Тема 1: Требования и анализ. Занятие 1.4 Требования и их виды. Сбор и анализ требований. Введение в UML. Диаграмма вариантов использования.

Цели занятия:

- Ознакомиться с понятием и важностью определения требований в процессе разработки программного обеспечения
- Изучить основы сбора и анализа требований.

Анализ требований

Правильное определение требований является одним из ключевых и критически важных шагов в процессе разработки программного обеспечения. Это этап, на котором выявляются и формулируются ожидания заказчика и пользователей от будущего продукта. Важность этого этапа не может быть недооценена по нескольким причинам:

Понимание потребностей заказчика: Правильное определение требований позволяет разработчикам и команде проекта точно понять, что требуется от создаваемого ПО. Это минимизирует риск разногласий между заказчиком и исполнителями, а также снижает вероятность неудачных итераций разработки.

Снижение рисков и затрат: Ошибки и недоразумения на этапе определения требований могут привести к дорогостоящим изменениям и переделкам на более поздних этапах разработки. Правильное определение позволяет избежать подобных ситуаций, что экономит как время, так и деньги.

Фокус на существенном: Хорошо определенные требования помогают избегать излишней функциональности, которая не является приоритетной для заказчика. Это позволяет сосредотачиваться на разработке и тестировании тех аспектов, которые действительно важны для пользователей.

Обеспечение качества: Правильно определенные требования становятся основой для планирования тестирования и проверки корректности разработанного продукта. Это способствует созданию ПО, которое соответствует ожиданиям пользователей и заказчика.

Прозрачность коммуникации: Определение требований способствует ясной коммуникации между всеми участниками проекта. Это помогает предотвратить недопонимания, которые могут возникнуть из-за неясных или противоречивых требований.

Основа для дальнейшего проектирования и разработки: Правильно определенные требования становятся основой для проектирования архитектуры, разработки кода, создания интерфейсов и тестирования. Если требования не определены правильно, все последующие этапы могут пострадать.

Следует отметить, что определение требований - это итеративный процесс, который может изменяться по мере уточнения понимания заказчика и разработчиков. Грамотное взаимодействие с заказчиком, анализ потребностей пользователей и использование подходящих методик сбора информации обеспечивают успешное определение требований и, как следствие, успешное выполнение проекта.

Основные понятия: требования, функциональные и нефункциональные требования.

Требования - это описание того, что должно быть реализовано в программном продукте, чтобы удовлетворить потребности и ожидания заказчика или пользователей. Требования определяют функциональность, характеристики и ограничения продукта.

Существует два основных типа требований:

1.Функциональные требования.

2.Нефункциональные требования.

Функциональные требования: Эти требования описывают, как продукт должен выполнять определенные функции, задачи или операции. Они охватывают функциональность, которую продукт должен обеспечивать, чтобы достичь целей заказчика. Примеры функциональных требований: "Пользователь должен иметь возможность авторизоваться в системе", "Приложение должно позволять пользователям создавать и редактировать посты".

Нефункциональные требования: Эти требования определяют характеристики и ограничения продукта, которые не связаны напрямую с его функциональностью, но влияют на общее качество и производительность. К нефункциональным требованиям относятся аспекты, такие как производительность, безопасность, надежность, масштабируемость и т. д. Примеры нефункциональных требований: "Система должна поддерживать одновременное обслуживание не менее 1000 пользователей", "Время отклика системы не должно превышать 2 секунды".

Основные понятия, связанные с требованиями:

- **Заказчик (Клиент):** Организация или лицо, для которых разрабатывается продукт. Они определяют требования, основываясь на своих потребностях и бизнес-целях.
- **Пользователи:** Люди, которые будут использовать программный продукт. Их потребности и ожидания определяют функциональные требования.
- **Аналитики:** Специалисты, которые собирают и анализируют требования. Они работают с заказчиком и пользователями, чтобы понять их потребности и преобразовать их в конкретные требования.

- **Требования пользователей:** Это требования, выраженные пользователями в их естественном языке. Они часто требуют дополнительного анализа и уточнения, чтобы превратить их в конкретные функциональные и нефункциональные требования.
- **Требования системы:** Это конкретные функциональные и нефункциональные требования, описывающие, каким образом продукт должен работать.
- **Спецификации:** Формальные документы, в которых описываются требования. Они могут включать текстовые описания, диаграммы, таблицы и другие средства для ясного изложения требований.

Правильное понимание и правильная формулировка требований являются ключевыми факторами успешной разработки программного продукта, поскольку они служат основой для всех последующих этапов процесса разработки.

Идентификация и устранение неоднозначностей и противоречий.

Идентификация и устранение неоднозначностей и противоречий являются важными этапами в процессе определения требований при разработке программного обеспечения. Неоднозначности и противоречия могут возникнуть из-за недостаточной ясности или точности в формулировке требований, различных интерпретаций со стороны разработчиков и заказчиков, а также из-за изменений в проекте и окружающей среде. Вот как этот процесс может выглядеть подробнее:

Этапы процесса определения требований.

Процесс определения требований - это критически важный этап в разработке программного продукта, который включает в себя несколько этапов и подходов для выявления, анализа и документирования требований заказчика и пользователей. Вот основные этапы этого процесса:

Идентификация стейкхолдеров:

- **Стейкхолдеры** - это все лица и группы, которые имеют интерес и влияние на проект. Это может быть заказчик, пользователи, менеджеры, разработчики, тестировщики и другие.
- Цель этого этапа - точно определить, кто будет использовать продукт и какие у них могут быть потребности.
- Интервью с представителями различных групп стейкхолдеров позволяет выявить разнообразные ожидания и требования.

Сбор информации о потребностях пользователей:

- Здесь исследуется, что именно пользователи ожидают от продукта, как они планируют его использовать и какие функциональные возможности им необходимы.
- Различные методы используются для сбора информации: интервью, наблюдение, анкетирование, прототипирование.
- Важно активно слушать и взаимодействовать с пользователями, чтобы точно понять их потребности.

Анализ и классификация требований:

- На этом этапе собранные требования анализируются и структурируются.
- Функциональные требования - описывают, что продукт должен делать. Например, какие функции и операции должны быть включены.
- Нефункциональные требования - описывают ограничения, условия, поддержку и другие аспекты. Например, производительность, безопасность, доступность и т.д.

Документирование требований:

- На этом этапе требования формализуются и документируются для дальнейшей передачи команде разработки.
- Документы с требованиями могут включать текстовые описания, диаграммы, таблицы, примеры использования.
- Цель - создать точное и однозначное описание требований, чтобы избежать недоразумений и ошибок в процессе разработки.

Проверка и утверждение:

Предоставление документа с требованиями заказчику и пользователям для проверки и утверждения.

Согласование с заказчиком:

Обеспечение понимания и одобрения требований заказчиком и пользователями.

Устранение возможных недоразумений и противоречий с помощью обратной связи.

Верификация и валидация:

Верификация - проверка того, что все требования были реализованы.

Валидация - убеждение, что продукт действительно удовлетворяет потребности заказчика.

Процесс определения требований должен быть взаимодействующим и итеративным, чтобы убедиться, что все стороны понимают и согласны с требованиями. Правильное определение требований помогает избежать ошибок и несоответствий в последующих этапах разработки и обеспечивает достижение бизнес-целей проекта.

Методы сбора информации о требованиях:

Интервью с заказчиком и пользователями:

- Этот метод включает в себя беседы с представителями заказчика и будущими пользователями для получения информации о их потребностях, ожиданиях и желаниях относительно продукта.
- Интервью позволяют уточнить детали, выявить скрытые требования и понять, как продукт будет использоваться в реальной жизни.
- Хорошая подготовка вопросов и активное слушание помогают получить ценные данные.

Наблюдение за рабочим процессом:

- Этот метод предполагает наблюдение за пользователями в их ежедневной деятельности, чтобы понять, как они работают, какие задачи выполняют и каким образом продукт может улучшить их рабочий процесс.
- Наблюдение позволяет выявить реальные потребности и проблемы, которые могут быть учтены в требованиях.

Анкетирование:

- В данном методе пользователи заполняют анкеты или опросники, где отвечают на вопросы о своих предпочтениях, потребностях и ожиданиях относительно продукта.
- Анкетирование может охватить большее количество пользователей и помочь собрать статистические данные о требованиях.

Прототипирование:

- Прототипирование включает создание временного образца продукта, который демонстрирует основные функции и интерфейс.
- Прототип помогает пользователям и заказчику лучше понять, как будет работать готовый продукт, и дает возможность выявить недоразумения или несоответствия требованиям на ранних стадиях.

Анализ требований.

Анализ требований – это этап процесса разработки программного обеспечения, на котором требования, собранные на предыдущих этапах, подвергаются более детальному и структурированному анализу для того, чтобы обеспечить их полноту, корректность, непротиворечивость и понимание.

В рамках анализа требований выполняются следующие шаги:

Идентификация и устранение неоднозначностей и противоречий:

- Подробный анализ требований может выявить неоднозначности (различные интерпретации) и противоречия (когда одно требование противоречит другому).
- Аналитик и команда разработки работают с заказчиком и пользователями, чтобы разъяснить все неясные моменты и устранить противоречия.

