Министерство Образования, Культуры и Исследований

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

Отчёт по предмету

,, Проектирование информационных систем ”

Лабораторная работа № 1

,,Архитектура АИС ”

**Научный руководитель**: Гладей Анатолий

**Автор**: студент группы I2102 (РУ), Жураковски Владислав

**КИШИНЕВ – 2023**

**Оглавление**

[I. Архитектура системы 1](#_Toc7165)

[1.1 Введение 1](#_Toc25654)

[1.2 Сопутствующие понятия 1](#_Toc31792)

[1.3 Виды архитектуры 2](#_Toc17223)

[1.3.1 Логическая архитектура 2](#_Toc24096)

[1.3.2 Физическая архитектура 2](#_Toc9639)

[1.4 Архитектурное описание 3](#_Toc3469)

[1.5 Типы групп описаний архитектуры 4](#_Toc18011)

[1.5.1 Функциональная группа описаний 4](#_Toc1695)

[1.5.2 Логическая группа описаний 4](#_Toc4413)

[1.5.3 Физическая группа описаний 4](#_Toc11473)

[II. Архитектура программного обеспечения 5](#_Toc3774)

[2.1 Введение 5](#_Toc22060)

[2.2 Виды (views) 5](#_Toc22193)

[2.3 Архитектурные шаблоны 7](#_Toc13662)

[2.4 Фреймворк 8](#_Toc2709)

[III. Вывод 9](#_Toc16746)

[IV. Литература 9](#_Toc6893)

# 

# I. Архитектура системы

## 1.1 Введение

**Архитектура системы** — [принципиальная](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF" \o "Принцип) организация [системы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0" \o "Система), воплощённая в её [элементах](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_(%D1%84%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D0%B8%D1%8F)" \o "Элемент (философия)), их взаимоотношениях друг с другом и со средой, а также принципы, направляющие её [проектирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Проектирование) и эволюцию.

Понятие архитектуры в значительной мере [субъективно](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D1%84%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D0%B8%D1%8F)" \o "Субъект (философия)) и имеет множество противоречивых толкований; в лучшем случае оно отображает общую точку зрения команды разработчиков на результаты проектирования системы.

В настоящее время существует сильная тенденция рассматривать архитектурное и не архитектурное проектирование как различные виды деятельности; делаются попытки определить их как отдельные практики, однако эти виды проектирования в значительной мере «переплетены». Архитектурные решения в сравнении с обычными проектными решениями рассматриваются как более абстрактные, концептуальные и глобальные; они нацелены на успех всей миссии и на наиболее высокоуровневые структуры системы.

## 1.2 Сопутствующие понятия

Для более подробного описания принципов построения архитектуры стандарт ISO/IEC/IEEE 42010-2011 вводятся следующие понятия:

* **Архитектурная группа описаний**  — представление системы в целом с точки зрения связанного набора интересов. Каждая группа описаний относится к одному или более [стейкхолдеру](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%B9%D0%BA%D1%85%D0%BE%D0%BB%D0%B4%D0%B5%D1%80" \o "Стейкхолдер). Термин «группа описаний» употребляется для выражения архитектуры системы при некотором методе описания.
* **Архитектурное описание**  — рабочий продукт, использующийся для выражения архитектуры.
* **Архитектурный подход**  — соглашения, принципы и практики для описания архитектуры, установленные для конкретной области применения или конкретным сообществом стейкхолдеров.
* **Архитектурный метод описания**  — спецификация соглашений для конструирования и применения группы описаний. Шаблон или образец, по которому разрабатываются отдельные группы описаний посредством установления назначений и аудитории для группы описаний, а также приемы их создания и анализа. Метод описания устанавливает соглашения, по которым группа описаний создается, отображается и анализируется. Тем самым метод описания определяет языки, применяемые для определения группы описаний, а также все связанные методы моделирования или приемы анализа, применяемые к данным представлениям группы описаний. Данные языки и приемы применяются для получения результатов, имеющих отношение к адресуемым интересам.
* **Вид модели** — соглашения по средствам моделирования, например [диаграммы классов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2" \o "Диаграмма классов), организационные диаграммы и т. д.

## 1.3 Виды архитектуры

### 1.3.1 Логическая архитектура

Логическая архитектура поддерживает функционирование системы на протяжении всего её жизненного цикла на логическом уровне. Она состоит из набора связанных технических концепций и принципов. Логическая архитектура представляется с помощью методов, соответствующих тематическим группам описаний, и как минимум, включает в себя функциональную архитектуру, поведенческую архитектуру и временную архитектуру.

**Функциональная архитектура** — набор функций и их подфункций, определяющих преобразования, осуществляемые системой при выполнении своего назначения.

