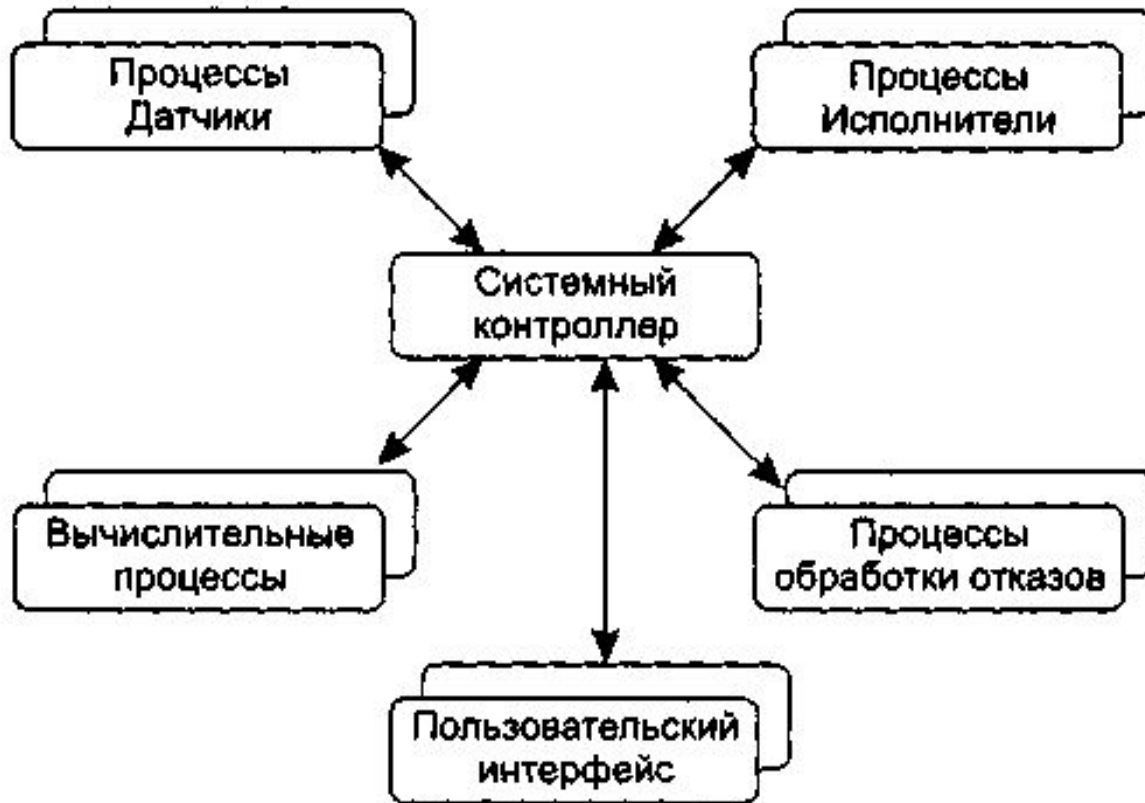


Control Patterns



Модель «Вызов-возврат»

