Дюжева Мария Александровна

# UML (унифицированный язык программирования)

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

UML упрощает процесс проектирования ИС, снижая его стоимость и повышая эффективность.

## Цели разработки

* Предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий разрабатывать осмысленный модели и обмениваться ими.
* Предусмотреть механизмы расширяемости и специализации для расширения базовых концепций
* Обеспечить независимость от конкретных языков программирования и процессов разработки
* Обеспечить формальную основу для понимания этого языка моделирования (язык должен быть точным и доступным для понимания)
* Стимулировать рост рынка объектно-ориентированных средств

## Типы диаграмм

* Структурные
* Классов
* Объектов
* Компонентов
* Развертывания
* Поведения
* Прецедентов
* Состояний
* Активности
* Взаимодействия
* Кооперации
* Последовательности

## Основные понятия языка UML

Объект – сущность реального мира или концептуальная сущность, либо программная единица, состоящая из атрибутов (полей) или методов для их обработки. Характеризуется состоянием поведением и уникальностью. Состояние определяется атрибутами, а поведение – методами

Класс – описание множества объектов с общими атрибутами, определяющими состояние, и операциями, определяющими поведение. Класс изображается прямоугольником, разделенным на три секции. В первом прямоугольнике приводится имя класса, а во втором – список атрибутов, в третьем – список методов. Ассоциация – наиболее общий тип отношений первого рода, связанных с передачей сообщений ассоциированными класса путем делегирования.

## Диаграмма прецедентов

Диаграмма прецедентов применяется для моделирования вида системы с точки зрения внешнего наблюдателя. На диаграмме прецедентов графически показана совокупность прецедентов и субъектов, а также отношения между ними. Прецедент (use case) – это описание множества последовательностей действий (включая их варианты), которые выполняются системой для того, чтобы актер получил результат, имеющий для него определенное значение. Стандартным графическим обозначением прецедента является эллипс, внутри которого содержится краткое название прецедента ил имя в форме глагола с пояснительными словами.

## Диаграмма классов

Диаграмма классов – основной способ описания структуры системы. Поскольку UML в первую очередь объектно-ориентированный язык, классы в нем являются основным, так называемым строительным материалом. Для построения диаграммы классов необходимо составить список атрибутов и операций для выбранных классов.

## Диаграмма объектов

Диаграмма объектов демонстрирует полный или частичный снимок моделируемой системы в заданный момент времени. На диаграмме объектов отображаются экземпляры классов (объекты) системы с указанием текущих значений их атрибутов и связей между ними.

С помощью диаграмм объектов, как и с помощью диаграмм классов, моделируют статический вид системы с точки зрения проектирования или процессов, но принимая во внимание реальные экземпляры или прототипы. Этот вид описывает главным образом функциональные требования к системе, то есть услуги, которые она должна предоставлять конечным пользователям.

# Диаграмма классов

Атрибут - это именованное свойство класса, описывающее диапазон значений, которые может принимать экземпляр атрибута. Класс может иметь любое число атрибутов или не иметь ни одного. В последнем случае блок атрибутов оставляют пустым. Атрибут представляет некоторое свойство моделируемой сущности, которым обладает все обеты данного класса. Имя атрибута, как и имя класса может представлять собой текст. На практике для именования атрибута используются одно или несколько коротких существительных.

Операция - это реализация метода класса. Класс может иметь любое число операций либо не иметь ни одной. Часто вызов операции объекта изменяет его атрибуты. Графически операции представлены в нижнем блоке описания класса. Допускается указание только имен операций. Имя операции, как и имя класса, дробно представляет собой текст. На практике для именования операции используются короткие глагольные конструкции, описывающие некое поведение класса. Которому принадлежит операция. Обычно каждое слово в именно операции пишется с заглавной буквы.

Типы связей в UML:

- Зависимость

- Ассоциация

- Обобщение

- Реализация

Зависимость - семантически представляет собой связь между двумя элементами модели, в которой изменения одного элемента (независимого) может привести к изменению механики другого элемента (зависимого). Графически представлена пунктирной линией, иногда со стрелкой, направленной к той сущности, которая зависит ещё одна, может быть снабжена меткой.

Ассоциация - это структурная связь между элементами модели, которая омывает набор связей, существующих между объектами. Ассоциация показывает, что объекты одной сущности (класс) связанны с объектами другой сущности таким образом, что можно перемешаться от объектов одного класса к другому.

Множественность ассоциации представляет собой диапазон целых чисел, указывающих возможное количество связанных объектов. Он записывается в виде выражения с минимальным и максимальным значением, для их распределения используются две точки. Уставлявшая множественность дальнего конца ассоциации, вы указываете, сколько объектов может существовать на дальнем конце ассоциации для каждого объекта.

Агрегация - особая разновидность ассоциации, представляющая структурную связь целого с его частями. Как тип ассоциации, агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое).

Агрегация встречается, когда Один класс является коллекцией или контейнером других.