**Поведенческая архитектура** — соглашение о функциях и их подфункциях, а также интерфейсах входы и выходы, которые определяют последовательность выполнения, условия для управления или потока данных, уровень производительности, необходимый для удовлетворения системных требований. Поведенческая архитектура может быть описана как совокупность взаимосвязанных сценариев, функций и/или эксплуатационных режимов.

**Временная архитектура** — классификацией функций системы, которая получена в соответствии с уровнем частоты её исполнения. Временная архитектура включает в себя определение синхронных и асинхронных аспектов функций. Мониторинг решений, который происходит внутри системы, следует той же временной классификации.

### 1.3.2 Физическая архитектура

Цель проектирования физической архитектуры заключается в создании физического, конкретного решения, которое согласовано с логической архитектурой и удовлетворяет установленным системным требованиям.

После того, как логическая архитектура определена, должны быть идентифицированы конкретные физические элементы, которые поддерживают функциональные, поведенческие, и временные свойства, а также ожидаемые свойства системы, полученные из нефункциональных требований к системе.

Физическая архитектура является систематизацией физических элементов , которые реализуют спроектированные решения для продукта, услуги или предприятия. Она предназначена для удовлетворения требований к системе и элементам логической архитектуры и реализуется через технологические элементы системы. Системные требования распределяются как на логическую, так и физическую архитектуру. Глобальная архитектура системы оценивается с помощью системного анализа и, после выполнения всех требований, становится основой для реализации системы.

## 1.4 Архитектурное описание

Архитектура может быть зафиксирована с помощью полного архитектурного описания. Стандарт ISO/IEC/IEEE 42010-2011 предписывает различать концептуальную архитектуру системы и одно из описаний данной архитектуры, являющееся конкретным продуктом или артефактом.

В сложных системах АО может разрабатываться не только для системы в целом, но и для компонентов системы. Два разных концептуальных АО могут включать группы описаний, которые будут соответствовать одному и тому же методу описания. Хотя системы, описываемые данными двумя группами описаний, будут соотноситься как целое и часть, это не пример множества групп описаний, соответствующих одному методу. Эти АО считаются отдельными, даже хотя они связаны через системы (Рисунок 1).

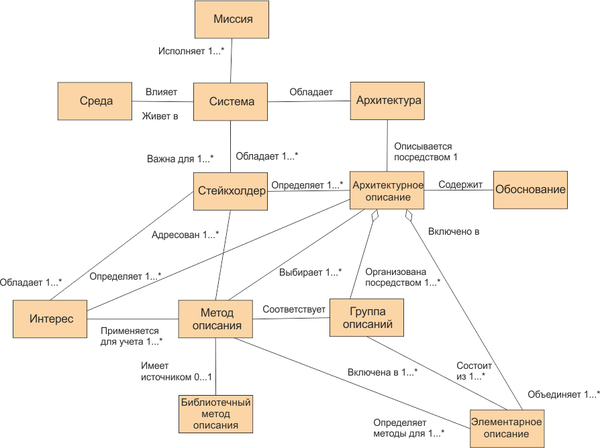


Рисунок 1

## 1.5 Типы групп описаний архитектуры

### Существует три типа группы описаний: функциональные, логические и физические. Каждая из групп предназначена для описания собственных точек зрения и соответствующего им уровня сложности.

### 1.5.1 Функциональная группа описаний

### Данная группа обеспечивает представление с точки зрения пользователей или операторов, которое включает продукты, относящиеся к фазам, сценариям и потокам задач операционной системы. Информационный поток может быть рассмотрен с пользовательского ракурса, также описываются и пользовательские интерфейсы. Примером продуктов, которые могут быть включены в это описание, будут функциональные данные или графики, сценарное описание (включая использование кейсов), блок-схемы задач, организационные диаграммы и схемы информационных потоков.

### 1.5.2 Логическая группа описаний

### Данная группа обеспечивает представление с точки зрения руководителя или заказчика. Логическое представление включает продукты, которые определяют системные границы с её окружением и функциональные интерфейсы с внешними системами, также основные функции и поведение системы, потоки информации, внутренние и внешние наборы данных, внутренних и внешних пользователей, и внутренние функциональные интерфейсы. Примером продуктов могут быть [блочные диаграммы функциональных потоков](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2&action=edit&redlink=1" \o "Блочная диаграмма функциональных потоков (страница отсутствует)), контекстные диаграммы,  данные поточных диаграмм.