Control Patterns



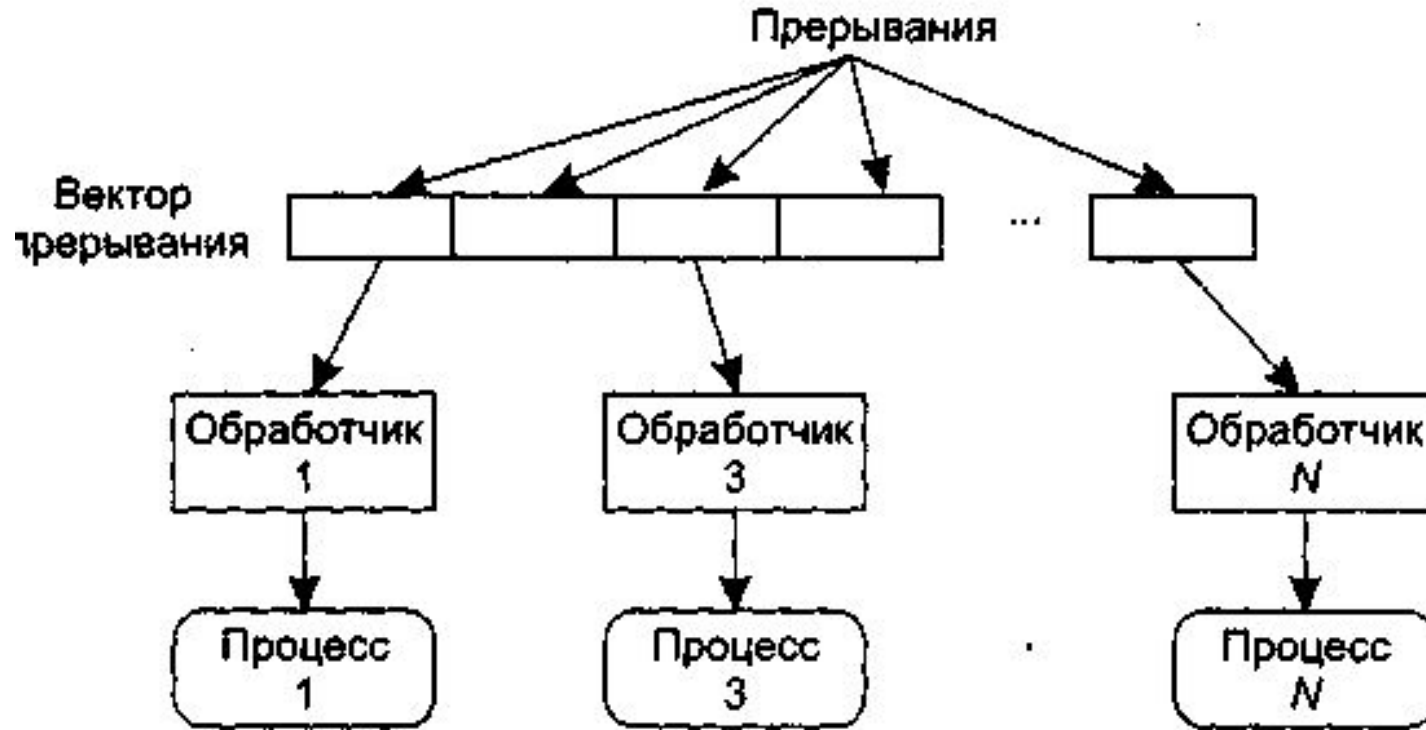
Модель диспетчера

Control Patterns



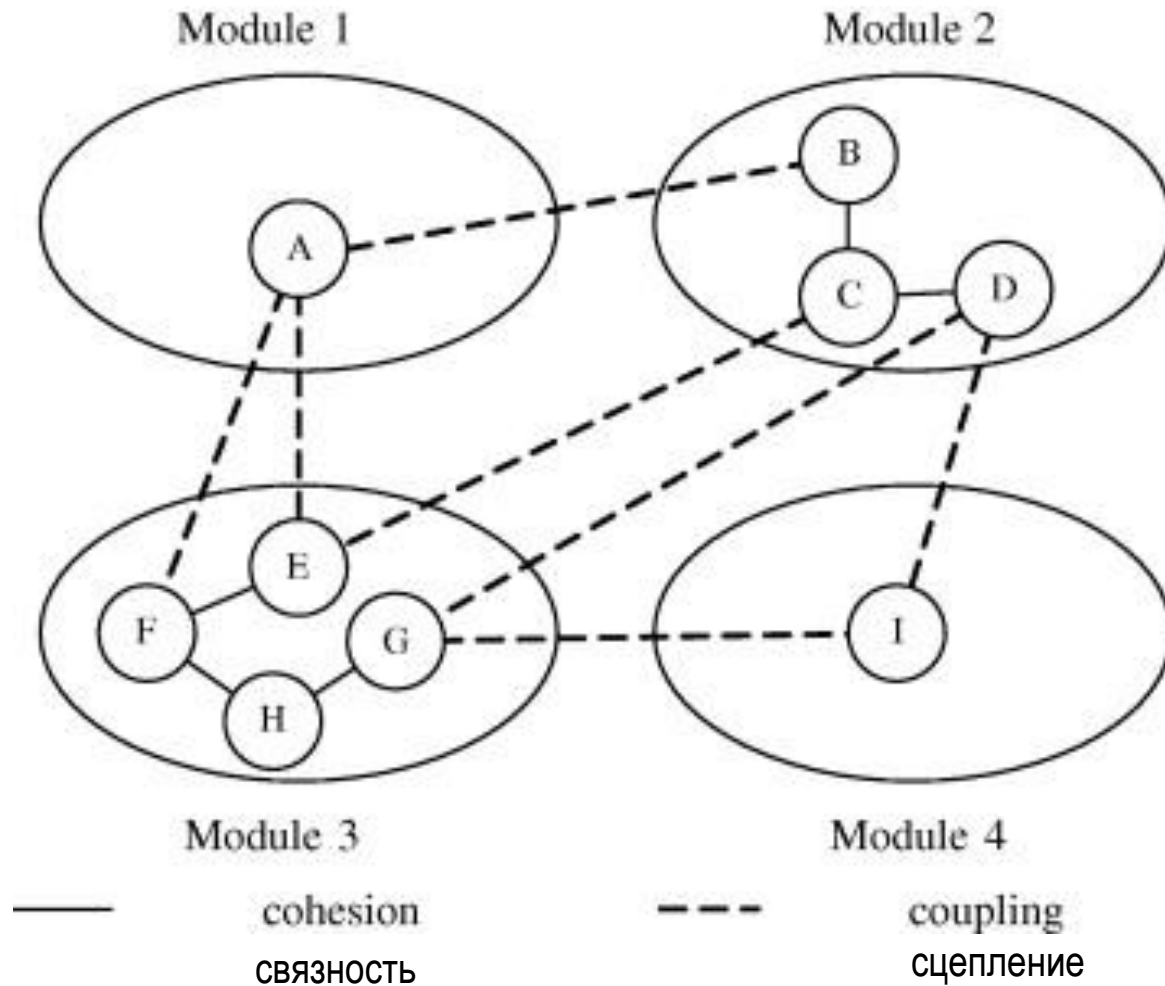
Широковещательная модель

Control Patterns



Прерывания

Module Decomposition



Module Cohesion

СВЯЗНОСТЬ	Cohesion	Сила СВЯЗНОСТИ
Функциональная	Functional	10
Последовательная	Sequential	9
Коммуникативная	Communicational	7
Процедурная	Procedural	5
Временная	Temporal	3
Логическая	Logical	1
По совпадению	Coincidental	0

Module Coupling

Сцепление	Coupling	Степень Сцепления
По данным	Data coupling	1
По образцу	Stamp coupling	3
По управлению	Control coupling	4
По внешним ссылкам	External coupling	5
По общей области	Common coupling	7
По содержанию	Content (Pathological) coupling	9

SDD (Software Design Document)

IEEE 1016.1-1993

