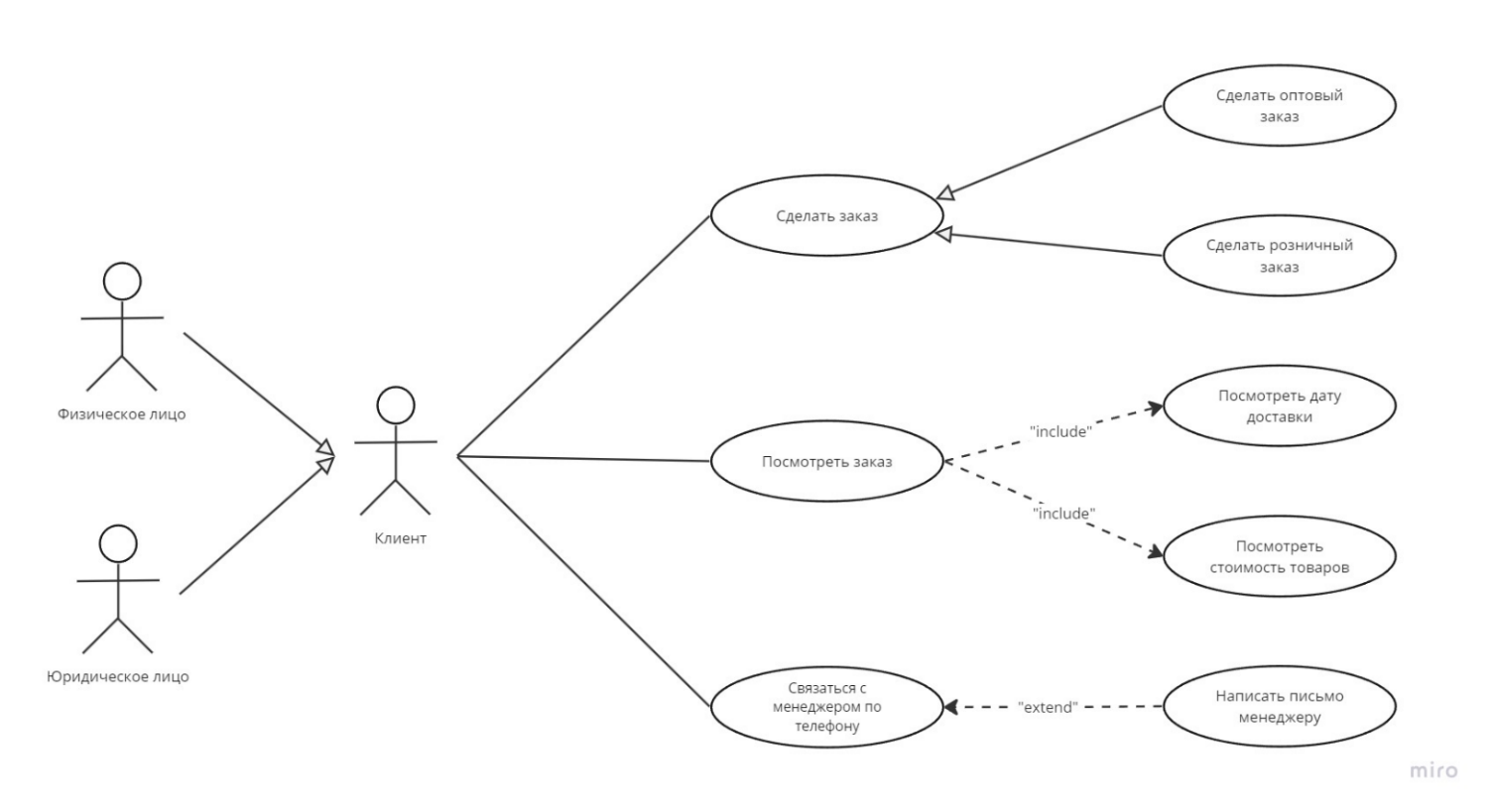
1. UML-диаграмма прецедентов (предназначение, сущности, отношения,

стереотипы)

Диаграмма прецедентов представляет собой графическое изображение вариантов использования системы, актtров и их взаимодействия



Она помогает лучше понять требования к системе и определить ее функциональные возможности.

**Актер** — это пользователь, который взаимодействует с системой. Это может быть человек, организация или даже сервер/система.

**Варианты использования:** овалы горизонтальной формы, обозначающие различные варианты использования, которые может иметь пользователь.

**Ассоциация:** линия между актерами и вариантами использования.

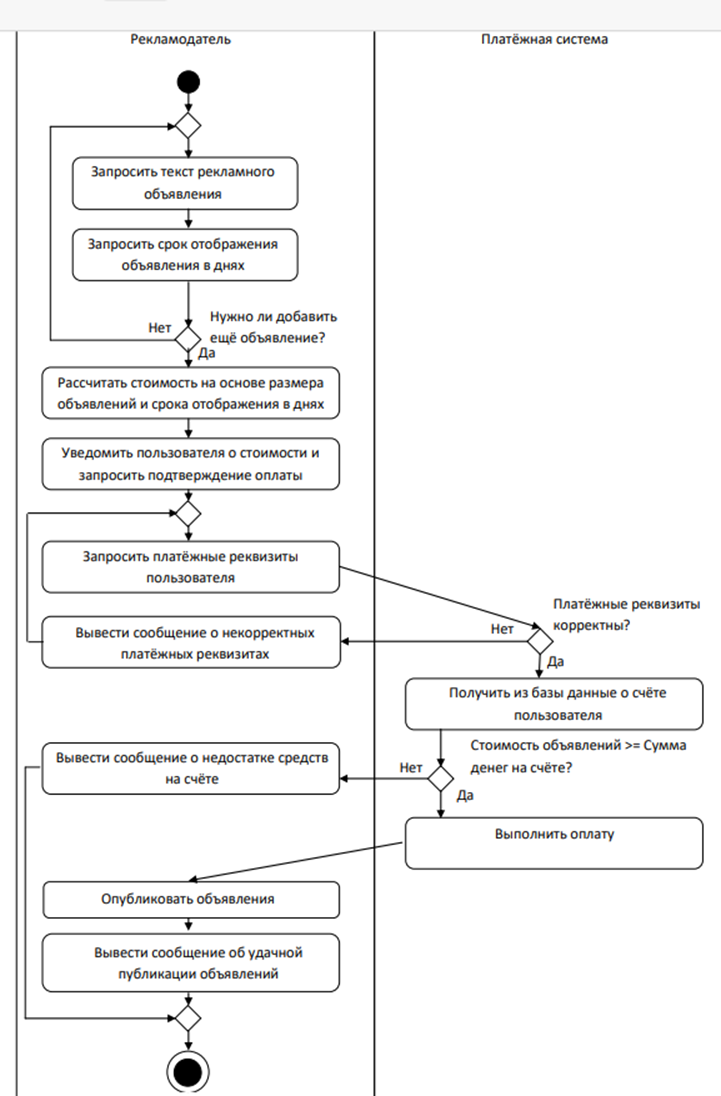
Существуют 3 типа отношений (Relationships):

* Включение (Include) - Это отношение показывает, что один вариант использования является составной частью другого варианта использования.
* Расширение (Extend) - Это отношение показывает, что расширенный вариант использования является необязательным и может быть выполнен только при определенных условиях.
* Обобщение (Generalization) - используется, когда несколько акторов имеют общие характеристики, но один актор является более общим, чем другой. Например, акторы «Клиент» и «Администратор» могут быть представлены более общим автором «Пользователь».

2. UML-диаграмма деятельности (предназначение, сущности и

отношения, моделирование условий, моделирование многопоточности).