Композиция - обклей строки вариант агрегации, Известна также как агрегация по значению

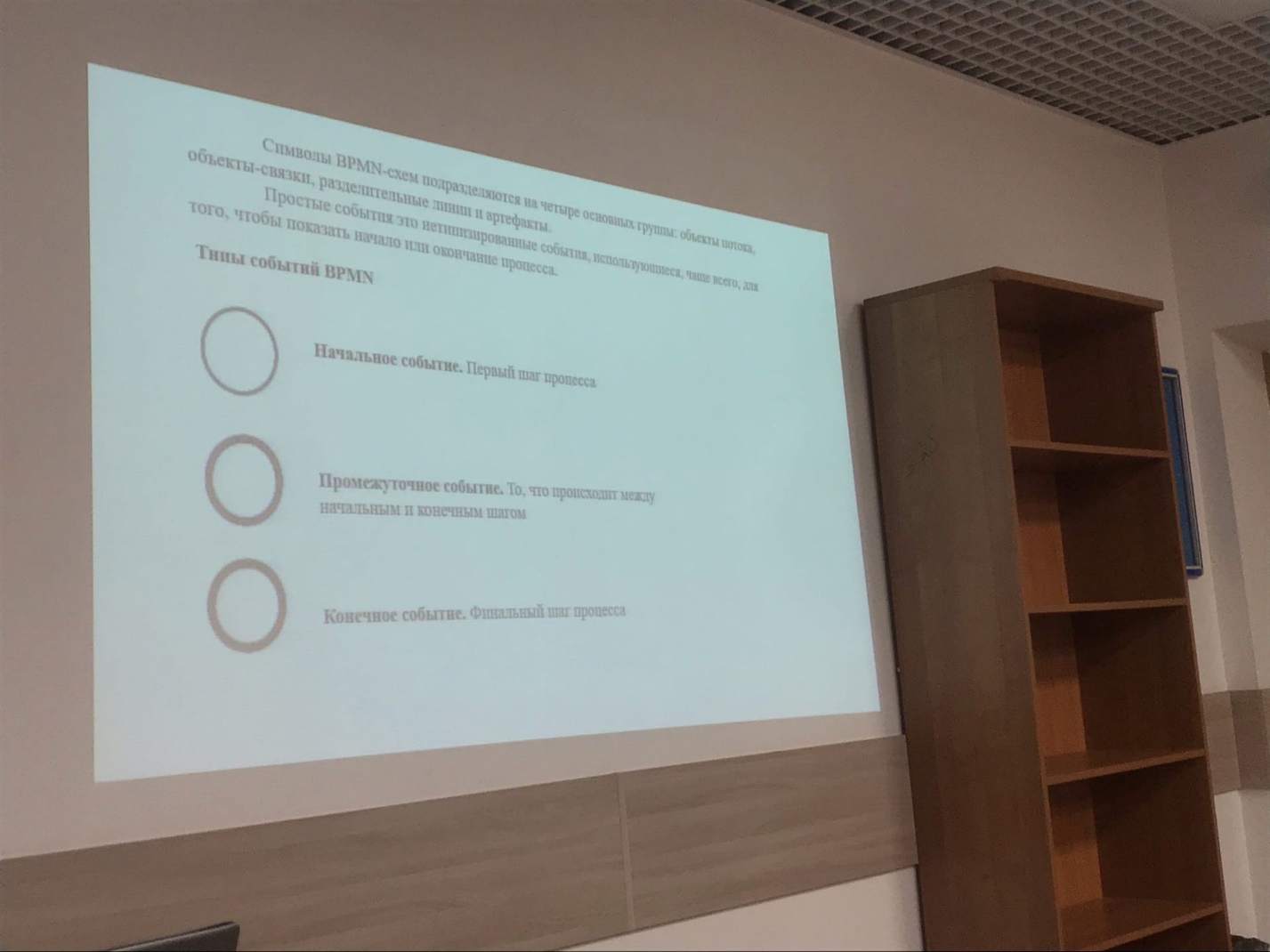
Композиция - это форма агрегации с четко выраженными отношениями владения и совпадением времени жизни частей целого.

Обобщение - выражает специализацию или наследованное, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в видео сплошной линии с пустой стрелкой, указывающей на родителя.