1. Введение

- 1.1. Цель
- 1.2. Описание проекта
- 1.3. Определения, сокращения, термины

2. Ссылки

3. Описание декомпозиции

- 3.1. Модульная декомпозиция
 - 3.1.1. Описание модуля 1
 - 3.1.2. Описание модуля 2
- 3.2. Декомпозиция на параллельные процессы
 - 3.2.1. Описание процесса 1
 - 3.2.2. Описание процесса 2
- 3.3. Декомпозиция данных
 - 3.3.1. Описание блока данных 1
 - 3.3.2. Описание блока данных 2

4. Описание зависимостей

- 3.1. Межмодульные зависимости
- 3.2. Межпроцессные зависимости
- 3.3. Зависимости внутри данных

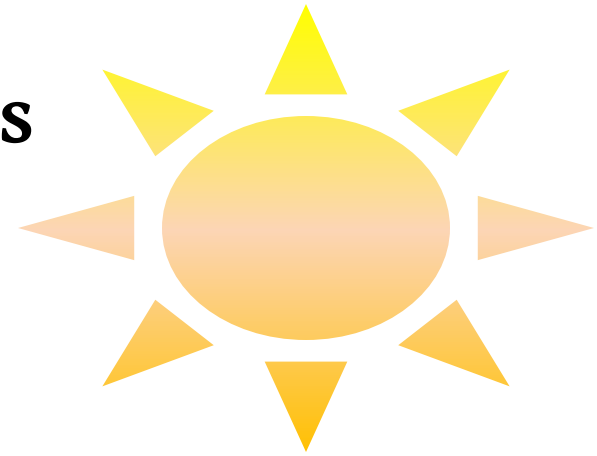
5. Описание интерфейса

- 5.1. Модульный интерфейс
 - 5.1.1. Описание модуля 1
 - 5.1.2. Описание модуля 2
- 5.2. Интерфейс процесса
 - 5.2.1. Описание процесса 1
 - 5.2.2. Описание процесса 2