Показывает функционирование прецедента или группы прецедентов системы в виде алгоритма.

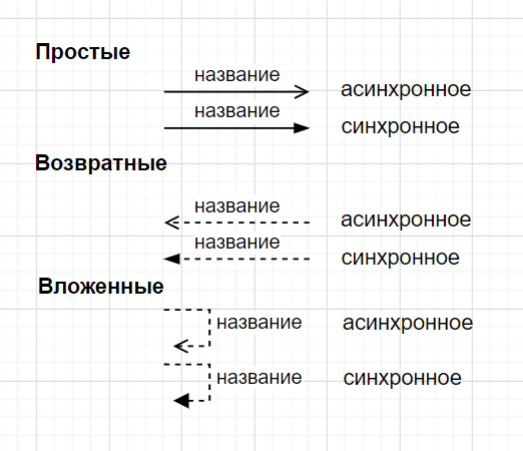


3. UML-диаграмма последовательностей (предназначение, сущности, виды сообщений, фреймы взаимодействия).

Показывает алгоритм работы прецедента или группы прецедентов в разрезе взаимодействия объектов.

Сообщения:

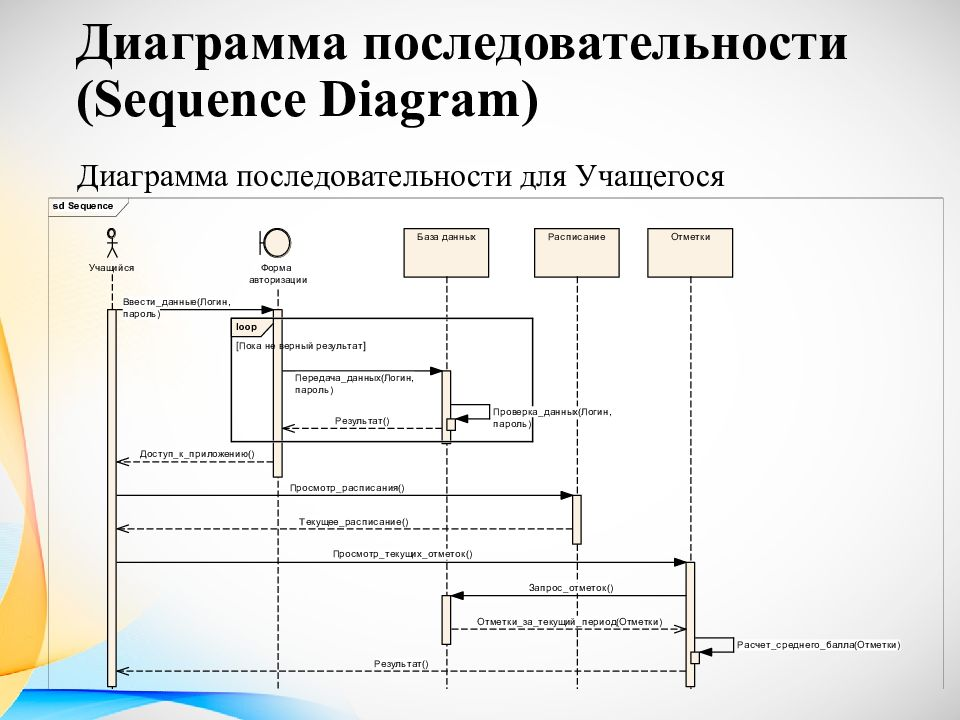
* Синхронные – важен порядок следования во времени
* Асинхронные – порядок следования во времени не важен
* Простые – запрос от одного объекта к другому
* Возвратные – ответ на запрос
* Вложенные – сообщение, передаваемое объектом к нему же самому



**Фрейм** – прямоугольная область, охватывающая часть диаграммы

* Alt – условие с 2-мя ветвями
* Opt – условие с 1-й ветвью
* Loop - цикл

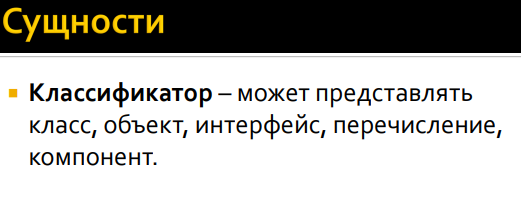
**Объекты — это сущности**, которые взаимодействуют друг с другом. Например, пользователь, топик или очередь, микросервис.



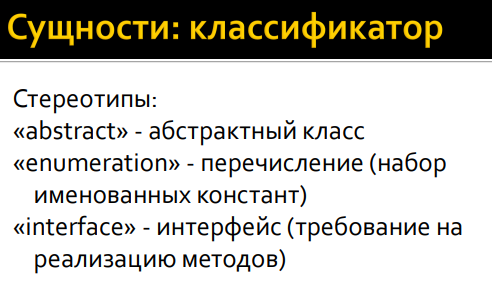
4. UML-диаграмма классов (предназначение, сущности, отношения,

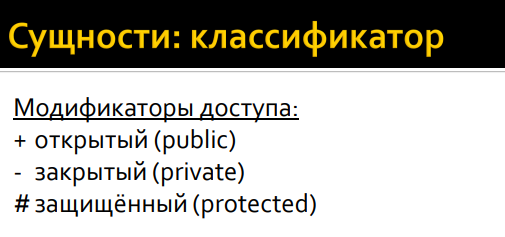
стереотипы, синтаксис описания полей и методов).

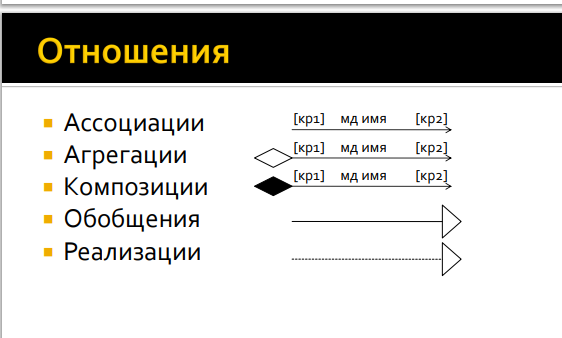
Показывает, какие классы и объекты реализуют систему, а также их взаимосвязи.