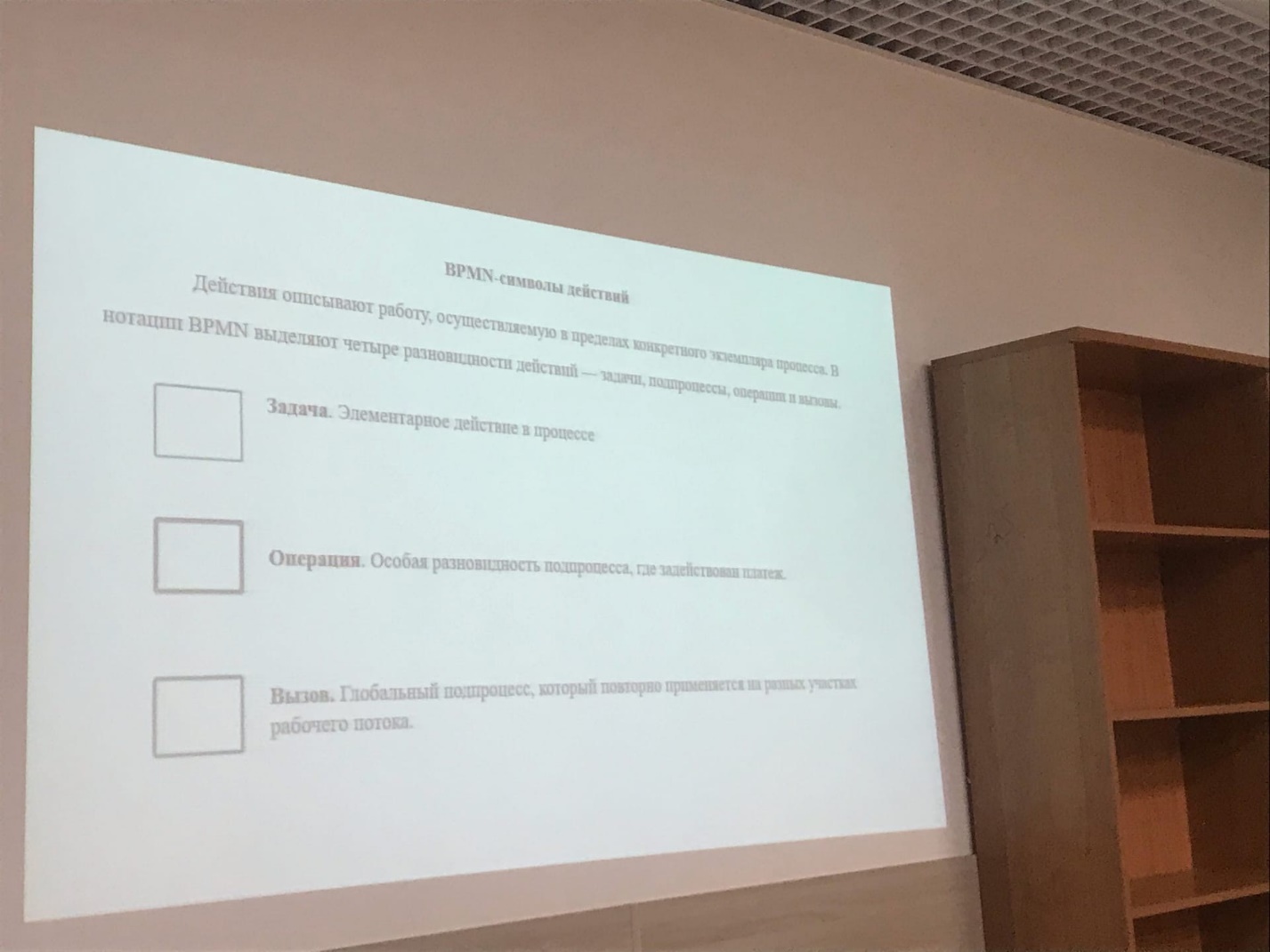
Реализация - это семантическая связь между классами, когда один из них (поставщик) определяет соглашение, которого второй (клиент) обязан придерживаться.

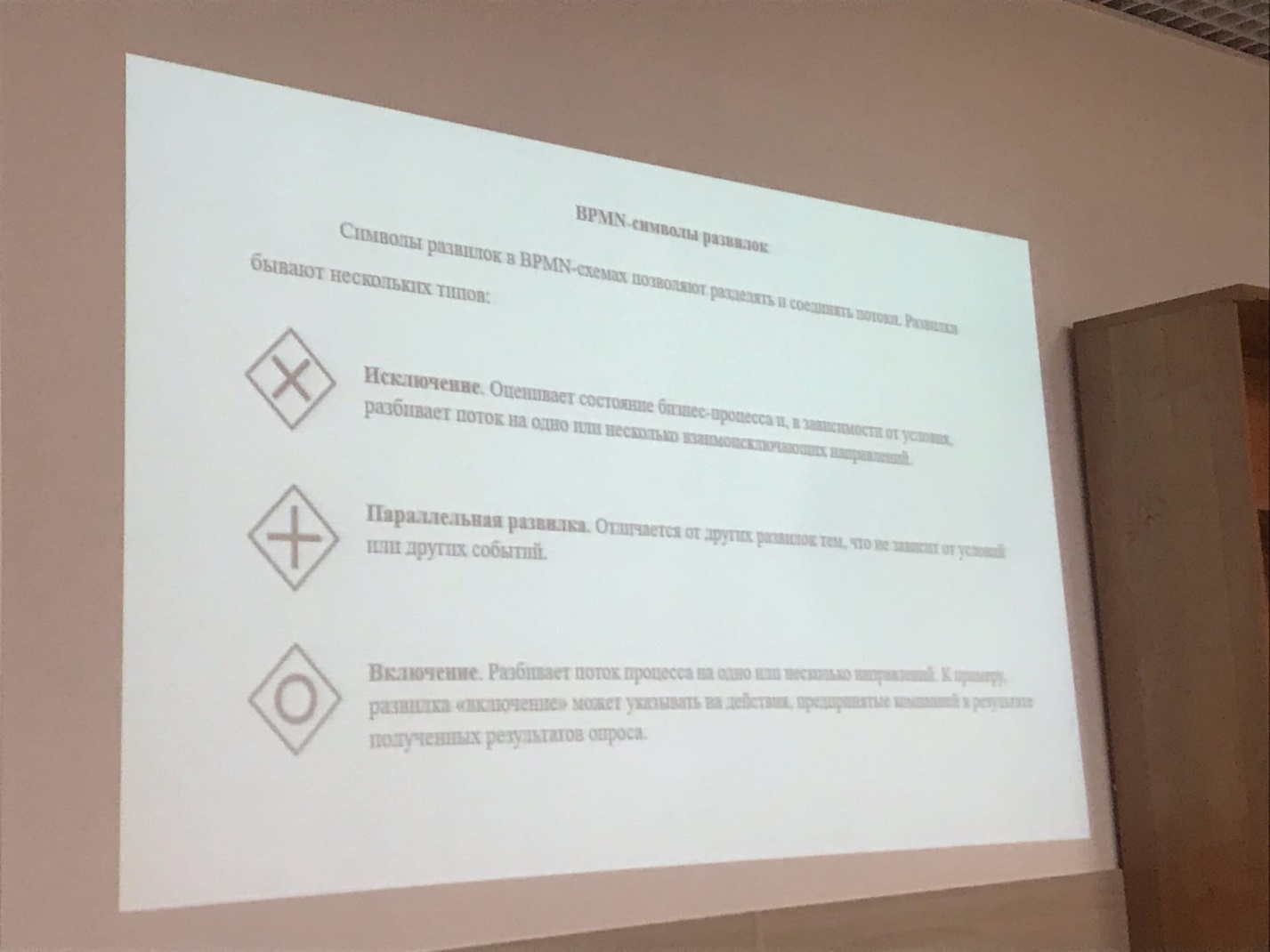
# BPMN – Нотация моделирования бизнес-процессов