Определение приоритетов:

- На этом этапе требования оцениваются и ранжируются по их важности и необходимости.
- Это позволяет определить, какие требования критически важны для первой версии продукта, а какие могут быть отложены на будущие релизы.
- Определение приоритетов помогает сосредотачиваться на наиболее значимых функциональностях и обеспечивать максимальное удовлетворение потребностей пользователей.

3. Введение в UML.

Основы UML

UML, или Unified Modeling Language (Единый язык моделирования), это стандартный графический язык, используемый для визуализации, спецификации, конструирования и документирования программных систем и процессов разработки. UML предоставляет набор нотаций и графических элементов, которые позволяют разработчикам и аналитикам лучше понимать, проектировать и коммуницировать сложные концепции в рамках проектов.

Основной целью UML является облегчение обмена информацией между участниками проекта, создание общего языка для общения, а также повышение уровня абстракции, что позволяет увидеть общую картину и детали проекта. В разработке ПО UML облегчает создание архитектуры, определение требований, проектирование, документирование и тестирование.

Преимущества использования UML:

1.Стандартизация: UML является международным стандартом, что позволяет участникам проекта понимать и коммуницировать друг с другом, даже если они говорят на разных языках.

2.Визуализация: UML предоставляет графические элементы и диаграммы, которые делают сложные концепции более понятными и визуально привлекательными.

3.Документирование: С помощью UML можно создавать детальную документацию, которая описывает различные аспекты проекта, что облегчает понимание для будущих разработчиков и аналитиков.

4.Абстракция: UML позволяет создавать абстрактные модели, которые помогают увидеть общую структуру проекта, а также сосредоточиться на ключевых аспектах.

5.Улучшение коммуникации: Участники проекта могут использовать UML-диаграммы для обсуждения, обмена идеями и принятия решений на более высоком уровне абстракции.

Основные виды диаграмм UML:

1.Диаграммы классов: Показывают структуру классов, их атрибутов, методов и отношений между ними.*

2.Диаграммы вариантов использования (Use Case Diagrams)(Диаграмма Прецедентов): Моделируют взаимодействие между актерами (пользователями) и системой, описывая сценарии использования.*

3.Диаграммы последовательности: Демонстрируют последовательность взаимодействия между объектами или компонентами в определенных сценариях.*

4.Диаграммы активности: Иллюстрируют потоки управления и данных в рамках различных процессов или операций.

5.Диаграммы состояний: Описывают различные состояния и переходы между ними для конкретных объектов.

6.Диаграммы компонентов и развертывания: Показывают структуру компонентов системы и их развертывание на физических устройствах.

Основные понятия UML

1.Класс (Class): Описывает абстрактный шаблон для создания объектов. Класс включает в себя атрибуты (переменные) и методы (функции), которые определяют его поведение и состояние.

2.Объект (Object): Экземпляр класса, существующий во время выполнения программы.

3.Атрибут (Attribute): Переменная, хранящая информацию о состоянии объекта.

4.Метод (Method): Функция, выполняющая определенные действия с объектом.

5.Связь (Association): Отношение между классами или объектами, показывающее, как они связаны и каким образом могут взаимодействовать.

6.Агрегация (Aggregation): Отношение, при котором один объект (часть) может быть частью другого объекта (целого). Например, класс "Студент" может быть частью класса "Университет".

7.Композиция (Composition): Отношение, более строгое, чем агрегация. Здесь объекты неразрывно связаны друг с другом, и один объект не может существовать без другого. Например, класс "Человек" состоит из классов "Голова", "Тело" и т.д.

8.Наследование (Inheritance): Отношение между классами, при котором один класс (подкласс) наследует свойства и методы другого класса (суперкласса).

9.Интерфейс (Interface): Спецификация методов, которые класс должен реализовать, чтобы соответствовать данному интерфейсу.

Диаграммы вариантов использования (или диаграммы прецедентов) являются частью нотации UML и служат для визуализации взаимодействия между различными акторами (пользователями, внешними системами) и системой. Эти диаграммы помогают понять, как различные сценарии использования будут взаимодействовать с системой и как она будет отвечать на запросы акторов.

Главные элементы диаграммы вариантов использования:

1. **Акторы (Actors):** Пользователи, системы или другие сущности, которые взаимодействуют с системой.
2. **Варианты использования (Use Cases):** Описывают конкретные сценарии взаимодействия между акторами и системой.

Процесс построения диаграммы вариантов использования включает:

- Идентификацию акторов и их ролей.
- Определение основных сценариев использования, которые покрывают основные функциональные возможности системы.
- Выделение вариантов использования, описывающих дополнительные и альтернативные сценарии.
- Создание диаграммы, на которой отображаются связи между акторами и вариантами использования.

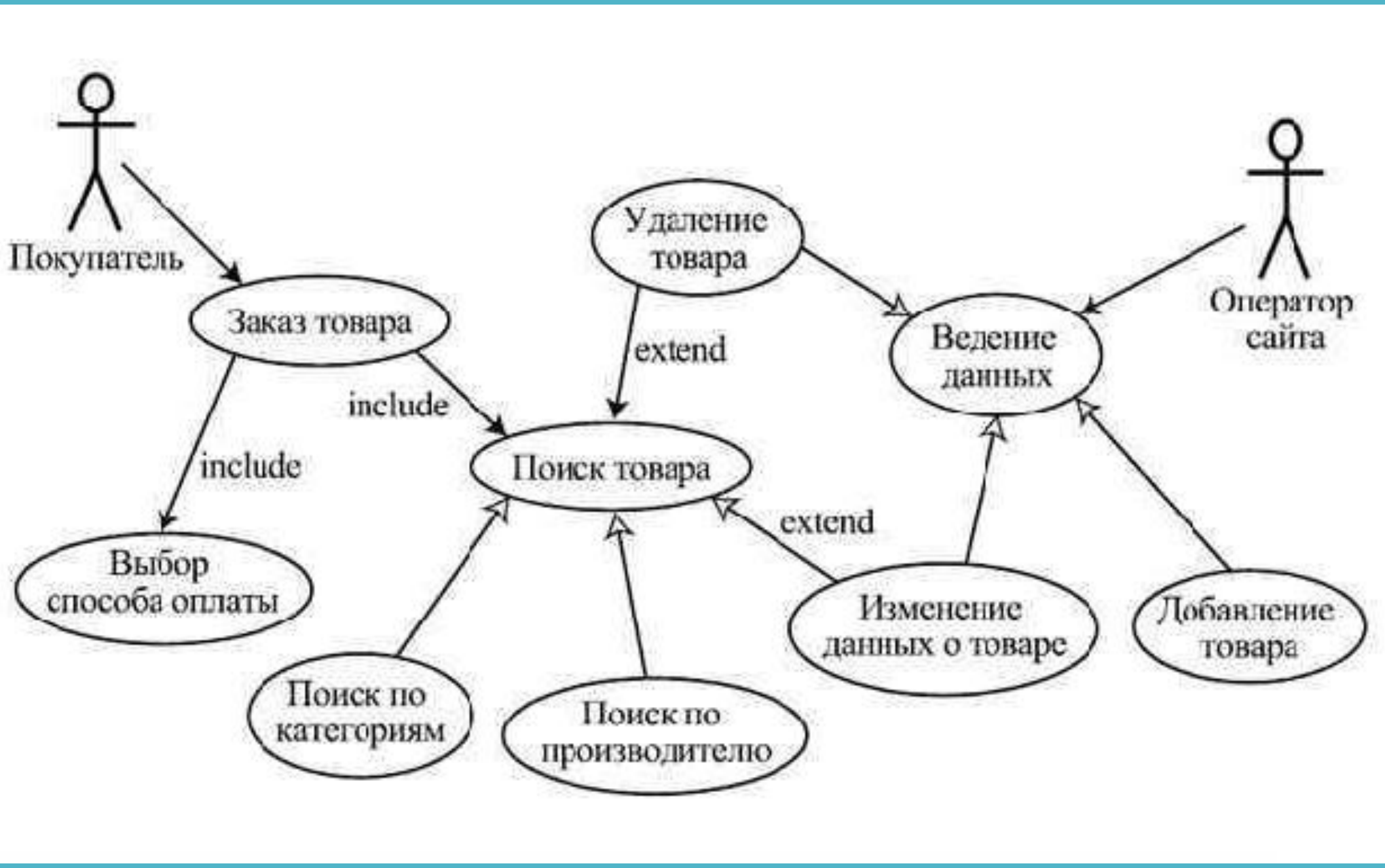
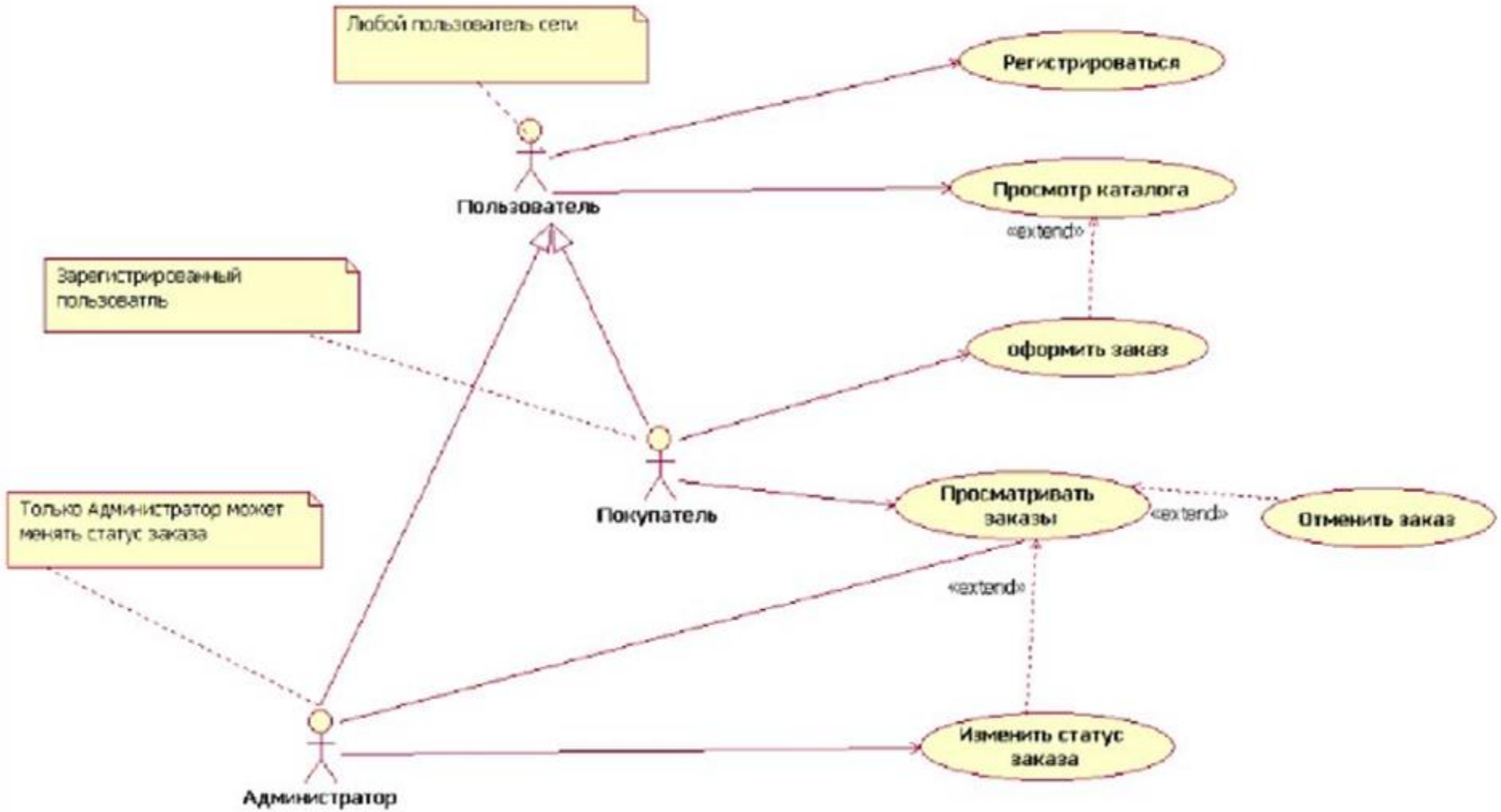