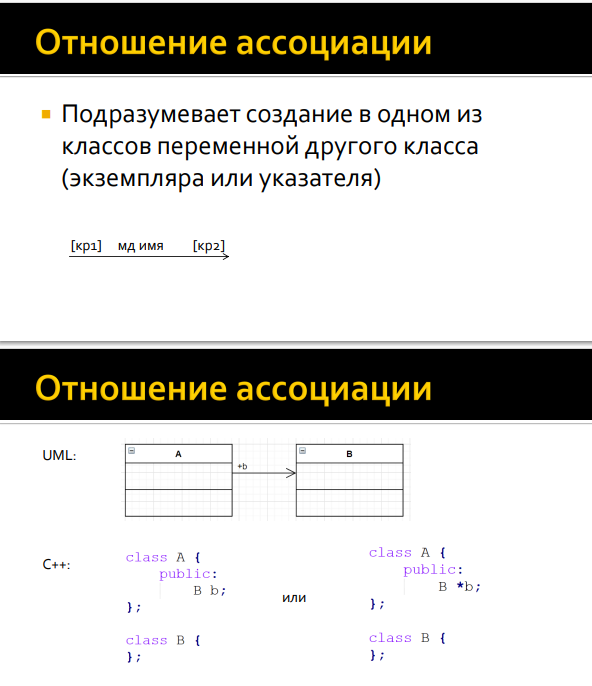


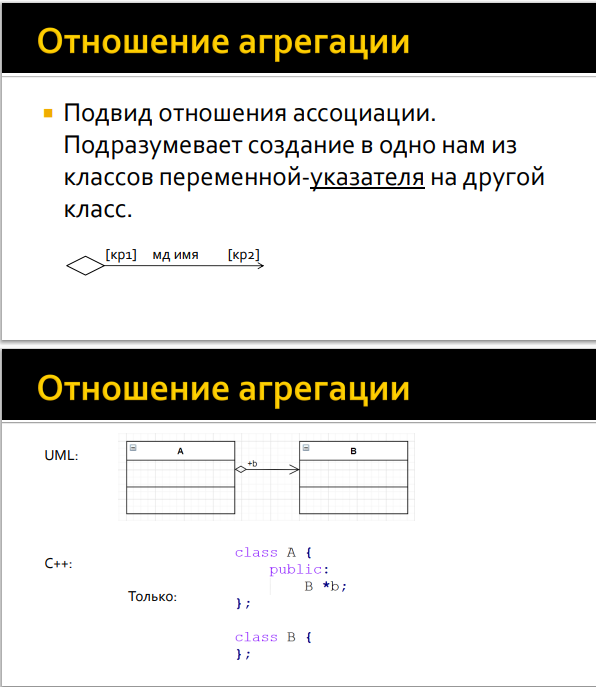


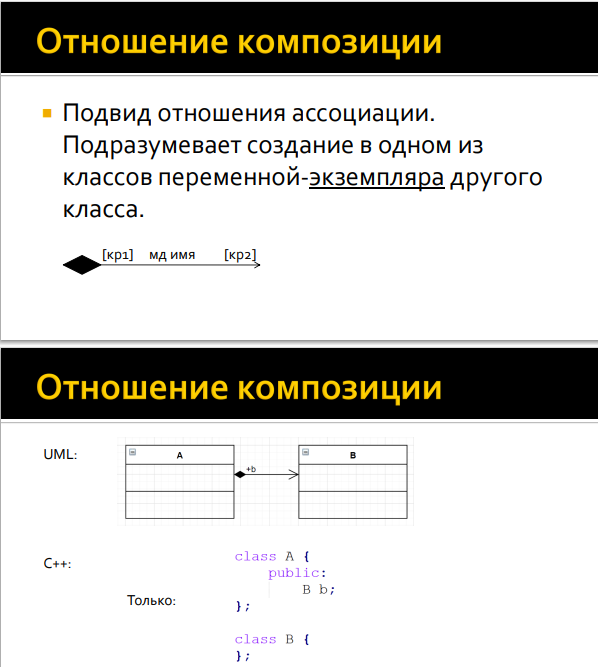


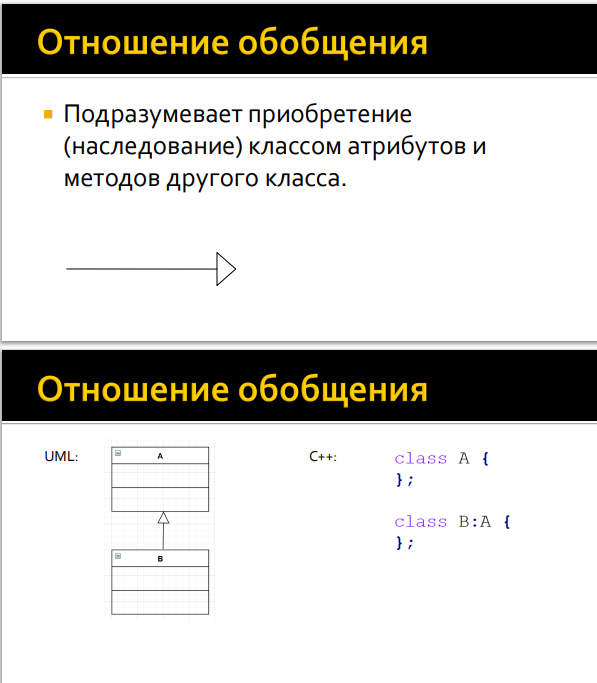


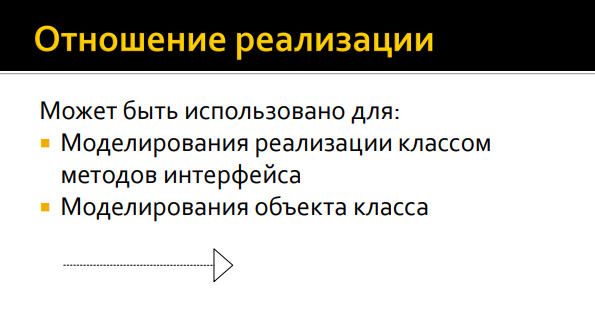


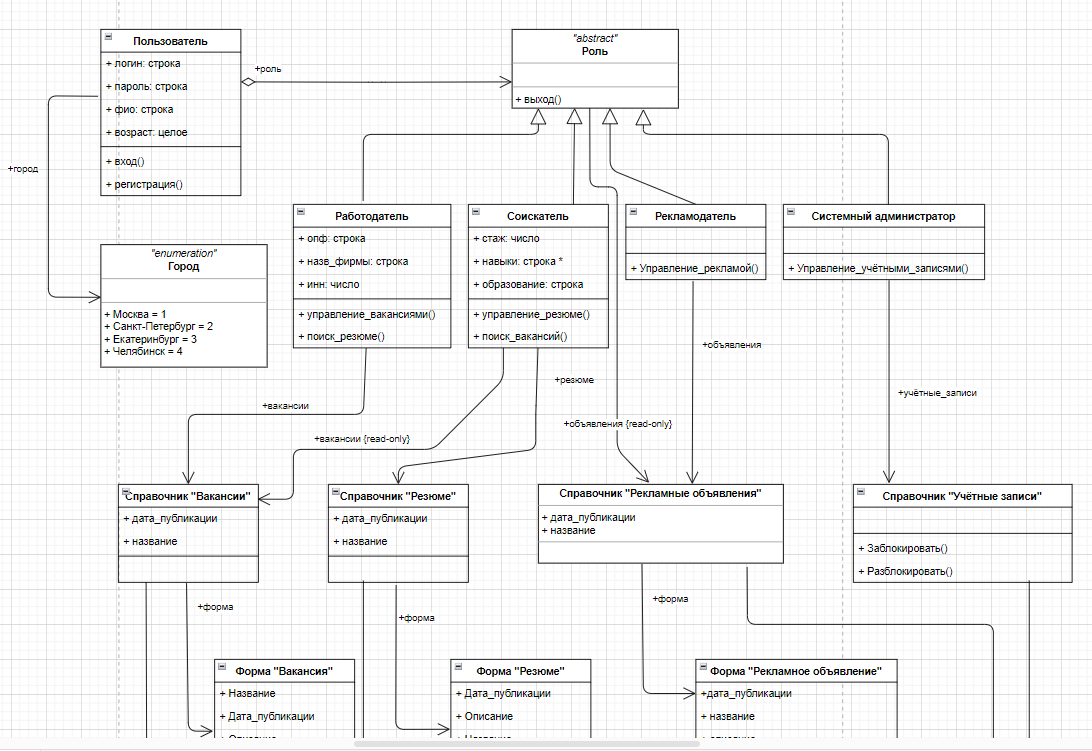








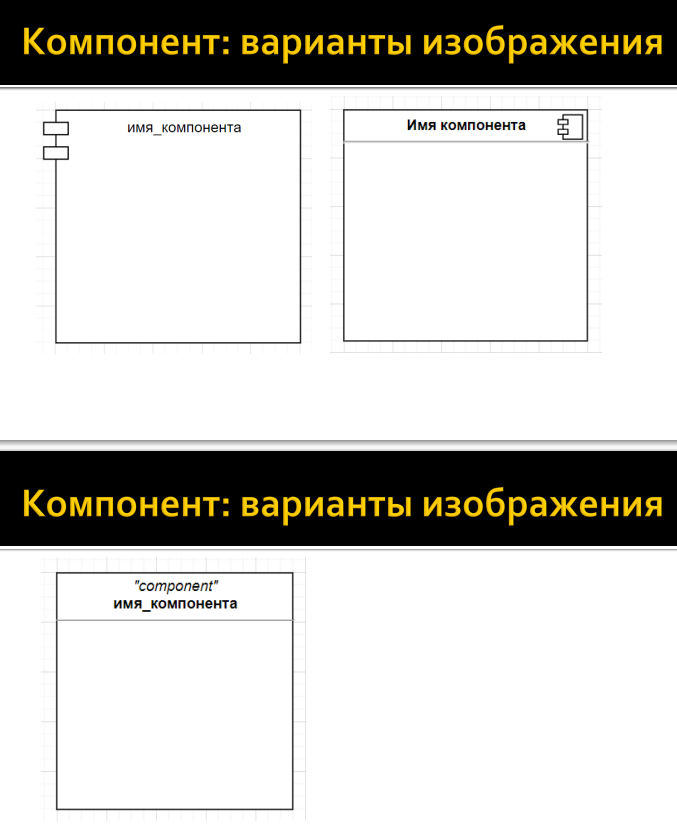


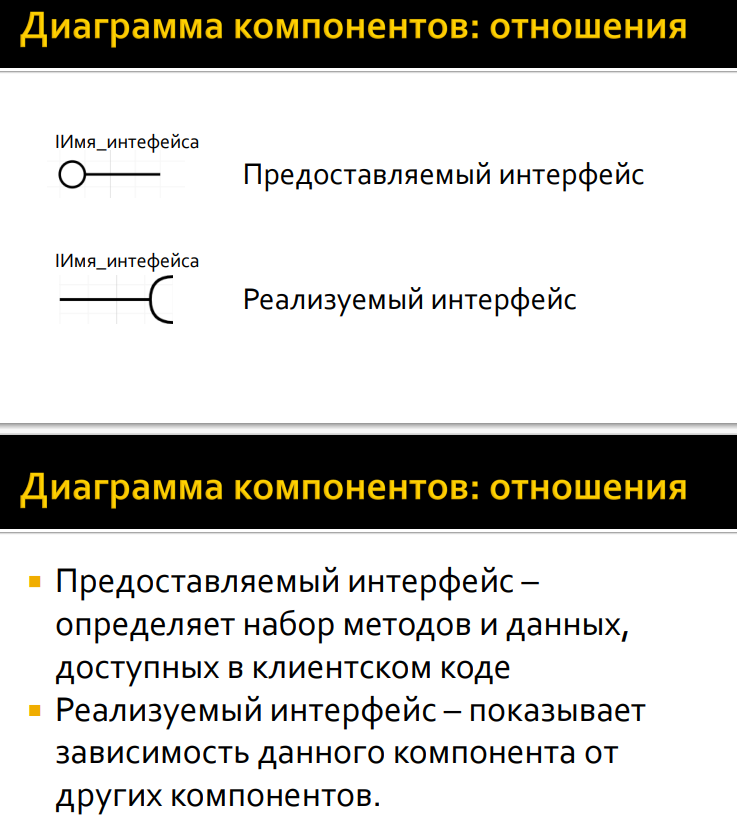


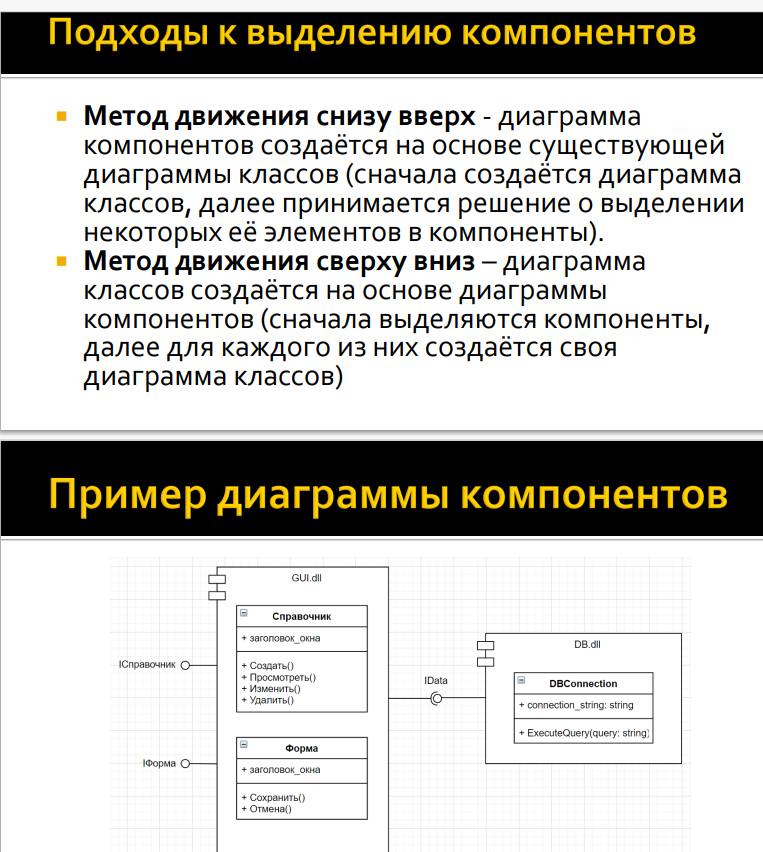