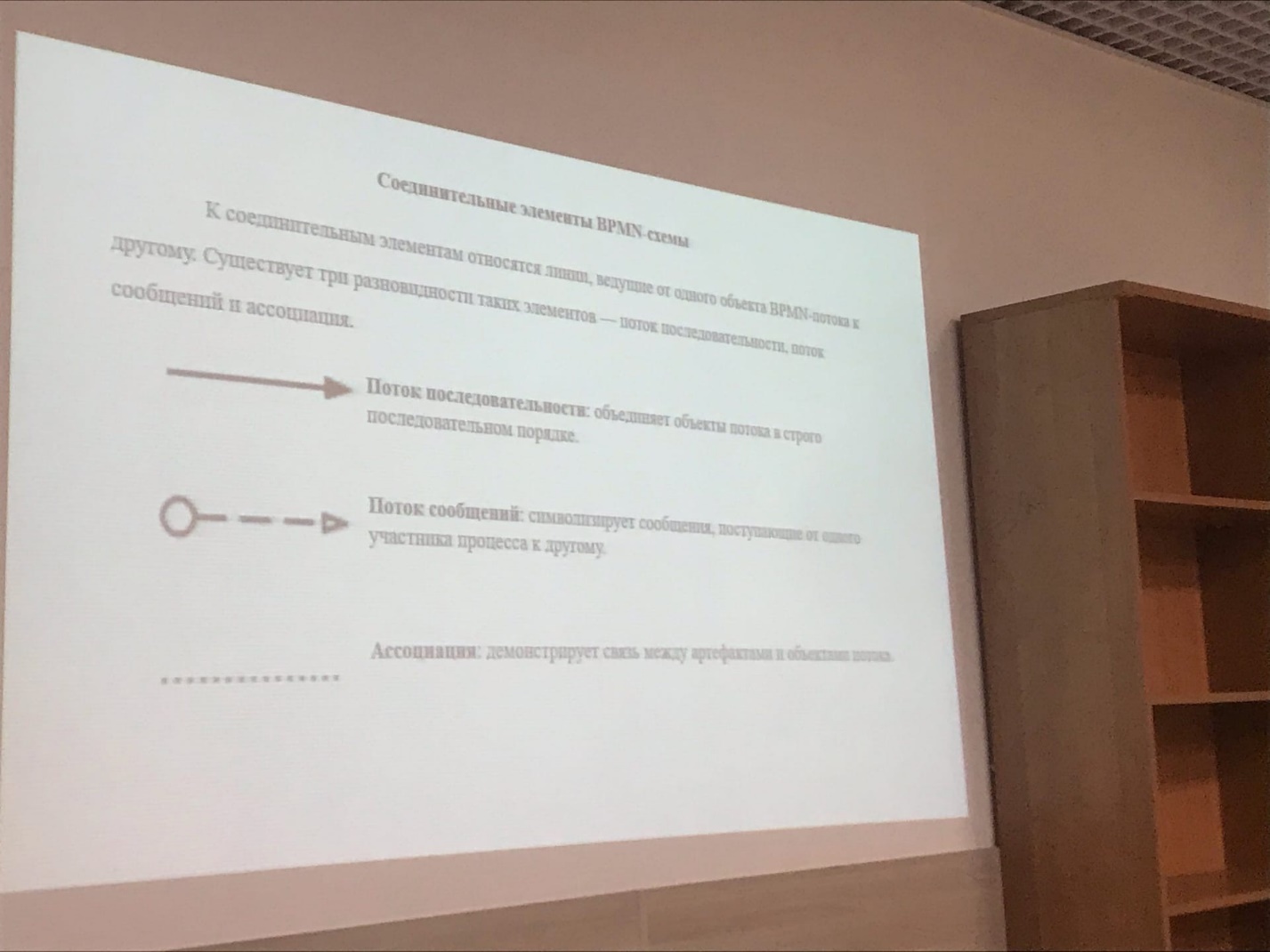
Это метод составления блок-схем выполнения бизнес-процессов от начала и до конца. Спецификация BPMN описывает условные обозначения и их описание в XML для отображения бизнес-процессов в виде диаграмм.