Покупатель
Заказ товара
include
include
Выбор
способа оплаты
Поиск товара
Удаление
товара
extend
Ведение
данных
Оператор
сайта
extend
Изменение
данных о товаре
Лобавление
товара
Поиск по
категориям
Поиск по
производителю

Диаграмма вариантов использования

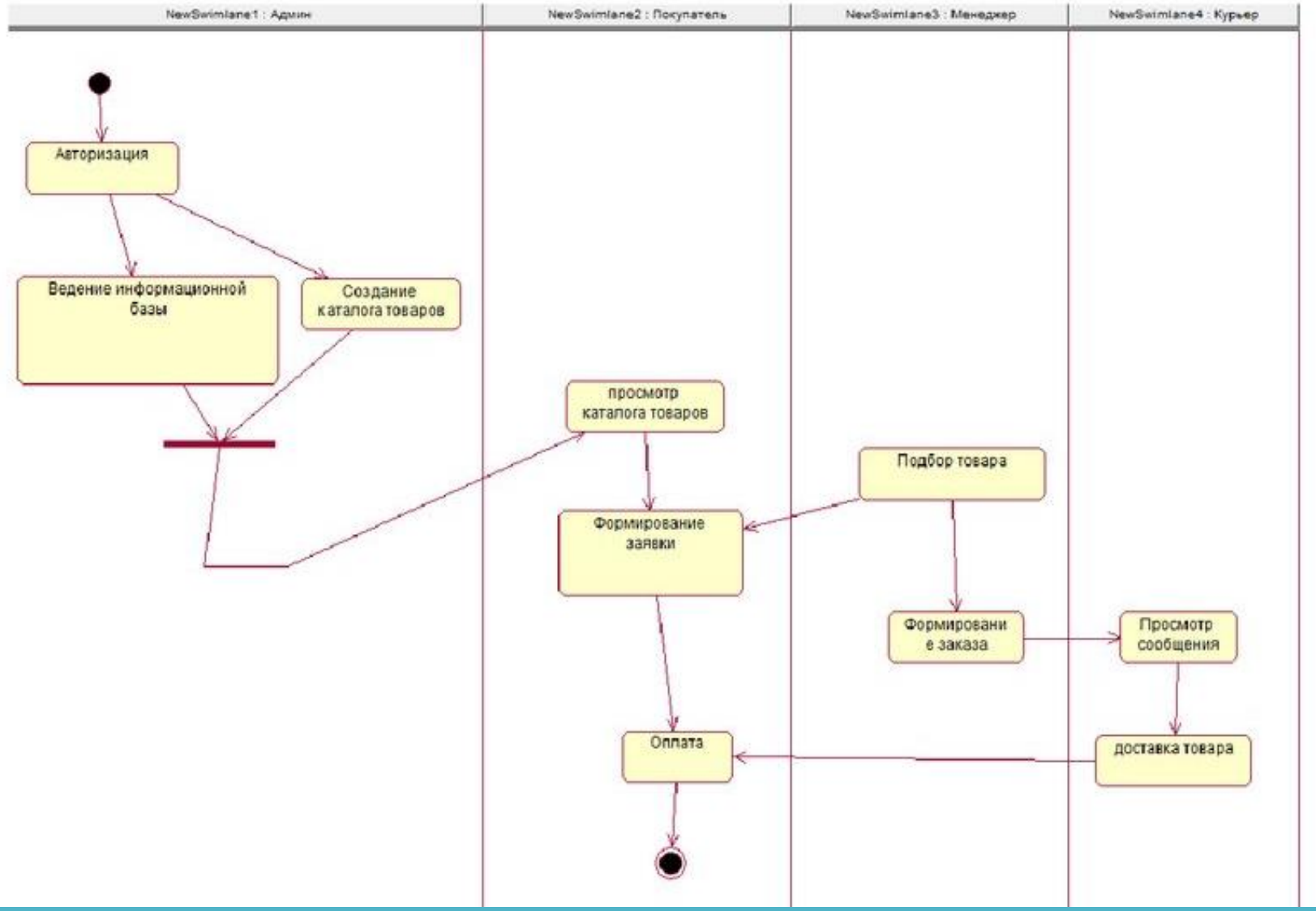


Диаграммы активностей (Activity Diagrams): Позволяют описать бизнес-процессы и последовательность действий в различных сценариях.

Диаграмма активности (деятельности, activity diagram)

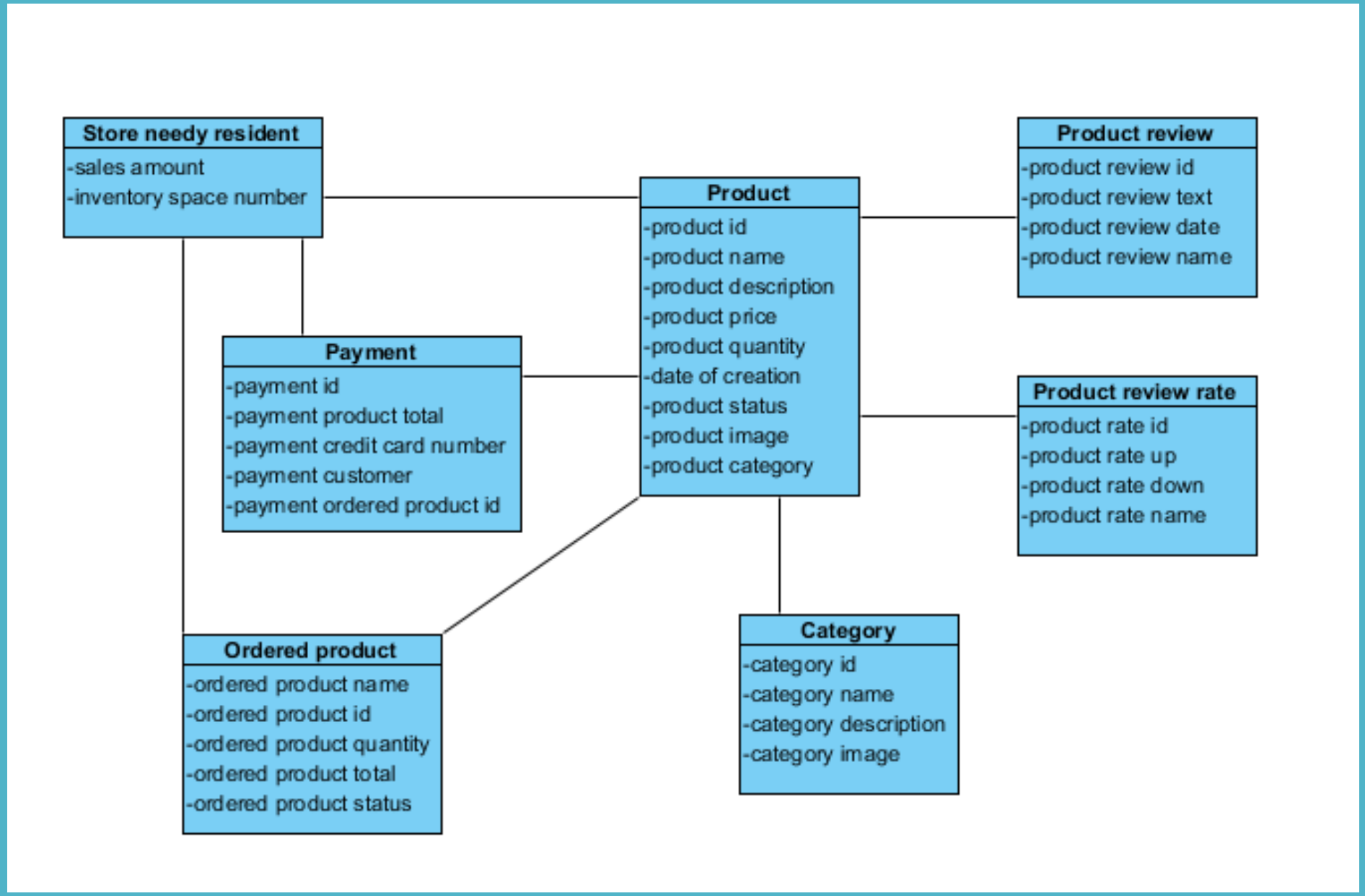
Диаграммы последовательностей (Sequence Diagrams): Отображают взаимодействие между различными объектами и компонентами системы в определенном временном порядке.