5. UML-диаграмма компонентов (предназначение, сущности и отношения, методы движения сверху вниз и снизу вверх).

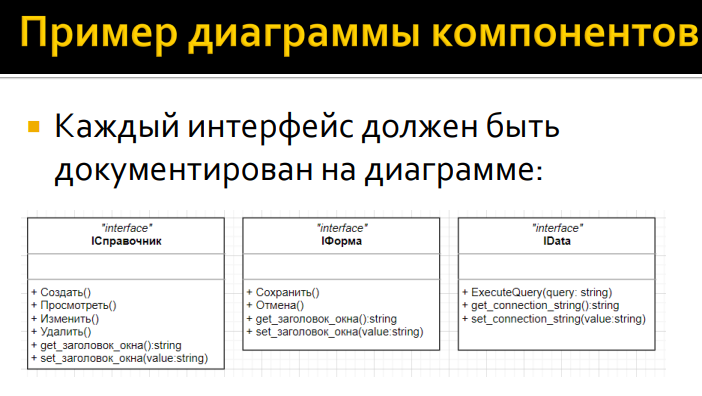
Показывает, как составные части системы (классы) распределены по модулям. Модуль (компонент) – некоторая независимая единица – например подключаемая библиотека (DLL).

Компонент - некоторый модуль системы









6. UML-диаграмма развёртывания (предназначение, сущности,

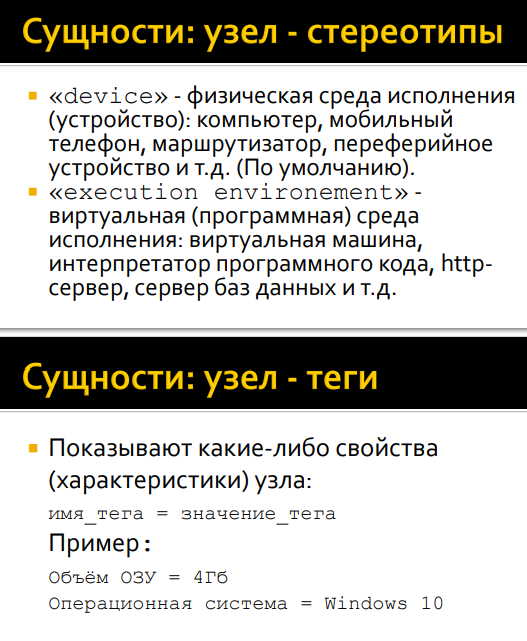
отношения и стереотипы).