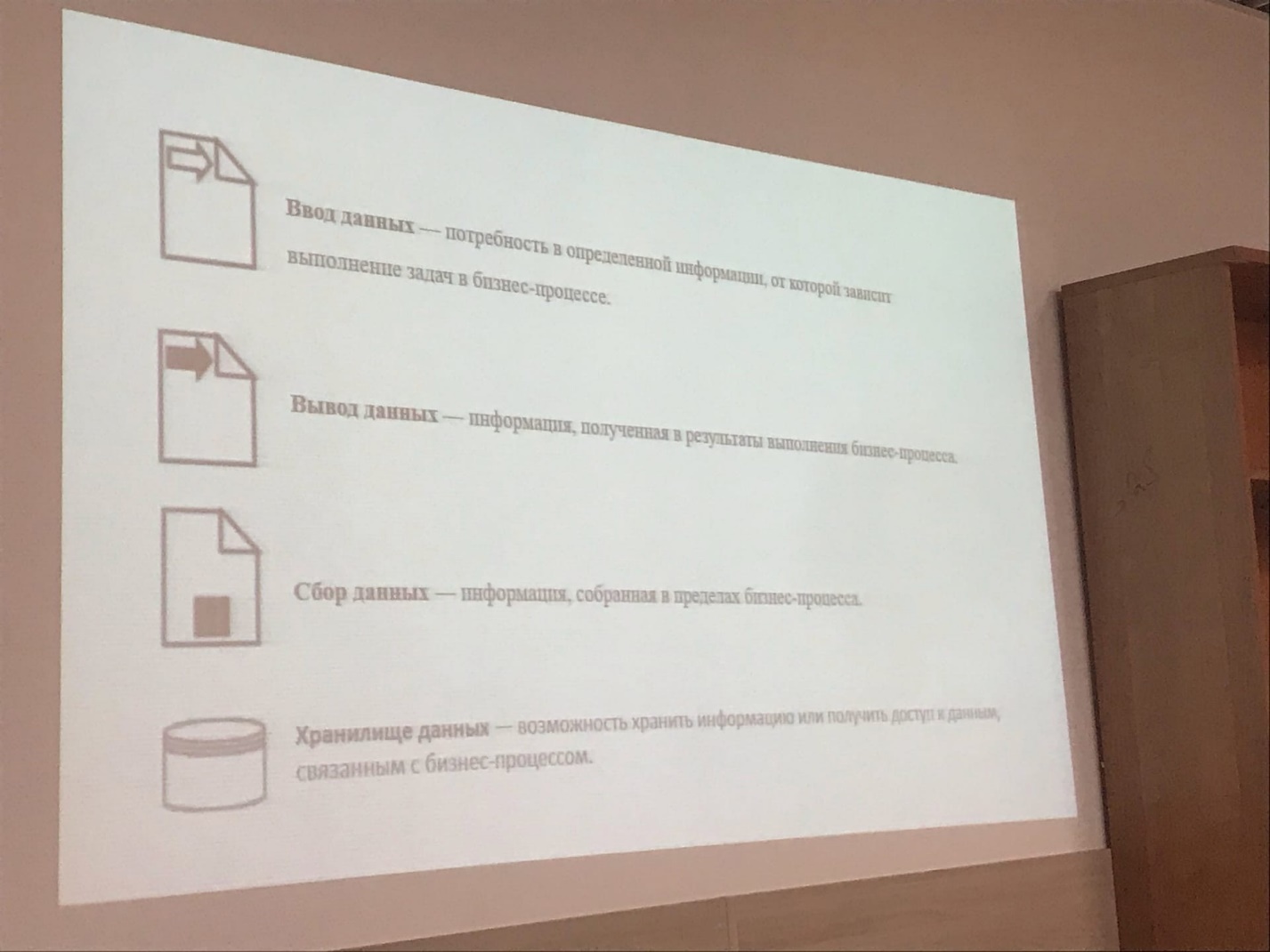
Разделительные дорожки помогают упорядочить разные аспекты процесса в пределах BPMN. Каждый аспект процесса – отдельная дорога.