Диаграмма последовательности



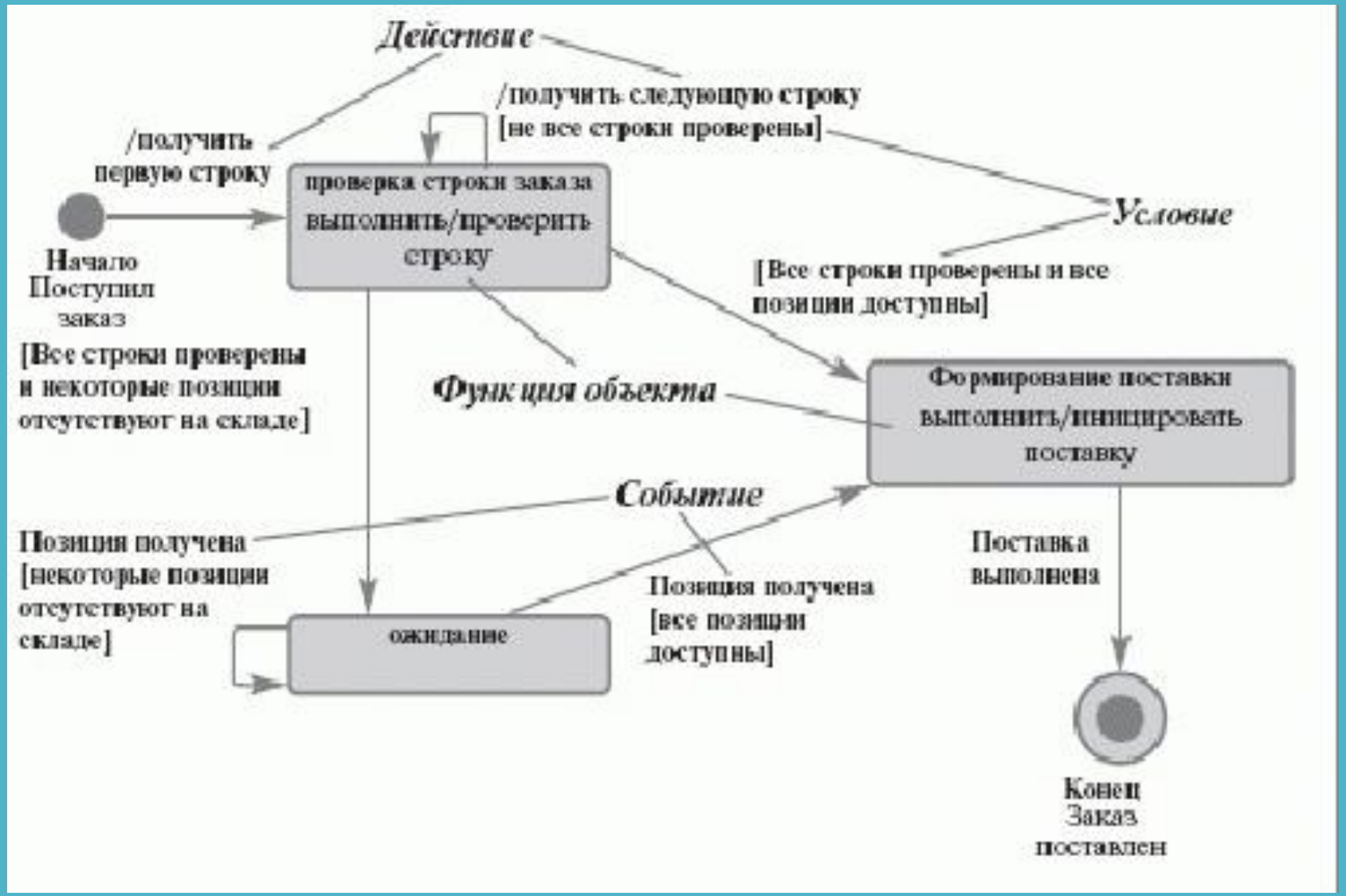
Диаграммы классов (Class Diagrams): Позволяют определить структуру данных и отношения между классами.

Диаграмма классов (Class Diagrams)

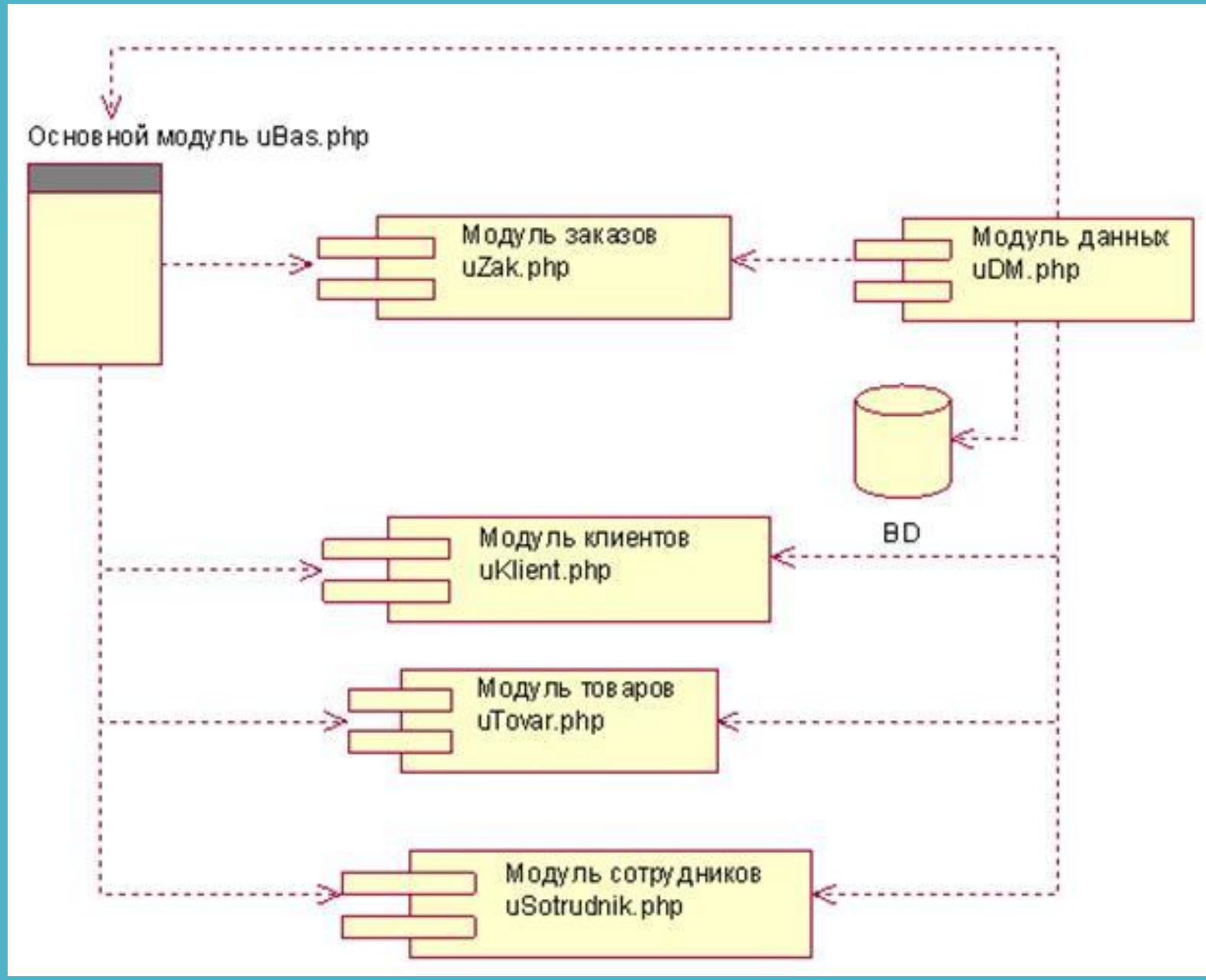


Диаграммы состояний (State Diagrams): Показывают различные состояния, в которых может находиться объект или компонент.