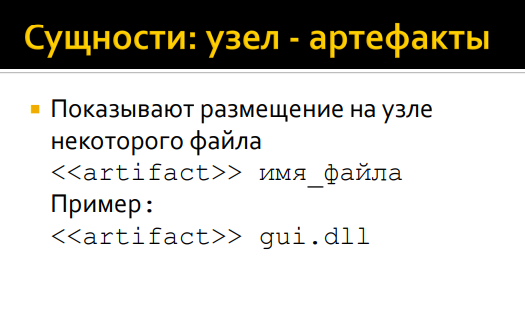
* Показывает, в какой аппаратной среде функционирует система
* Показывает размещение программных элементов по аппаратным узлам, и как эти узлы взаимосвязаны
* Может использоваться для моделирования топологии сети и аппаратного профиля системы (системных требований ПО).

Сущности:

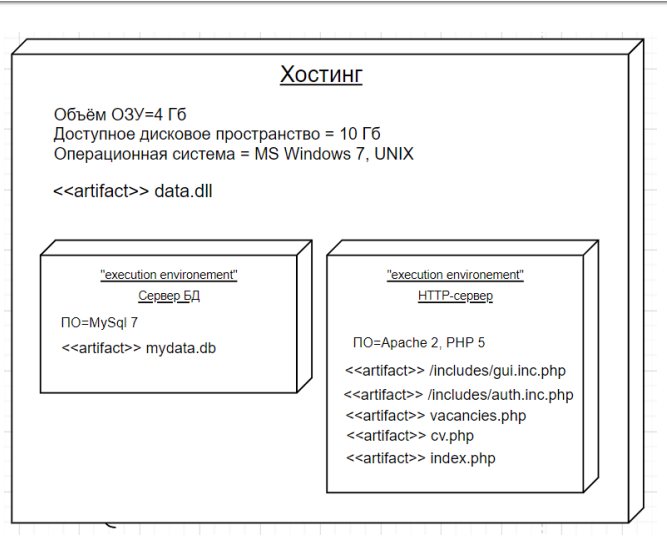
* Узел - некоторая среда исполнения



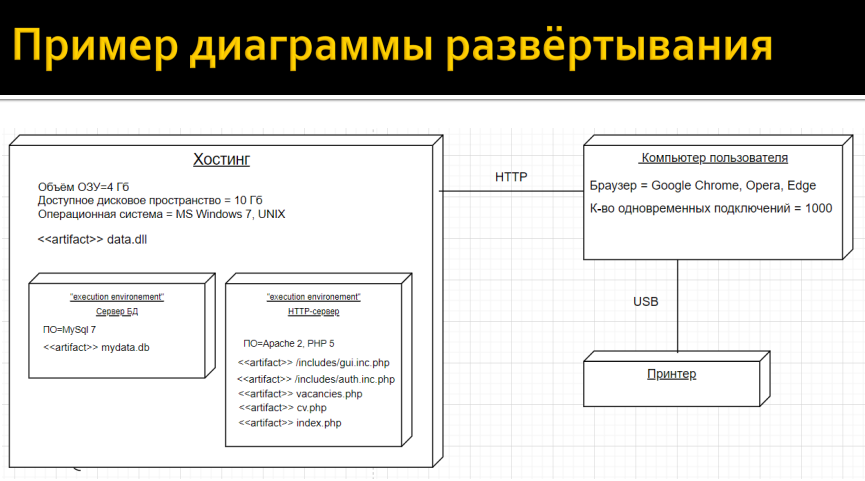




**Узлы могут вкладываться. Таким образом моделируется устройство внутри устройства, или программная среда, развёрнутая на устройстве.**

****

****

****

7. UML-диаграмма состояний (предназначение, сущности, отношения,

стереотипы, виды состояний).

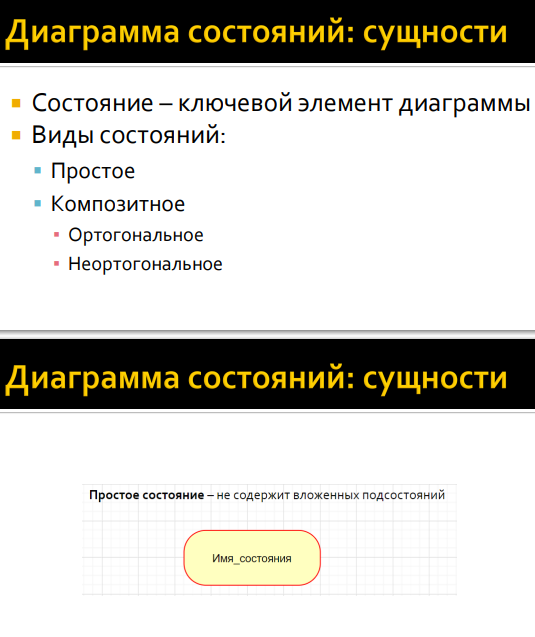
Показывает последовательность устойчивых состояний, через которую проходит система во время работы

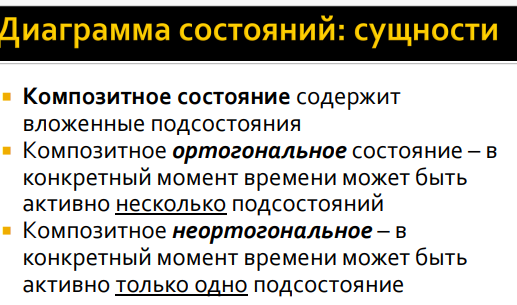
Применение диаграммы состояний:

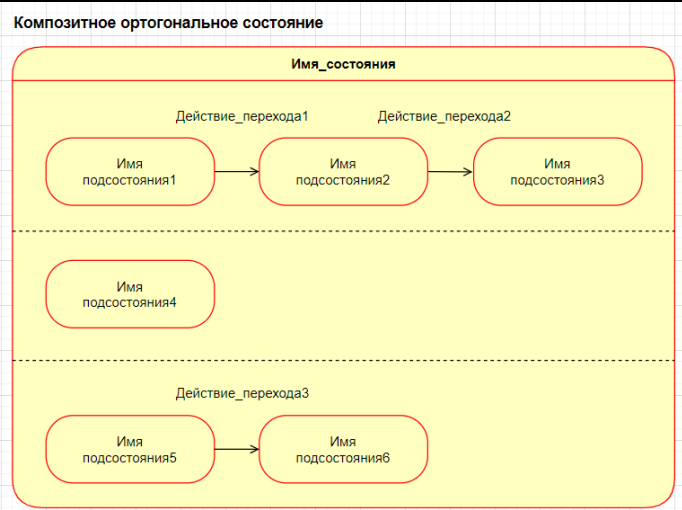
* Моделирование протоколов
* Создание карты пользовательских интерфейсов
* Работа некоторых алгоритмов приложения

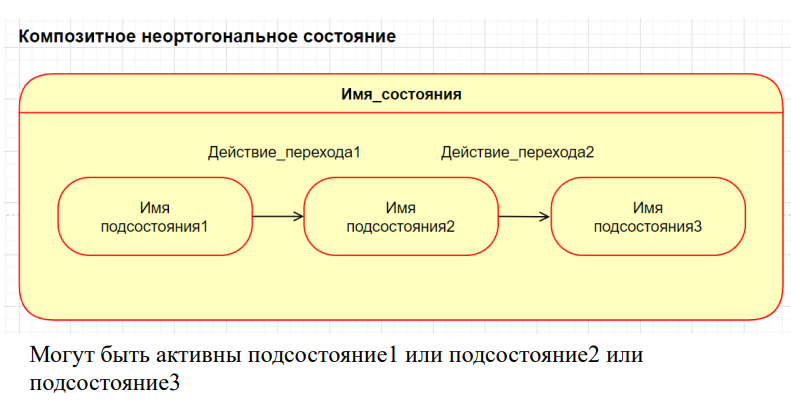
Карта интерфейсов не показывает детали реализации логики приложения, только очерёдность и условия смены экранов приложения.