Артефакты:

Аннотация – описание дополнительных участков модели

Группа – объединение задач или процессов в контексте общего процесса

Данные – информация, возникшая в результате процесса/требующая сбора или хранения



# Паттерн проектирования программного обеспечения

Паттерн – определенный способ построения программного кода для решения встречающихся проблем проектирования. Паттерны предоставляют ряд принципов для решения этих проблем.

Существует большое количество паттернов, но их можно объединить в 3 группы:

1. Порождающие
2. Структурные
3. Поведенческие

**Порождающие** – абстрагируют процесс порождения классов и объектов. Отвечают за удобное и безопасное создание новых объектов и целых семейств объектов.

Методы:

1. Фабричный метод – общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.
2. Абстрактная фабрика – позволяет создавать семейства связанных объектов, не привязываясь к конкретному классу создаваемых объектов.
3. Строитель – позволяет создавать сложные объекты пошагово. Строитель дает возможность использовать один и тот же код строительства для получения разных представлений объектов.
4. Прототип – порождающий паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.
5. Одиночка – гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

**Структурные паттерны** – Отвечают за построение удобных в поддержке иерархий классов.

Методы:

1. Адаптер – позволяет объектам с несовместимыми интерфейсами работать вместе.
2. Мост – разделяет абстракцию и реализацию, позволяя изменять их независимо.
3. Декоратор – позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные обертки.
4. Компоновщик – позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, как будто это единичный объект.

**Поведенческие паттерны** – решают задачи эффективного и безопасного взаимодействия между объектами программы.

Методы:

1. Стратегия – определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программы
2. Команда – превращает запросы в объекты, позволяя тем самым передавать их как аргументы при вызове методов, ставить запросы в очередь, а также поддерживать отмену операций
3. Наблюдатель – создает механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.
4. Шаблонный метод – определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

# Архитектура программного обеспечения

Архитектура программного обеспечения - набор внутренних структур ПО которые видны из разных точек зрения и состоят из компонентов их связей и взаимодействий между компонентами, а также доступных извне свойств этих компонентов.

Компонент – это произвольный структурный элемент программного обеспечения, который можно выделить, определив интерфейс взаимодействия между этим компонентом и всем, что его окружает

* Многослойная архитектура
* Многоуровневая архитектура
* Сервис-ориентированная архитектура
* Микросервисная архитектура

**Многослойная архитектура**

Программное обеспечение разделено на слои, лежащие друг на друге, каждый из них выполняет определенную обязанность

**Слой представления** – содержит пользовательский интерфейс и отвечает за обеспечение хорошего пользовательского опыта.

**Слой бизнес-логики** – содержит бизнес логику приложения. Он отделяет UI/UX от вычислений, связанных с бизнесом. Это позволяет с легкостью изменять логику в зависимости от постоянно меняющихся бизнес-требований, никак не влияя на другие слои.

**Слой передачи данных** – отвечает ха взаимодействие с постоянными хранилищами, такими как базы данных, и прочую обработку информации, которая не связана с бизнесом.

Данные и элементы управления переходят через каждый слой в дизайне и передаются от одного к другому.

Недостатки:

* Не предлагает большой масштабируемости
* Данные должны проходить по каждому слою, даже если нет необходимости передавать их с определенных слоев

**Многоуровневая архитектура**

Этот подход разделяет программное обеспечение на уровни по принципу взаимодействия «клиент-сервер». Архитектура может иметь один, два и больше уровней, разделяющих ответственности между поставщиком данных и потребителем.

В отличии от многослойной структуры, он предлагает масштабируемость, которая может быть, как горизонтальной, так и вертикальной.

**Сервис-ориентированная архитектура**

Эта архитектурная модель состоит их компонентов и приложений, которые связываются друг с другом

Она состоит из 5 элементов:

* Сервисы
* Сервисная шина (связующее по, обеспечивающее централизованный обмен сообщениями между различными информационными системами)
* Сервисный репозиторий
* Безопасность SOA
* Управление SOA

# Диаграмма состояния

Отображает состояния и переходы, а так же события, которые влияют на эти переходы.

Помогает визуализировать весь жизненный цикл объекта. Помогает лучше понять системы, основанные на состоянии.

Элементы:

Начальное состояние

Конечное состояние

Обычное состояние - Одобрено

Состояние активности: название состояния. Код, действие, выход – на сборке

Классическая развилка

Соединитель и разъединитель

Переход – это перемещение из оного состояния в другое.

Условие (всегда в квадратных скобках)

События – инциденты, которые заставляют переходить из одного состояния в другое.

Вход -> Действие -> Выход

Вход

Оформление

Получение

Выход (Машина получена)

# 4 курс

Соколова Лариса Алексеевна

Технологии разработки программного обеспечения - это совокупность процессов и методов создания программного продукта

Качество - совокупность свойств продукции, обусловливающих её пригодность удовлетворять определённые потребности в соответствии с её назначением.

Диаграммы:

1) Классов

2) Прецендентов

3) Объектов

4) Компонентов

5) Деятельности

6) Кооперации

7) Пакетов

8) Составной группы

9) Состояний

10) Последовательности

11) Коммуникации

12) Временная

13) Конечного автомата

14) Связей

15) Зацепления

16) Взаимодействий

17) Синхронизации

18) Активности

19) Развертывания

20) Структурная

21) Поведения

22) Схемы данных

23) Использования

24) Композитной структуры

25) Архитектуры

После тестирования тестирует пользователь при помощи обратной связи.