### 1.5.3 Физическая группа описаний

### Данная группа обеспечивает представление с точки зрения проектировщиков. Включает в себя:

### продукты, которые определяют физические границы системы

### физические компоненты системы и то, как они взаимодействуют и влияют друг на друга

### внутренние базы данных и структуры данных

### инфраструктуру информационных технологий (ИТ) системы

### внешнюю ИТ-инфраструктуру, с которой система взаимодействует

### требования, необходимые для развития системы

### Продукт может включать в себя физические блок-схемы на довольно высоком уровне детализации, топологии [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85" \o "База данных), интерфейс управления документами и стандарты. Все из трёх типов групп должны присутствовать в каждом описании архитектуры.

# II. Архитектура программного обеспечения

## 2.1 Введение

### **Архитектура программного обеспечения** — совокупность важнейших решений об организации программной системы. Архитектура включает:

### выбор структурных элементов и их [интерфейсов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81" \o "Интерфейс), с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов

### соединение выбранных элементов структуры и поведения во всё более крупные системы

### архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение

### [Документирование](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5" \o "Документация на программное обеспечение) архитектуры [программного обеспе́чения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5" \o "Программное обеспечение) (ПО) упрощает коммуникацию разработчиков, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы, [повторно использовать](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0" \o "Повторное использование кода) [компоненты](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0" \o "Компонентная архитектура) и [шаблоны](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Шаблон проектирования) проекта в других.

## 2.2 Виды (views)

### Архитектура ПО обычно содержит несколько видов, аналогичных типам чертежей в строительстве зданий. В [онтологии](https://ru.wikipedia.org/wiki/%D0%9E%D0%BD%D1%82%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Онтология (информатика)), установленной ANSI / IEEE 1471—2000, виды являются экземплярами точки зрения, где точка зрения существует для описания архитектуры с точки зрения заданного множества заинтересованных лиц.

Архитектурный вид состоит из двух компонентов:

* Элементы
* Отношения между элементами

Архитектурные виды можно поделить на три основных типа:

* **Модульные виды** (module views) — показывают систему как структуру из программных блоков.
* **Компоненты-и-коннекторы** (component-and-connector views) — показывают систему как структуру из параллельно запущенных элементов (компонентов) и способов их взаимодействия (коннекторов).
* **Размещение** (allocation views) — показывает размещение элементов системы во внешних средах.

Примеры модульных видов:

* **Декомпозиция** (decomposition view) — состоит из модулей в контексте отношения «является подмодулем».
* **Использование** (uses view) — состоит из модулей в контексте отношения «использует» (то есть один модуль использует сервисы другого модуля).
* **Вид уровней** (layered view) — показывает структуру, в которой связанные по функциональности модули объединены в группы (уровни).
* **Вид классов/обобщений** (class/generalization view) — состоит из классов, связанные через отношения «наследуется от» и «является экземпляром».

Примеры видов компонентов-и-коннекторов:

* **Процессный вид** (process view) — состоит из процессов, соединённых операциями коммуникации, синхронизации и/или исключения.
* **Параллельный вид** (concurrency view) — состоит из компонентов и коннекторов, где коннекторы представляют собой «логические потоки».
* **Вид обмена данными** (shared-data (repository) view) — состоит из компонентов и коннекторов, которые создают, сохраняют и получают постоянные данные.
* **Вид клиент-сервер** (client-server view) — состоит из взаимодействующих клиентов и серверов, а также коннекторов между ними (например, протоколов и общих сообщений).

Примеры видов размещения:

* **Развертывание** (deployment view) — состоит из программных элементов, их размещения на физических носителях и коммуникационных элементов.
* **Внедрение** (implementation view) — состоит из программных элементов и их соответствия файловым структурам в различных средах (разработческой, интеграционной и т. д.).
* **Распределение работы** (work assignment view) — состоит из модулей и описания того, кто ответственен за внедрение каждого из них.

Хотя было разработано несколько языков для описания архитектуры программного обеспечения, в настоящий момент нет согласия по поводу того, какой набор видов должен быть принят в качестве эталона. В качестве стандарта «для моделирования программных систем (и не только)» был создан язык UML.

## 2.3 Архитектурные шаблоны

Для удовлетворения проектируемой системы атрибутам качества применяются архитектурные шаблоны (паттерны). Каждый шаблон имеет свои задачи и свои недостатки.