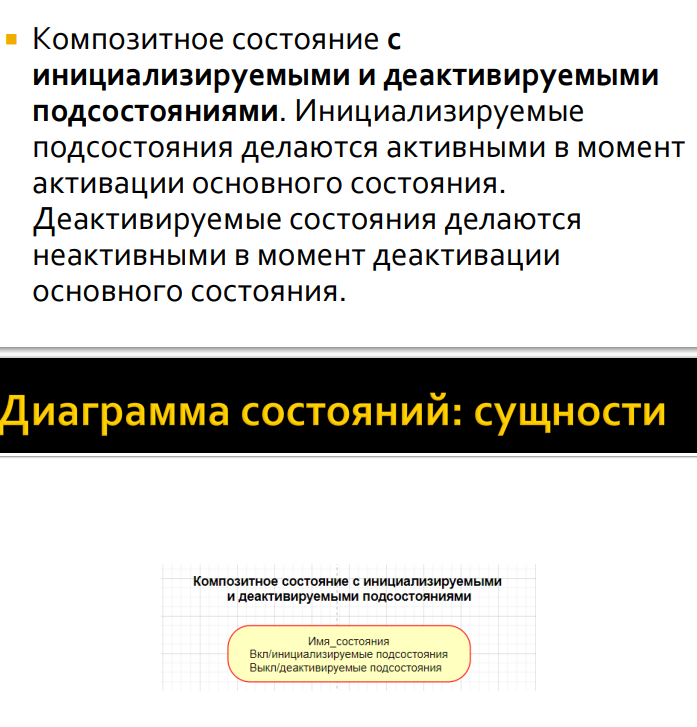


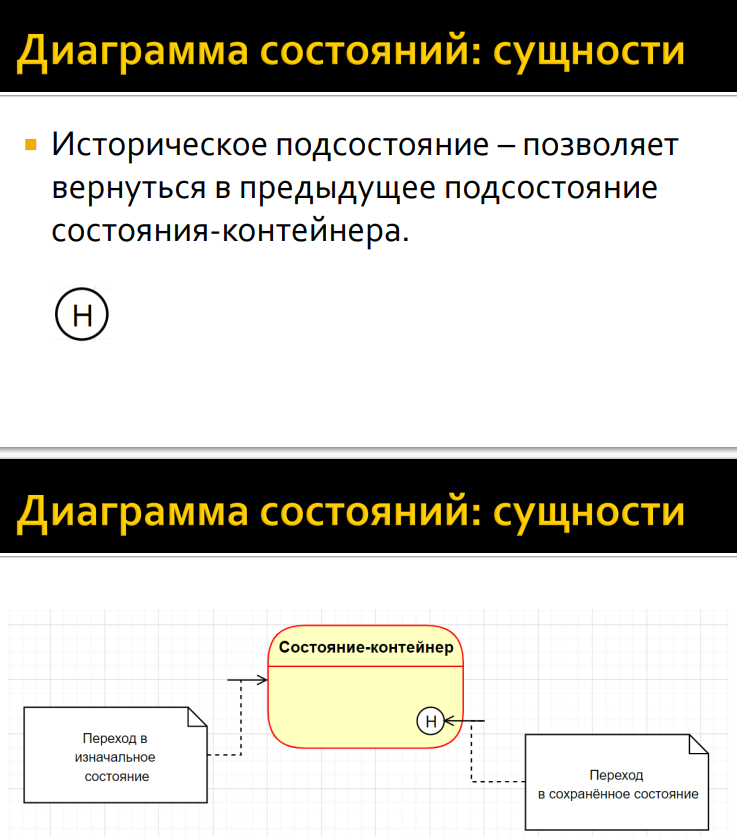


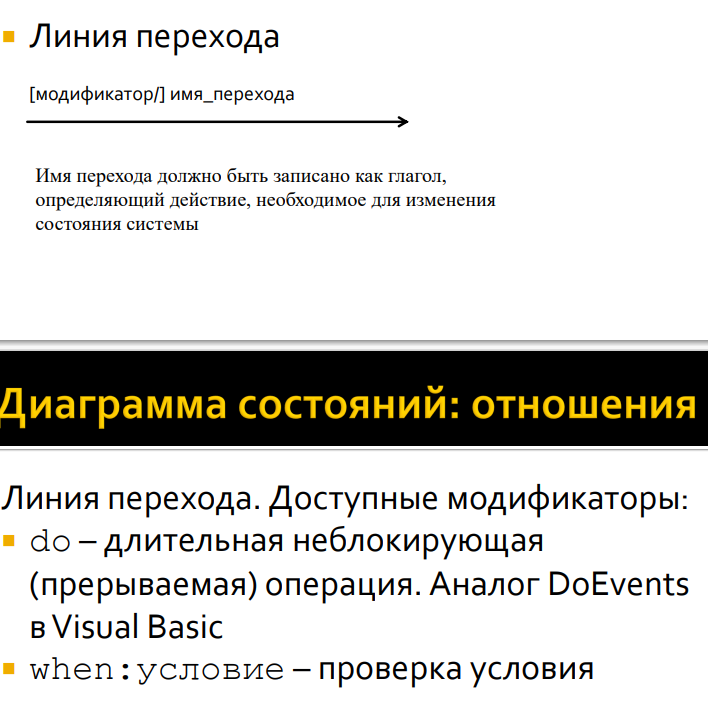


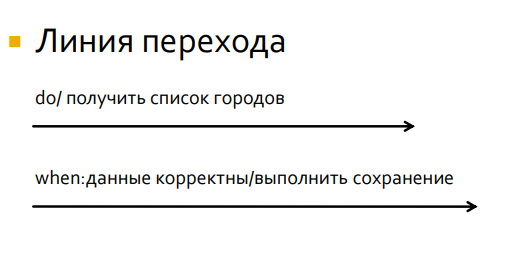


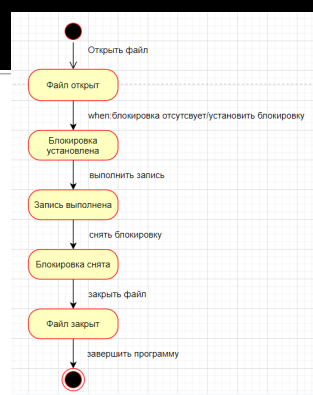












8. Тестирование программного обеспечения (Общая характеристика;