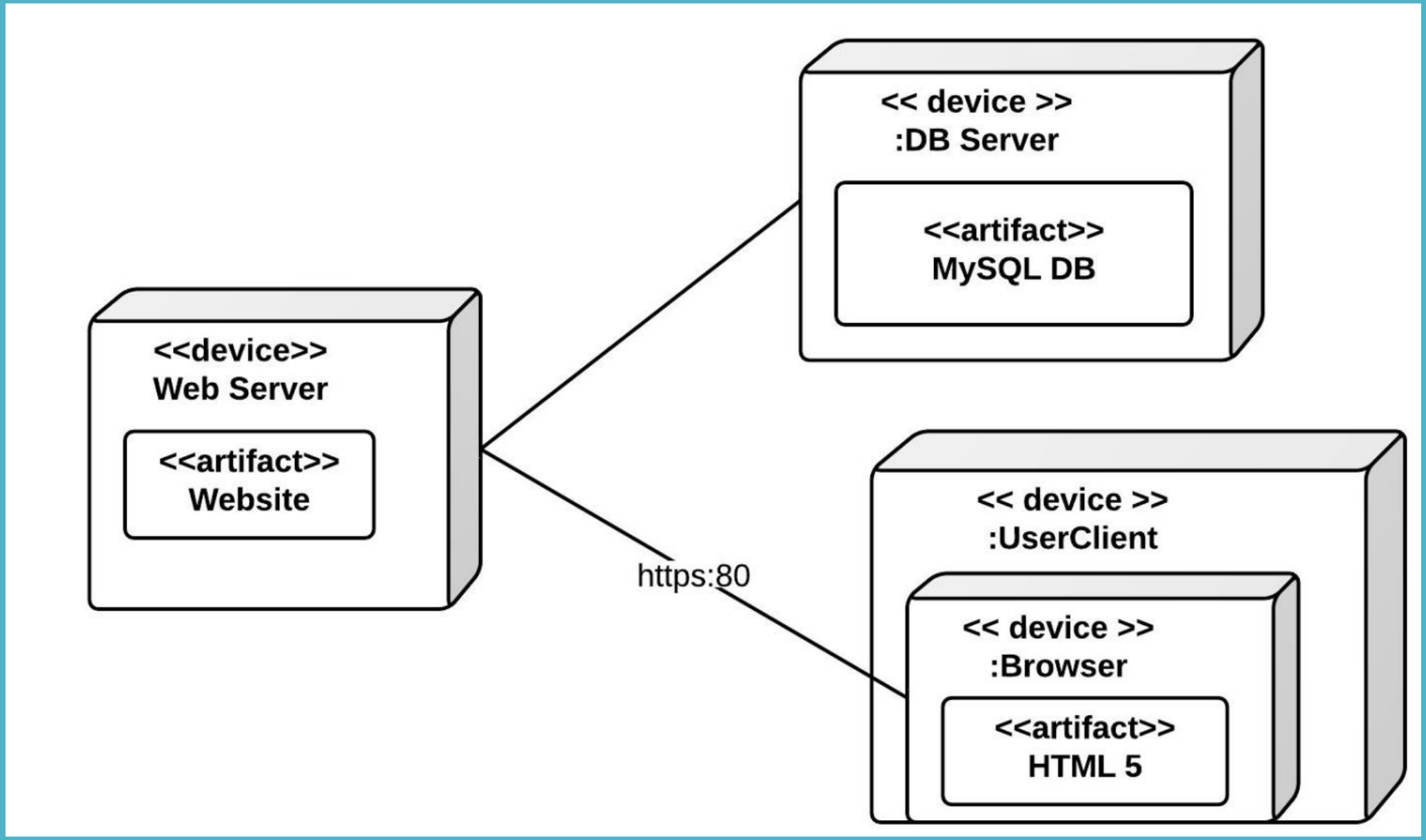
Диаграмма состояний (State Diagrams)



Диаграммы компонентов (Component Diagrams):
Описывают структуру системы и ее компонентов.


Основной модуль uBas.php
Модуль заказов
uZak.php
Модуль данных
uDM.php
BD
Модуль клиентов
uKlient.php
Модуль товаров
uTovar.php
Модуль сотрудников
uSotrudnik.php

Диаграммы развертывания (Deployment Diagrams): Позволяют описать размещение компонентов системы на аппаратных ресурсах.


<< device >>
:DB Server
<<artifact>>
MySQL DB
<<device>>
Web Server
<<artifact>>
Website
https:80
<< device >>
:UserClient
<< device >>
:Browser
<<artifact>>
HTML 5

Использование нотации UML позволяет более точно и структурированно описать требования и визуализировать их взаимосвязи, что упрощает понимание для разработчиков, аналитиков и других участников проекта.

Структурный анализ — это метод анализа и проектирования, который позволяет разбить сложную систему на более простые составные части и определить их взаимосвязи. В контексте анализа требований, структурный анализ может использоваться для декомпозиции требований на более конкретные и понятные компоненты.

Пример применения структурного анализа к анализу требований для онлайн-магазина:

Исходное требование: Реализовать возможность оформления заказа для клиентов.

Декомпозиция с использованием структурного анализа:

1.Процесс оформления заказа:

1. Ввод товаров в корзину.
2. Просмотр содержимого корзины.
3. Внесение информации о доставке и оплате.
4. Подтверждение заказа.

2.Ввод товаров в корзину:

1. Поиск товаров.
2. Добавление товаров в корзину.
3. Удаление товаров из корзины.

3.Просмотр содержимого корзины:

1. Отображение списка товаров.
2. Подсчет общей стоимости.

4. **Внесение информации о доставке и оплате:**
    1. Выбор способа доставки.
    2. Ввод адреса доставки.
    3. Выбор способа оплаты.
    4. Ввод информации для оплаты.
5. **Подтверждение заказа:**
    1. Проверка деталей заказа.
    2. Подтверждение заказа.
    3. Генерация уведомления о заказе.

Этот пример показывает, как структурный анализ позволяет разбить сложное требование на более простые и понятные этапы. Каждый этап может далее подвергаться более детальному анализу, чтобы определить конкретные действия и компоненты, необходимые для его реализации.

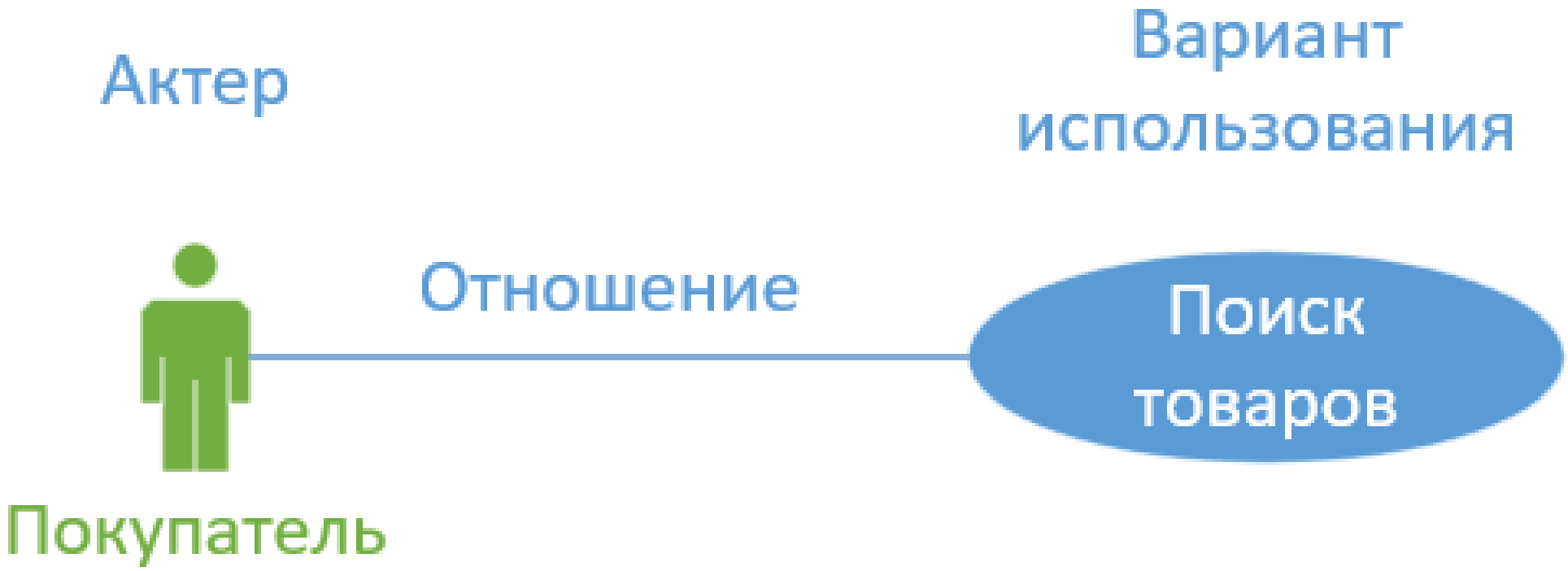
Построение Диаграммы вариантов использования (Use Case Diagram)

Важные компоненты диаграммы вариантов использования:

1. **Актеры (Actors):** Это внешние сущности, которые взаимодействуют с системой. Актеры могут быть пользователями, другими системами или даже временными процессами.