Примеры архитектурных шаблонов:

* Многоуровневый шаблон (Layered pattern). Система разбивается на уровни, которые на [диаграмме](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0" \o "Диаграмма) изображаются один над другим. Каждый уровень может вызывать только уровень на один ниже его. Таким образом, разработку каждого уровня можно вести относительно независимо, что повышает модифицируемость системы. Однако при таком подходе сложность системы возрастает, а её производительность снижается.
* Шаблон посредника (Broker pattern). Когда в системе присутствует большое количество модулей, их прямое взаимодействие друг с другом становится слишком сложным. Для решения проблемы вводится посредник (например, шина данных), по которой модули общаются друг с другом. Таким образом, повышается функциональная совместимость модулей системы. Все недостатки вытекают из наличия посредника: он понижает производительность, его недоступность может сделать недоступной всю систему, он может стать объектом атак и узким местом системы.
* Шаблон «[Модель-Представление-Контроллер](https://ru.wikipedia.org/wiki/Model-View-Controller" \o "Model-View-Controller)» (Model-View-Controller pattern, MVC). Так как требования к интерфейсу меняются чаще всего, то возникает потребность часто его модифицировать, при этом сохраняя корректное взаимодействие с данными (чтение, сохранение). Для этого в шаблоне MVC интерфейс отделён от данных. Это позволяет менять интерфейсы, равно как и создавать их разные варианты. В MVC система разделена на:
  + модель, хранящую данные;
  + представление, отображающее часть данных и взаимодействующее с пользователем;
  + контроллер, являющийся посредником между видами и моделью.

Однако концепция MVC имеет и свои недостатки. В частности, из-за усложнения взаимодействия падает скорость работы системы.

* Клиент-серверный шаблон (Client-Server pattern). Если есть ограниченное число ресурсов, к которым требуется ограниченный правами доступ большого числа потребителей, то удобно реализовать клиент-серверную архитектуру. Такой подход повышает масштабируемость и доступность системы. Но при этом сервер может стать узким местом системы, при его недоступности становится недоступна вся система.

## 

## 2.4 Фреймворк

Фреймворк — это структура, которая предоставляет заготовки, шаблоны и правила для разработки программных систем. Фреймворк определяет структуру программы и облегчает объединение разных модулей проекта. Здесь мы рассмотрим ключевые аспекты фреймворков:

* Определение фреймворка: Фреймворк, иногда называемый каркасом, является программным обеспечением, которое облегчает разработку программных систем, предоставляя структуру и правила для их создания.
* Сравнение с библиотекой: Фреймворк отличается от библиотеки тем, что фреймворк диктует правила построения архитектуры приложения и определяет поведение по умолчанию. Библиотека предоставляет набор функций, но не влияет на архитектуру приложения.
* Инверсия управления: Фреймворк может включать инверсию управления, где пользовательский код может реализовывать конкретное поведение, встраиваемое в более общий код фреймворка. Это отличает фреймворк от библиотеки, где пользовательский код вызывает функции библиотеки.
* Примеры фреймворков: Примерами фреймворков являются веб-каркасы, такие как Rocket на Rust, Zend Framework и Symfony на PHP, а также фреймворк Django, написанный на Python.
* Структура фреймворка: Фреймворк состоит из конкретных и абстрактных классов, а также спецификаций их взаимоотношений. Абстрактные классы представляют точки расширения, где пользовательский код может быть использован или адаптирован.
* Стандартизация структуры приложения: Одним из преимуществ фреймворков является стандартизация структуры приложения, что облегчает создание средств для автоматического создания графических интерфейсов и обеспечивает "стандартность" внутренней реализации кода приложения.
* Процесс создания фреймворка: Процесс создания фреймворка включает выбор подмножества задач и их реализацию. В результате конкретные классы реализуют отношения между классами, а абстрактные классы представляют точки расширения.
* Примеры фреймворков: Примерами кроссплатформенных каркасов приложений являются widget toolkit, wxWidgets, Qt и другие.
* Фреймворки и паттерны проектирования имеют много общего, но различие заключается в том, что фреймворк представляет собой структуру для решения конкретной задачи, в то время как паттерн представляет собой знание о том, как решать конкретные типовые проблемы в проектировании программ.

# III. Вывод

Архитектура системы и архитектура программного обеспечения проявляеться в способности обеспечивать эффективное и стабильное функционирование системы, а также в способности легко вносить изменения и обновления в систему. Архитектурная концепция дает ясное представление о структуре и компонентах системы, что способствует легкости в понимании и сотрудничестве среди разработчиков. Многоуровневая архитектура позволяет разделять функциональность системы на уровни, упрощая ее сопровождение и модификацию. Устойчивость и масштабируемость архитектуры позволяют системе справляться с различными нагрузками и обеспечивать надежную работу.

# IV. Литература

1. <http://it-claim.ru/Education/Course/ISDevelopment/Lecture_3.pdf>
2. <https://academia-moscow.ru/ftp_share/_books/fragments/fragment_16353.pdf>
3. http://lib.ulstu.ru/venec/disk/2017/574.pdf