После 50% выполненного плана занимаемся сопровождением.

Продвижение, разработка и сопровождение, анализ рынка

# Принципы разработки графического пользовательского интерфейса

Основа всего – пользователь

ПО должно разрабатываться с учетом требований и пожеланий пользователя

Принципы разработки интерфейса – это высокоуровневые концепции представления, которые могут использоваться при проектировании ПО. Нужно определить какой из принципов более важен.

Приступая к проектированию, необходимо выделить важнейший принцип, который будет определяющим при поиске компромиссов.

Принципы – это руководство к действию

Принципы формулируются так:

1. Контроль пользователем интерфейса;
2. Уменьшение загрузки памяти;
3. Последовательность пользовательского интерфейса;

Принципа Хансена:

1. Знать пользователя
2. Сократить запоминание
3. Оптимизировать операции
4. Устранить ошибки

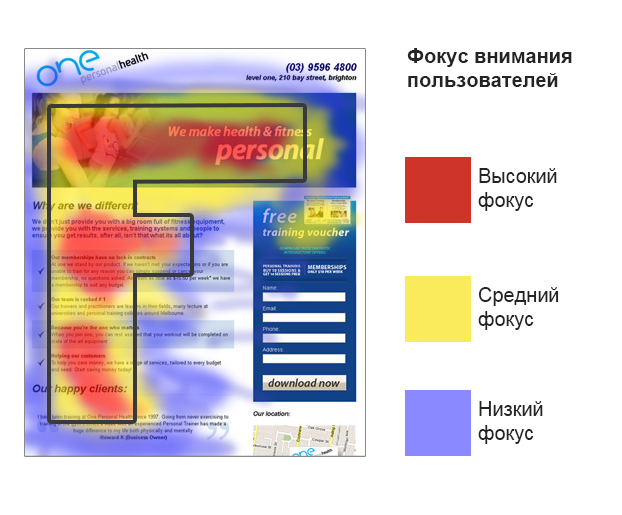
(Неправильный интерфейс может принести огромные убытки компании. Крупные разработчики ОС публикуют руководства подхода к проектированию интерфейса)

Правило 1: дать контроль пользователю – пусть пользователь сам решает, что хочет сделать

Принципы, которые дают пользователю контроль над системой:

1. Использовать режимы благоразумно (разрешить пользователю изменять интерфейс, но одновременно его ограничивать в определенных вещах)
2. Предоставить пользователю выбирать, работать либо мышью, либо клавиатурой, либо их комбинацией
3. Позволить пользователю сфокусировать внимание
4. Демонстрировать сообщения, которые помогут ему в работе
5. Создать условия для немедленных и обратимых действий, а также обратной связи
6. Обеспечить соответствующие пути и выходы
7. Приспосабливайте систему к пользователям с различным уровнем подготовки
8. Сделать пользовательский интерфейс более понятным
9. Дать пользователю возможность настраивать интерфейс по своему вкусу
10. Разрешить пользователю напрямую манипулировать объектами интерфейса

Карты распределения внимания пользователей







Правило 2: уменьшить нагрузку на пользователя

Принципы, позволяющие снизить нагрузку на память пользователя:

1. Не загружать кратковременную память
2. Полагаться на распознавание, а не на повторение
3. Представить визуальные заставки
4. Предусмотреть установки по умолчанию, команды UNDO и REDO
5. Предусмотреть «быстрые» пути
6. Активировать синтаксис действий с объектами
7. Использовать метафоры из реального мира
8. Применять раскрытие и объяснение понятий и действий
9. Увеличить визуальную ясность

Правило 3: сделать интерфейс совместимым

Принципы создания совместимости интерфейса:

1. Проектирование последовательного интерфейса
2. Общая совместимость всех программ
3. Сохранение результатов взаимодействия
4. Эстетическая привлекательность и цельность
5. Поощрение изучения

Найти эстетически-привлекательный шаблон сайта, мобильного приложения и настольного приложения