2. **Варианты использования (Use Cases):** Это конкретные действия или сценарии, которые актеры выполняют в системе. Они представляют собой функциональные требования к системе и описывают, как система должна взаимодействовать с актерами.

3. **Отношения между актерами и вариантами использования:** Отношения показывают, какие варианты использования доступны для каждого актера. Например, один актер может взаимодействовать с несколькими вариантами использования.


Актер
Вариант использования
Отношение
Поиск товаров
Покупатель

Процесс создания диаграммы вариантов использования:

1.Идентификация актеров: Определите, какие сущности будут взаимодействовать с вашей системой. Это могут быть конечные пользователи, роли, другие системы и т.д.

2.Определение вариантов использования: Для каждого актера определите, какие действия он может выполнять в системе. Это могут быть типичные задачи, сценарии использования или функциональные требования.

3.Создание диаграммы: Расположите актеров и варианты использования на диаграмме. Связи между актерами и вариантами использования показывают, какие действия могут быть выполнены актерами.

4.Добавление отношений: Дополните диаграмму отношениями, чтобы показать, какие варианты использования доступны для каждого актера.

5.Дополнительные детали: Вы можете добавить дополнительные аннотации, описания и сценарии для уточнения деталей каждого варианта использования.

Типы отношений:

Ассоциация (Association): Это отношение показывает связь между двумя классами. Например, "Заказ" ассоциирован с "Товарами".

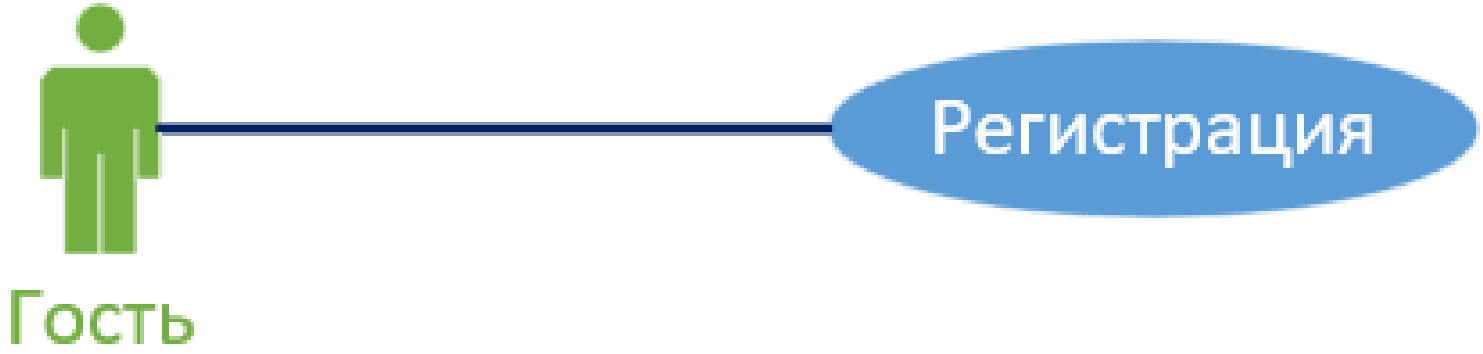
Зависимость (Dependency): в общем случае пунктирная линия с V-образной стрелкой. Для диаграммы вариантов использования выделяют различные виды зависимостей: отношение включения и отношение расширения.

Включение (Include): Показывает, что один вариант использования включает в себя другой вариант. Например, "Оформление заказа" может включать в себя "Выбор товаров". Отношение включения обозначает, что элемент **обязательно** включается в состав другого элемента

Расширение (Extend): Демонстрирует, что один вариант использования может быть расширен другим вариантом. Например, "Оформление заказа" может быть расширено вариантом "Применение скидки". Отношение расширения - это **выборочное** (необязательное) отношение включения.

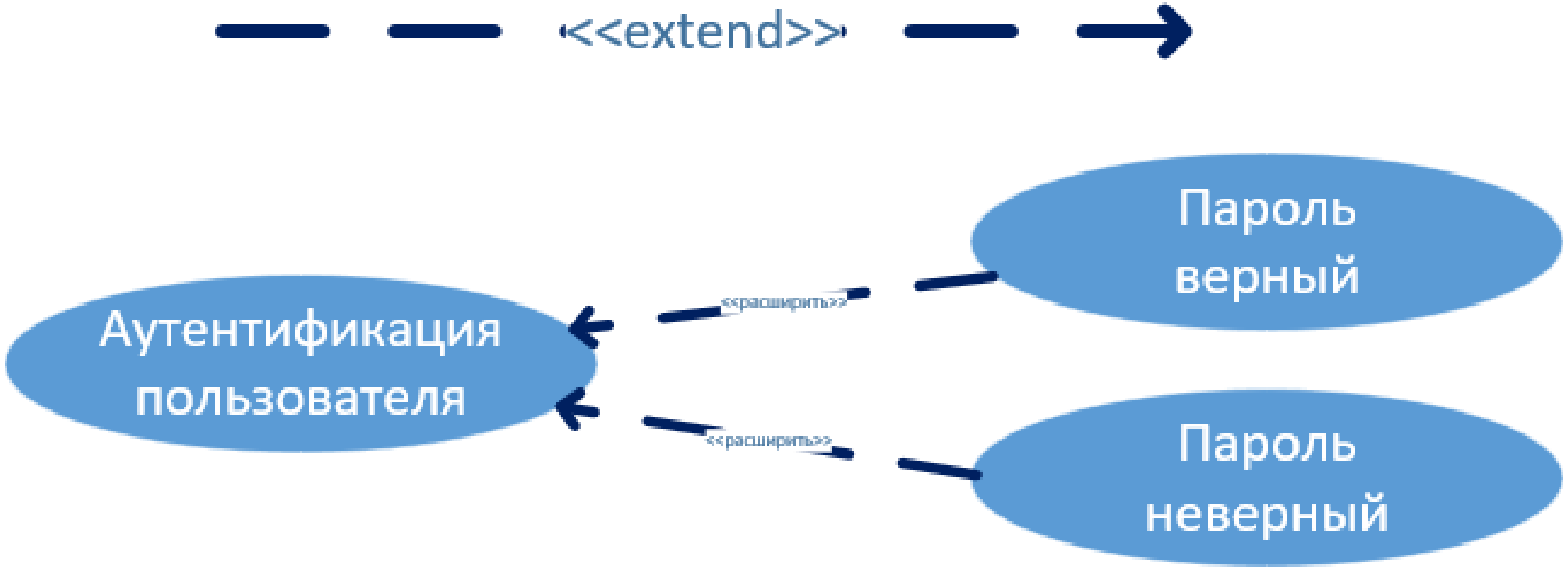
Наследование (Обобщение): Отражает отношение "является" между акторами. Например, "Покупатель" и «Продавец" является "Пользователями".