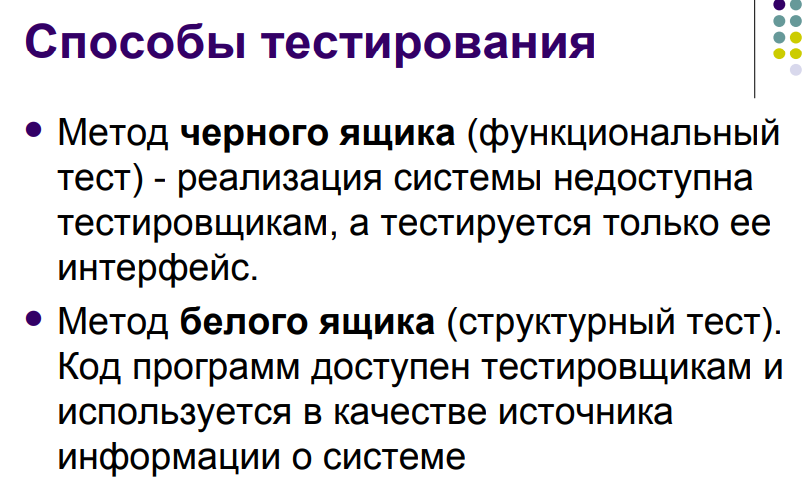
Ручное, автоматизированное, белого и чёрного ящика, виды

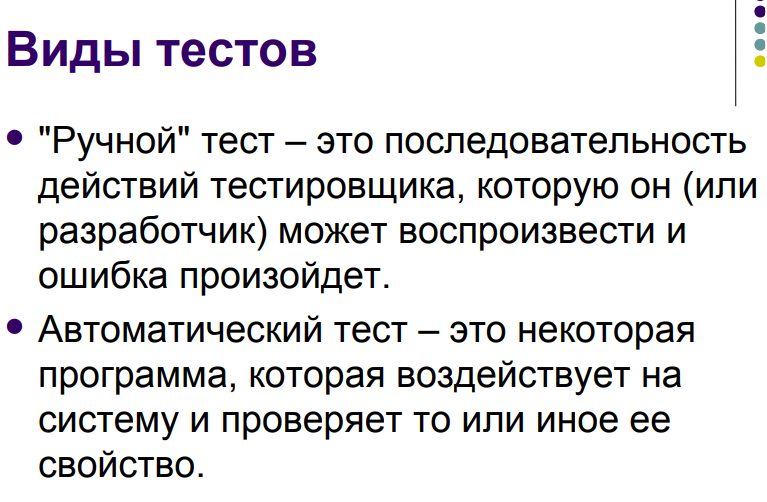
тестирования – нагрузочное, интеграционное и т.д. Методы отбора

данных для тестирования. Инструменты контроля ошибок. Модульное

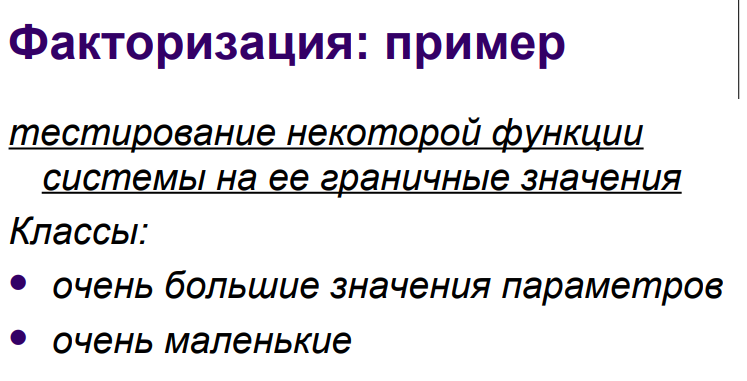
тестирование, тестирование веб-интерфейса (Selenium) и Windows интерфейса (Appium, закодированные тесты VS)).

**Тестирование ПО** - проверка соответствия между реальным поведением программы и ее ожидаемым поведением в специально заданных, искусственных условиях.



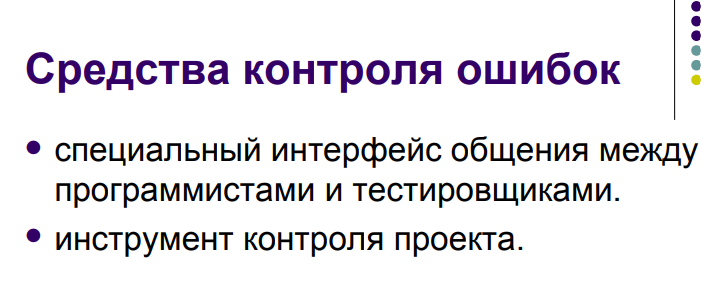


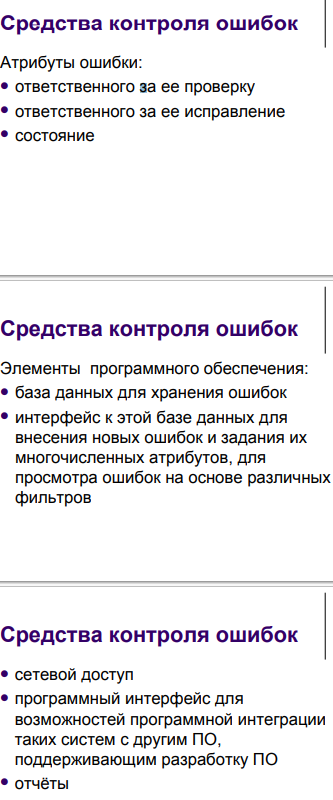
**факторизация** - множество всех возможных входных значений разбивают на значимые с точки зрения тестирования классы и "прогоняют" тесты не на всех возможных входных значениях, а берут по одному набору значений из каждого класса.



Виды тестирования:

* **Нагрузочное тестирование** – тестирование системы на корректную работу с большими объемами данных.
* **Стрессовое тестирование** – тестирование системы на устойчивость к непредвиденным ситуациям.
* Приемочное тестирование – тестирование, выполняемое при приемке систем заказчиками.
* Модульное тестирование – тестируется отдельный модуль, в отрыве от остальной системы.
* Интеграционное тестирование - две и более компонент тестируются на совместимость
* Системное тестирование – это тестирование всей системы в целом, как правило, через ее пользовательский интерфейс.
* Регрессионное тестирование – тестирование системы в процессе ее разработки и сопровождение на регресс (ухудшение качества).





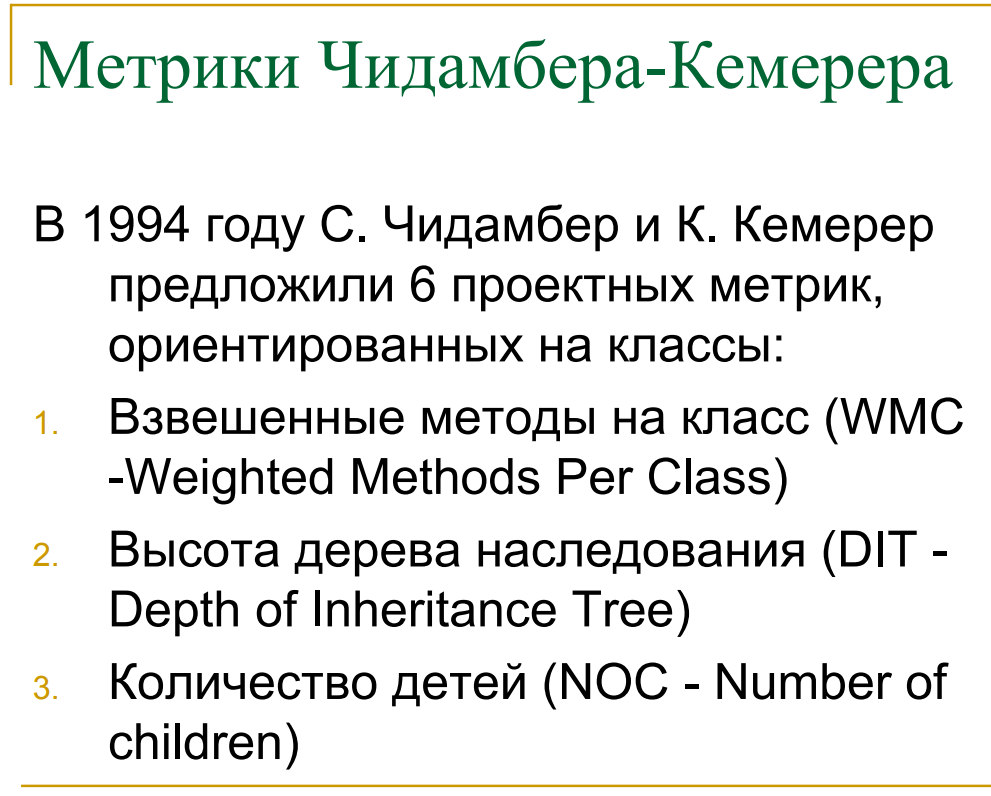
С помощью закодированных тестов пользовательского интерфейса можно проверить работоспособность пользовательского интерфейса.