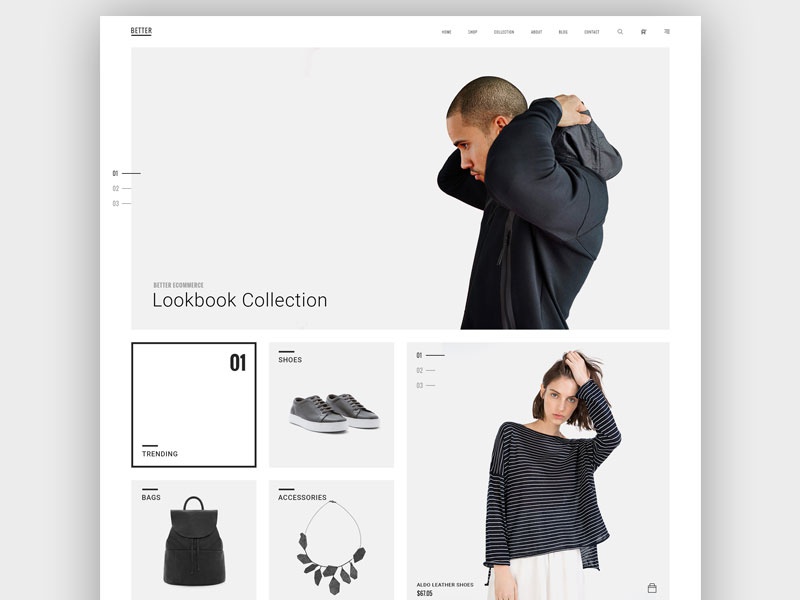


Рисунок 1 - Шаблон сайта



Рисунок 2 - Шаблон мобильного приложения

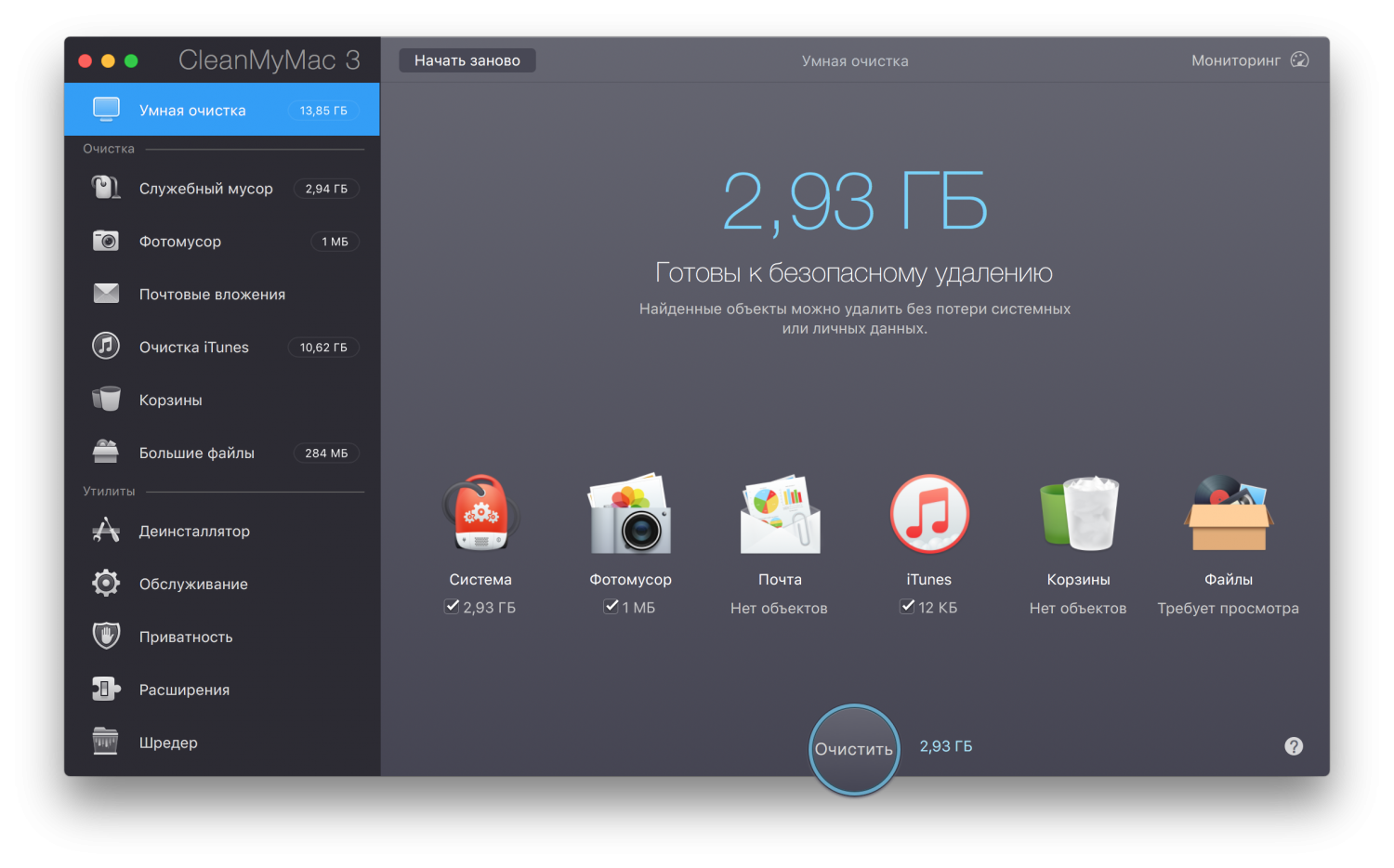


Рисунок 3 - Шаблон десктопного приложения

# Практическая работа 1

Пять ролей, несколько пользовательских сценариев, не менее 10 функций, не менее 7 схем окон ПИ, самостоятельно оценить дизайн

охранник, инженер микроэлектроники, специалист отдела качества, документовед.

«Раскрой материала» функция, производственный отдел,

Консультация, прием заявок, поставка/отгрузка,

Пользователь