Ассоциация



Актеры и прецеденты связываются посредством сообщений или вызовов

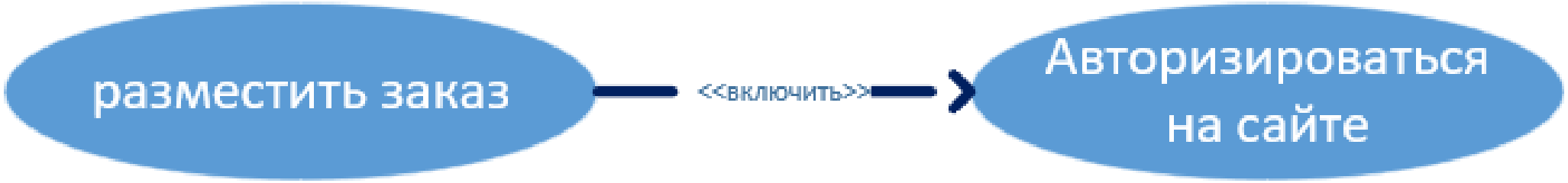
Расширение



Базовый прецедент расширяется еще несколькими

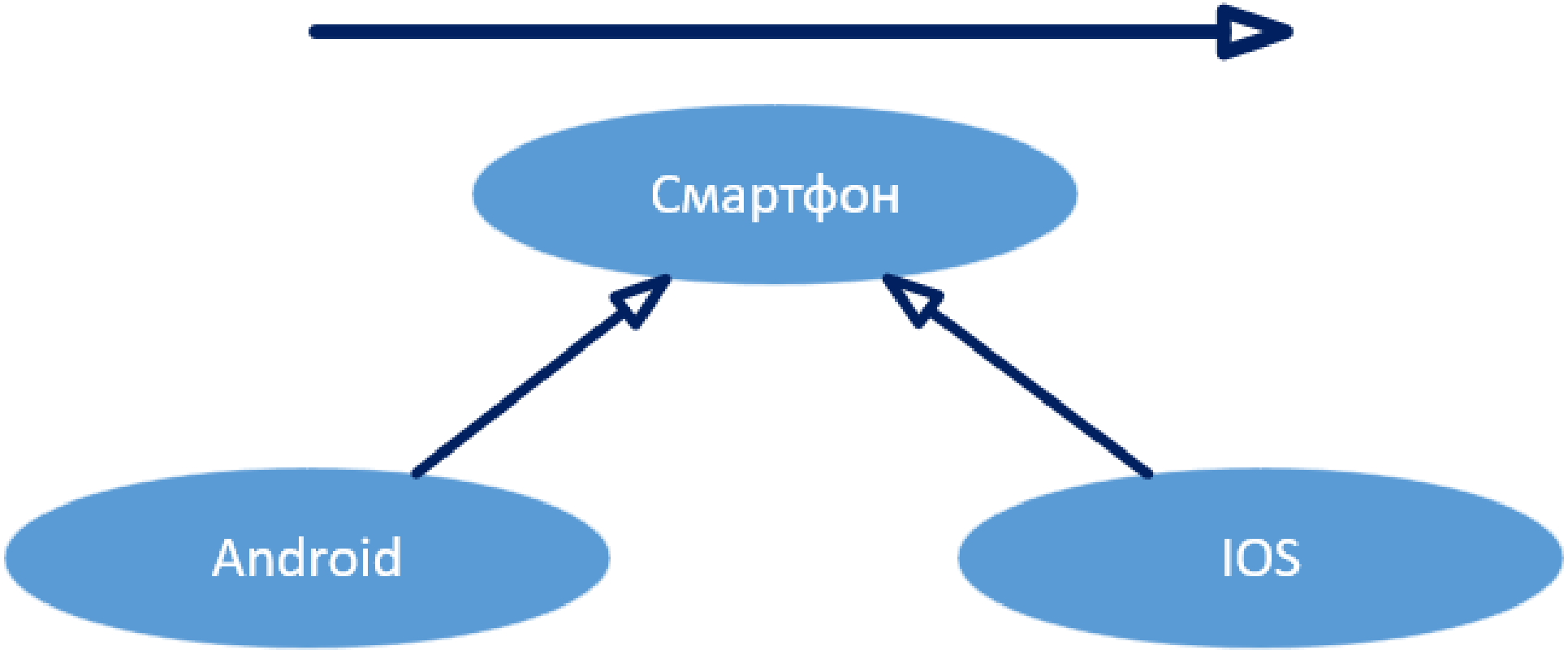
Включение

<<include>>



Один прецедент использует функционал другого

Наследование(обобщение)



Отношение Родитель – Ребенок (Наследник)

Пример UML-диаграммы для интернет-магазина


Интернет-магазин
Обработать заказ
«расширить»
Выполнить заказ
«расширить»
Отменить заказ
Продавец
Обобщение
Войти в систему
Пользователь
Обобщение
просмотр товаров
«расширить»
Добавить в корзину
«расширить»
Оформить заказ
«расширить»
«включить»
Поиск товаров
«расширить»
Ввести номер карты
Покупатель

Применение диаграмм вариантов использования для моделирования требований

Диаграмма вариантов использования (Use Case Diagram) в UML (Unified Modeling Language) - это графический инструмент для моделирования функциональности системы с точки зрения взаимодействия между акторами (пользователями, внешними системами или другими ролями) и вариантами использования (сценариями) системы. Эта диаграмма позволяет визуализировать, как разные акторы будут взаимодействовать с системой для достижения конкретных целей.

1.Акторы (Actors): Представляют роли или сущности, которые взаимодействуют с системой. Акторы могут быть конечными пользователями, внешними системами, а также другими частями окружения (Существительное).

2.Варианты использования (Прецедент) (Use Cases): Это сценарии взаимодействия между акторами и системой. Варианты использования описывают конкретные действия и события, которые происходят в системе в процессе выполнения задач(глагол, действие).

3.Отношения:

1. **Ассоциация (Association):** Определяет связь между актором и вариантом использования. Показывает, какой актор участвует в выполнении конкретного варианта использования (связывает актора и прецедент).
2. **Отношение включения (Include):** Показывает, что один вариант использования может включать в себя другой вариант использования (только между прецедентами).
3. **Отношение расширения (наследования) (Extend):** Показывает, что один вариант использования может расширяться другим вариантом использования в определенных ситуациях (только между прецедентами).
4. **Обобщение (Наследование) (Generalization)**: Это отношение используется для обозначения общих характеристик между прецедентами и акторами. (только между однотипными сущностями, т.е. от актора к актору или от прецедента к прецеденту)

Диаграммы вариантов использования обычно используются на начальных этапах проектирования для уточнения требований и понимания функциональности системы. Они помогают командам разработчиков и заказчикам лучше оценить, как система будет взаимодействовать с пользователями и другими системами.

Диаграммы вариантов использования выполняют важные роли в моделировании требований:

1.Визуализация: Они предоставляют понятное и наглядное представление о том, как система будет взаимодействовать с различными акторами. Это помогает стейкхолдерам, включая разработчиков и заказчиков, лучше понять функциональные потребности и ожидания пользователей.

2.Анализ: Диаграммы вариантов использования позволяют выявить сценарии, которые могут быть недостаточно освещены или тщательно проработаны. Анализ диаграмм помогает выявить потенциальные противоречия, ошибки и неоднозначности в требованиях.

3.Коммуникация: Они служат эффективным средством коммуникации между участниками проекта. Диаграммы вариантов использования позволяют разработчикам, дизайнерам, тестировщикам и заказчикам общаться на общем языке, что упрощает взаимопонимание и согласование требований.

4.Планирование: Опираясь на диаграммы вариантов использования, команда может лучше планировать разработку, тестирование и другие этапы проекта. Это помогает оценить объем работы, необходимый для реализации конкретных функциональных возможностей.

Принципы построения диаграммы вариантов использования :

Идентификация акторов и вариантов использования: Сначала определите, кто является акторами (пользователи, внешние системы) и какие действия они выполняют в системе (варианты использования).

Четкость и простота: Диаграмма должна быть понятной и легко читаемой. Избегайте перегруженности информацией и сложных связей.

Соответствие действительности: Диаграмма должна точно отражать функциональные возможности системы и реальные сценарии использования.

Именование акторов и вариантов использования: Дайте акторам и вариантам использования описательные и понятные имена.

Использование отношений: Используйте отношения между акторами и вариантами использования для указания взаимосвязей. Отношения должны быть логичными и адекватными.

Группировка вариантов использования: При необходимости, группируйте схожие варианты использования в подходящие категории для более логичной организации.

Применение стандартных символов: Используйте стандартные символы для акторов и вариантов использования, чтобы сделать диаграмму более понятной и согласованной.

Использование расширенных иключений и включений: Если сценарии имеют ветвления или дополнительные действия, используйте расширенные иключения (Extensions) и включения (Inclusions), чтобы показать эти дополнительные шаги.

Сгруппированные элементы: Если диаграмма становится сложной, можно использовать группы и контейнеры, чтобы сгруппировать элементы в логические блоки.

Понятные иерархии: Если у вас есть иерархия акторов или вариантов использования, используйте иерархические отношения для ее отображения.

Избегание излишних деталей: Диаграмма должна демонстрировать общую структуру взаимодействия, избегайте слишком детального описания каждого шага.

Основные этапы построения диаграммы вариантов использования:

Идентификация акторов и вариантов использования:

1. Определите всех акторов - пользователей, внешние системы или другие сущности, которые взаимодействуют с вашей системой.
2. Определите варианты использования - действия, которые акторы могут выполнять в системе.

Создание диаграммы:

1. Выберите инструмент для создания диаграммы (например, CASE-среду, онлайн-приложение или программу для рисования).
2. Создайте пустую диаграмму вариантов использования.

Добавление акторов и вариантов использования:

1. Разместите акторов на диаграмме. Это можно делать с помощью иконок, представляющих акторов.
2. Добавьте варианты использования на диаграмму, размещая их рядом с соответствующими акторами.

Установление отношений:

1. Добавьте линии (ассоциации) между акторами и вариантами использования для показа связей.
2. Определите характер связи: связь между актором и вариантом использования может быть более или менее тесной.

Добавление расширений и включений (при необходимости):

1. Для детализации сценариев добавьте расширенные иключения (Extensions) и включения (Inclusions) между вариантами использования.

Рассмотрение взаимодействий:

1. Проверьте диаграмму на логичность взаимодействия между акторами и вариантами использования.
2. Убедитесь, что диаграмма отражает сценарии использования системы.

Именование элементов:

1. Дайте акторам и вариантам использования описательные и понятные имена.

Проверка и анализ:

1. Проверьте диаграмму на предмет ошибок, неоднозначностей и неполадок.
2. Проведите анализ, чтобы убедиться, что все важные сценарии использования учтены.

Документация:

1. Добавьте дополнительные описания к акторам и вариантам использования, если необходимо.

Совместное обсуждение:

1. Проведите совместное обсуждение диаграммы с участием разработчиков, аналитиков и других заинтересованных сторон.

Обновление и доработка:

1. Внесите необходимые изменения на основе обсуждения и анализа.
2. Повторите процесс до тех пор, пока диаграмма не будет точно отражать требования.

Вставка в документацию:

1. Вставьте диаграмму в общий документ, описывающий требования к системе.

Пример. Интернет-магазин

Задача: Разработать диаграмму вариантов использования для Интернет-магазина.