6. Детальное проектирование

- 6.1. Детальное проектирование модулей
 - 6.1.1. Модуль 1: детали
 - 6.1.2. Модуль 2: детали
- 6.2. Детальное проектирование данных
 - 6.2.1. Блок данных 1: детали
 - 6.2.2. Блок данных 2: детали

Fundamental Patterns



Паттерн делегирования

Неизменяемый объект

Интерфейс

MVC, MVP, MVVM

S.O.L.I.D.

Aggregation

Факультет



Студент



Студент



Студент



```
class CFaculty // Класс Факультет
{
private:
    vector<CStudent*> m_Students;
public:
    void AddStudent( CStudent* pStud )
    {
        m_Students.push_back( pStud );
    }
    // делегирование действия Студенту
    void PrintStudentName( int nStud )
    {
        m_Students[ nStud ]->PrintName();
    }
}
```

```
void main()
{
    CStudent * pStudent = new CStudent("Bob");
    {
        CFaculty cFaculty;
        cFaculty.AddStudent( pStudent );
        cFaculty.PrintStudentName( 0 );
    } // Факультета уже нет, но Студент еще есть

    delete pStudent;
}
```

Composition

```
class CStadium           // класс Стадион
{
private:
    vector<CSector> m_Sectors;
public:
    void AddSector( CSector cSector )
    {
        m_Sectors.push_back( cSector );
    }
    // делегирование действия Сектору
    void PrintSectorInfo( int nSector )
    {
        m_Sectors[ nSector ].PrintInfo();
    }
}
```

```
void main()
{
    CStadium cStadium;
    {
        CSector cSector;
        cStadium.AddSector( cSector );
    }
    // Сектора уже нет, но у Стадиона своя копия
    cStadium.PrintSectorInfo( 0 );
}
```



OOP Basics Once Again

Класс «Животное»

```
class CAnimal
{
    private:
        int nColor;
    protected:
        int nSize;

    public:
        void Sound( )
        {
            cout << "Aaarrghh!!!" << endl;
        }
}
```

Класс «Собака» (наследуется от Жив.)

```
class CDog : public CAnimal
{
    public:
        void Bring() { ... }

        void Sound()
        {
            cout << "Bow Wow!!!" << endl;
        }
}
```

Демонстрация раннего (статического) связывания

```
void main()
{
    CDog d;
    d.Sound();           // Напечатает "Bow Wow!!!"
    d.CAnimal::Sound(); // можно вызвать родительский Sound() (если он public); напечатает «Aaarrghh!!!»
    CAnimal* a = &d;    // статическое связывание! Будет вызываться метод CAnimal
    a->Sound();          // Напечатает "Aaarrghh!!!"
}
```

OOP Basics Once Again

Класс «Животное»

```
class CAnimal
{
    private:
        int nColor;
    protected:
        int nSize;

    public:
        virtual void Sound( )
        {
            cout << "Aaarrghh!!!" << endl;
        }
}
```

Класс «Собака» (наследуется от Жив.)

```
class CDog : public CAnimal
{
    public:
        void Bring() { ... }

        void Sound()
        {
            cout << "Bow Wow!!!" << endl;
        }
}
```

Демонстрация позднего (динамического) связывания

```
void main()
{
    CDog d;
    CAnimal a;
    CAnimal* m[2] = { &d, &a }; // позднее связывание! Будет вызываться соотв. виртуальный метод
    m[0]->Sound(); // Напечатает "Bow Wow!!!"
    m[1]->Sound(); // Напечатает "Aaarrghh!!!"
}
```

OOP Basics Once Again

Пример абстрактного класса

```
class CAnimal
{
    private:
        int nColor;
    protected:
        int nSize;

    public:
        virtual void Move( ) =0;

        virtual void Sound( )
        {
            cout << "Aarrghh!!!" << endl;
        }
}
```

Пример интерфейса (ООП)

```
class IMovable
{
    public:
        virtual void Move()=0;
        virtual void Stop()=0;
}
```

Canvas