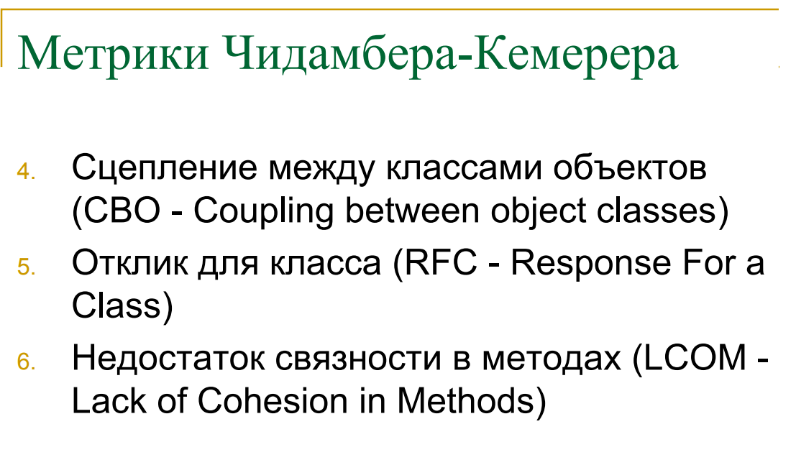
Selenium - это инструмент для тестирования веб-приложений. Он позволяет выполнять большое количество операций в браузере.

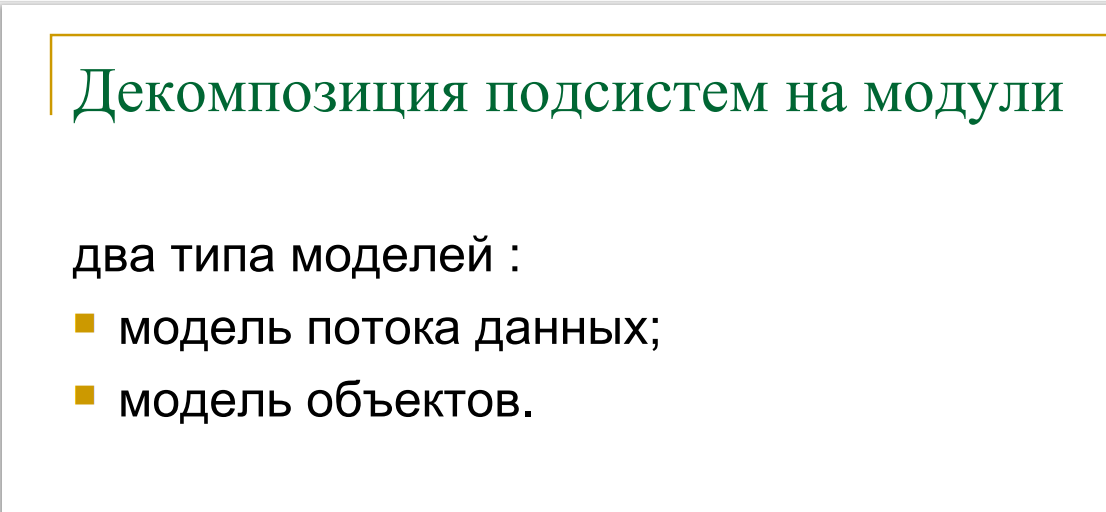
Windows Application Driver (WinAppDriver) — это драйвер, который используется для тестирования UWP-приложений с помощью Appium.

9. Метрики программного обеспечения

**Метрика программного обеспечения**-это стандарт измерения степени, в которой программная система или процесс обладает каким-либо свойством.







10.Системы контроля версий (GIT)

Git — это специальная программа, которая позволяет отслеживать любые изменения в файлах, хранить их версии и оперативно возвращаться в любое сохранённое состояние.

Git может быть локальным, централизованным или распределённым:

* **Локальный** установлен на одном компьютере и хранит файлы только в одном экземпляре в рамках настроенного окружения.
* **Централизованный** находится на общем сервере и хранит все файлы на нём.
* **Распределённый** хранит данные и в общем облачном хранилище, и в устройствах участников команды.

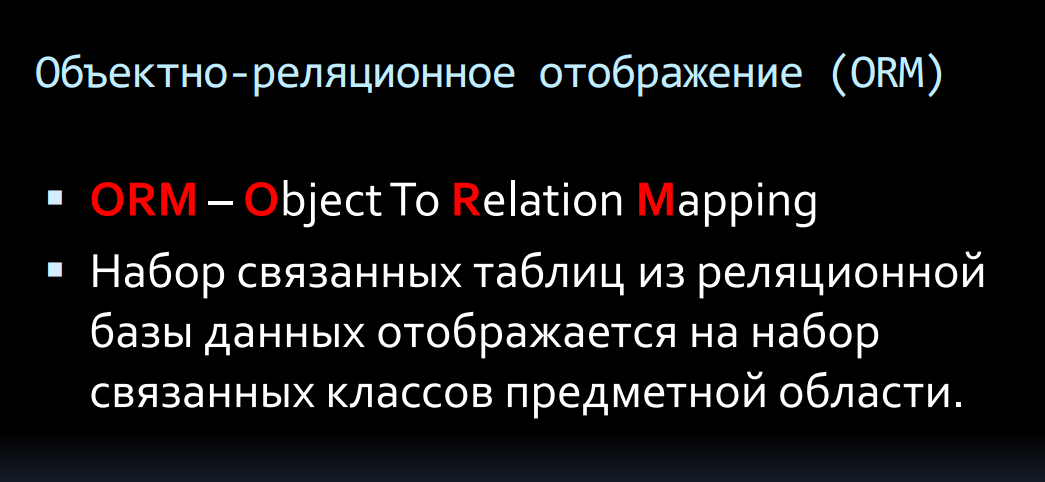
**GitHub** — это сайт-хранилище для историй версий проектов: вы подключаете Git, регистрируетесь на **GitHub**, создаёте онлайн-репозиторий и переносите файлы с Git на **GitHub**. Git — это самая популярная система контроля версий, а **GitHub** — онлайн-хранилище кода.

11.Объектно-реляционное отображение (NHibernate, LINQ)

**LINQ** (Language Integrated Query) – проект Microsoft по добавлению синтаксиса языка запросов,

напоминающего SQL, в языки программирования платформы .NET Framework.

LINQ позволяет выполнять запросы к объектам находящимся в памяти, в типизированной базе данных и в XML документе. В запросе указывается, какие сведения требуется извлечь из источника или источников данных. Также можно указать способ сортировки, группировки или структурирования сведений до их возвращения.



**NHibernate** - фреймворк позволяет делать мапинг Объектно ориентированных моделей к традиционным БД.

Маппинг — это описание соответствия между исходными и импортируемыми данными.