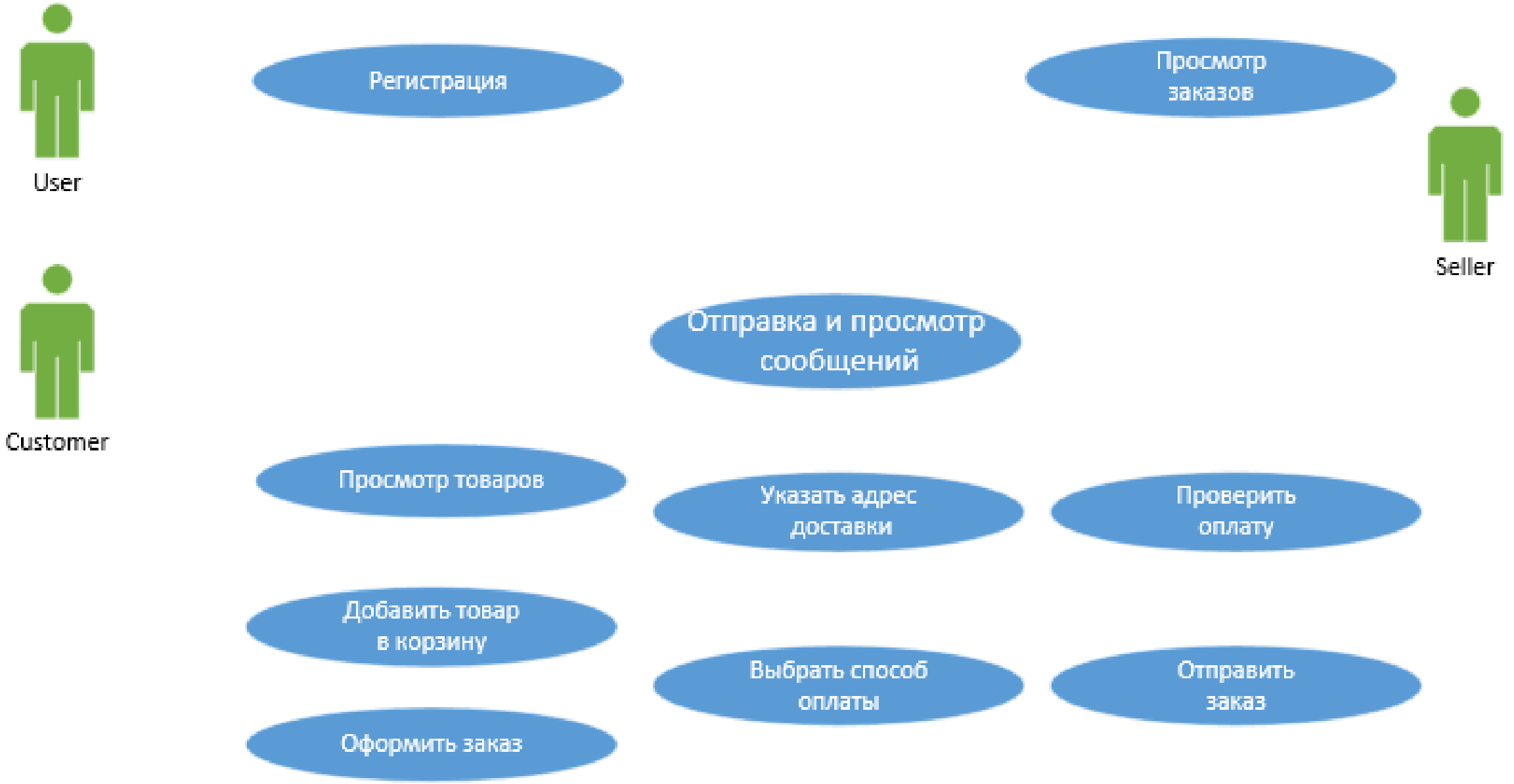
Акторы:

- Незарегистрированный пользователь (User)
- Покупатель (Customer)
- Продавец (Seller)

Пример 1. Интернет-магазин

Варианты использования:

- Просмотр каталога товаров
- Регистрация пользователя
- Добавить товар в корзину
- Оформить заказ
- Выбрать способ оплаты
- Указать адрес доставки
- Просмотр заказов
- Проверить оплату
- Отправить заказ
- Отправка и просмотр сообщений


User
Customer
Seller
Регистрация
Просмотр заказов
Отправка и просмотр сообщений
Просмотр товаров
Указать адрес доставки
Проверить оплату
Добавить товар в корзину
Выбрать способ оплаты
Отправить заказ
Оформить заказ


User
Регистрация
Просмотр товаров
Указать адрес доставки
Просмотр заказов
<<включить>>
Добавить товар в корзину
Проверить оплату
<<включить>>
<<включить>>
Оформить заказ
Выбрать способ оплаты
Отправить заказ
Customer
Отправка и просмотр сообщений
Seller

Пример 2. Телеграм-бота для погоды

Задача: Разработать диаграмму вариантов использования для погодного телеграм-бота.

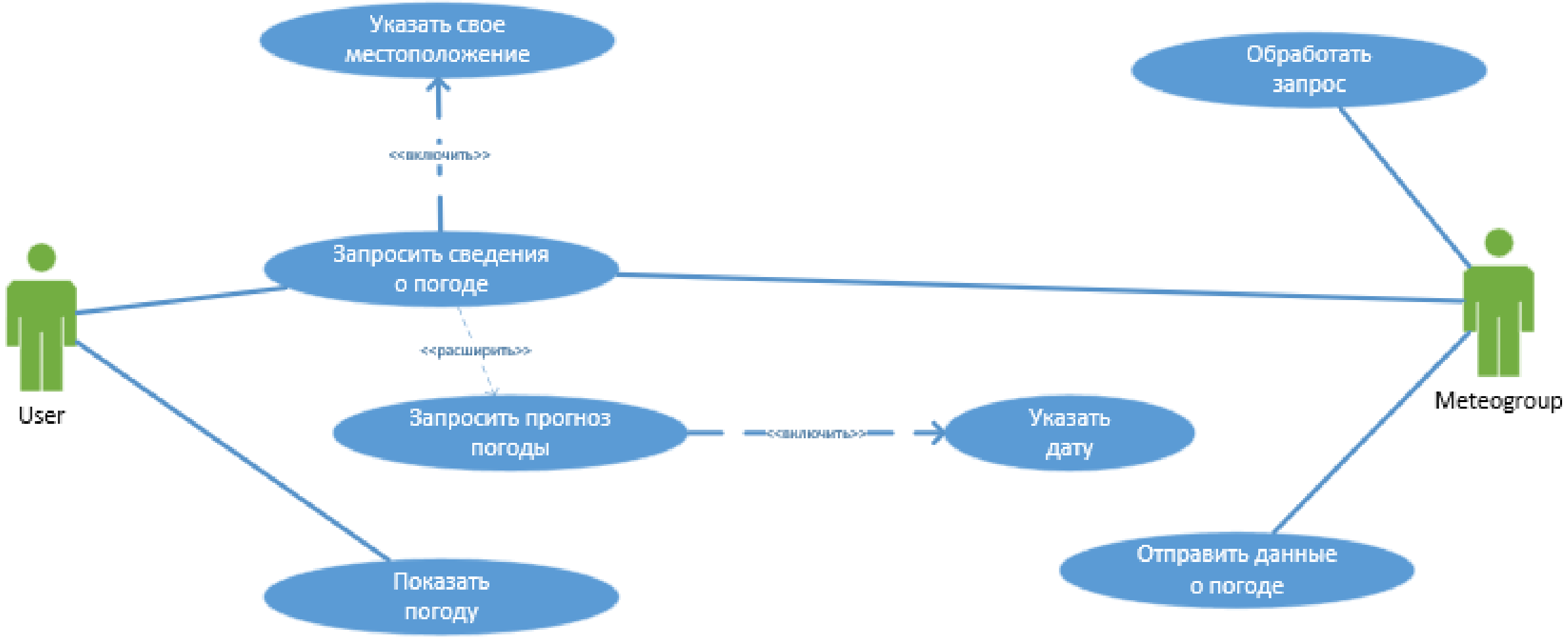
Акторы:

- Пользователь
- Сервис предоставляющий сведения о погоде

Пример 2. Телеграм-бота для погоды

Варианты использования:

- Показать погоду
- Запросить сведения
- Указать местоположение
- Запросить прогноз на дату
- Указать дату
- Обработать запрос
- Отправить сведения о погоде


Указать свое местоположение
Обработать запрос
<<включить>>
Запросить сведения о погоде
User
<<расширить>>
Meteogroup
Запросить прогноз погоды
<<включить>>
Указать дату
Показать погоду
Отправить данные о погоде

Некоторые правила построения диаграмм:

- Чрезмерная детализация не требуется
- Не рекомендуется перегружать диаграмму прецедентами (макс 15-20)
- Обязательные эл-ты: актеры, прецеденты, отношения.
- Обобщения, условия расширения- не обязательны
- размещать элементы желательно в логическом (хронологическом) порядке

Домашнее задание:

Самостоятельно выбрать приложение (мобильное приложение, веб-сайт и пр.), определить несколько требований к нему:

Функциональные требования

Нефункциональные требования

Требования пользователя

Системные Требования

Требования к производительности

Требования безопасности