**1-ый** *слайд*

**{** *Вы когда-нибудь бывали в нашей столовой? Кто-то посещает её регулярно, кто-то был там только раз. Наверняка вы отмечали для себя, что очереди в столовке просто гигантские и стоять в них приходилось как минимум 10 минут. А то и 20 (Диман подтвердит)* **}**

**2-ой** *слайд*

**{** (*Итак, выделим имеющуюся проблему*.) **}**

Проблема: километровая очередь в столовой (*в которой приходится стоять половину перемены*)

**{** *Решить данную проблему призвано приложение под названием "Онлайн-Столовка" (или "Mobile-Столовка"; я ещё не определился)* **}**

Цель проекта: разработать мобильное приложение, в котором можно будет заказывать еду онлайн (*и приходя в столовую только забирать свой заказ*)

**3-ий** *слайд*

**{** Каким образом это всё будет происходить:

Пользователь делает заказ в приложении, на панели у кассирши он высвечивается (*у неё есть какой-то небольшой экранчик с touch-screen'ом, на котором она как раз-таки и заполняет заказ*) и она идёт накладывать еду.

**{** *Заказов таких будет много и она сразу будет видеть сколько нужно положить котлет, сколько грудок и т.д.*

В принципе, так во всех подобных приложениях и происходит, если вы бывали в KFC или BK. **}**

**4-ый** *слайд*

**{** И тут как раз-таки мы плавненько переходим к конкурентам на рынке, и аналогам. **}**

Конкуренты:

Конкурентов не обнаружено. ( *I mean я не нашёл никакой информации про приложения именно институтских (или даже школьных) столовых.* )

Аналоги:

А если в принципе приложухи для заказа еды рассматривать, то это KFC, BK, Вкусно – и Точка. Они позволяют как сделать заказ в определённом ресторане (и съесть его там, или забрать с собой), так и заказать на дом. Приложения, специализирующиеся на доставке на дом – это "Яндекс.Еда", "Delivery Club", которые могут доставлять еду на дом из различных ресторанов (включая вышеперечисленные).

Однако они не ориентированы на институтские или школьные столовые. Студент (или тем более школьник) не будет для каждого своего приёма пищи заказывать дорогую доставку. Причём ему нужно будет это делать заранее, за минут 40-60 до большой перемены. А это придётся делать на паре, тем самым отвлекаясь от занятия. Поэтому, я считаю, конкуренции тут как таковой нету.

**{** И ориентироваться можно будет на эти приложения, на их дизайн и устройство. **}**

**5-ый** *слайд*

На чём написаны мобильные приложения?

Популярные мобильные приложения, подобные вышеперечисленным приложениям, обычно разрабатываются с использованием следующих технологий и инструментов:

1) Языки программирования:

iOS (для iPhone и iPad): Приложения для iOS разрабатываются на языке программирования Swift или Objective-C с использованием интегрированной среды разработки (IDE) Xcode.

Android: Приложения для Android разрабатываются на языке Java или Kotlin с использованием Android Studio, которая также предоставляет среду разработки и инструменты.

2) Фреймворки и SDK (Software Development Kit):

iOS: Разработчики iOS могут использовать Cocoa Touch (для Swift и Objective-C) и iOS SDK для создания пользовательского интерфейса, работы с графикой, а также доступа к устройству и функциональности iOS.

Android: Для Android доступен Android SDK, который включает в себя библиотеки и инструменты для разработки Android-приложений.

3) Базы данных:

Для хранения данных, таких как меню, цены и информация о заказах, используются базы данных. Популярные базы данных для мобильных приложений включают SQLite, Realm, Firebase Realtime Database и другие.

4) Дизайн и интерфейс пользователя:

Для создания пользовательского интерфейса и дизайна приложения используются инструменты, такие как Adobe XD, Sketch или Figma. Для создания анимаций и интерактивности могут применяться библиотеки и фреймворки, например, SwiftUI для iOS.

5) Среды разработки:

Для разработки приложений на iOS используется Xcode, а для Android - Android Studio. Эти среды предоставляют средства для создания, тестирования и отладки приложений.

6) Серверная часть (Backend):

Для реализации функциональности, связанной с обработкой заказов, оплатой и управлением меню, может потребоваться серверная часть. Здесь используются языки программирования, такие как Python, Java, Node.js, а также фреймворки для создания веб-сервисов, например, Django, Spring Boot или Express.js.

**6-ый** *слайд*

По поводу масштабируемости.

Масштабируемость: Если всё хорошо пойдёт в нашем институте, то можно предложить головному ВУЗу в Москве это приложение, а если и там всё хорошо пойдёт, то тогда предложить всем остальным институтам это приложение, а также школам.

**7-ый** *слайд*

Конкретно какую пользу приносит это приложение (*а также кто этим всем будет пользоваться*)? То есть эффект приложения заключается в:

– Экономии времени студентов и преподавателей

и

– Создании удобного способа предзаказа еды

а также

– Оптимизации персонала (сокращение расходов на оплату труда)

**8-ый** *слайд*

Как можно развить приложение в будущем:

- ввести систему акций (в зависимости от дня на какой-нибудь продукт)

- система подписки на еду (для преподов, к примеру), которая будет в целом выгоднее, чем разовые заказы

- добавить различные диеты

- добавить учёт изменений в меню и отслеживание статуса заказов (*если, конечно, это не будет сделано до этого*)

*Тем самым добавив ещё несколько пунктов в "список пользы".*

**9-ый** *слайд*

Что нужно сделать в ближайшее время (очевидные цели):

- Опросить студентов и преподавателей, а также работников столовой и руководство уника, что они думают насчёт этой идеи

- Узнать, какие блюда предлагаются, как изменяется меню.

- Обсудить вопрос регулирования количества товара. Т.е. придумать, как кассирша может у себя на панели регулировать наличие того или иного продукта. Например, она должна каждый день перед началом работы столовой вбивать количество котлет, которое будет уменьшаться, когда кто-то будет их заказывать. Или всегда делать одинаковое количество котлет, чтоб каждый раз не вбивать. А если вдруг так вышло, что мяса не хватило или вообще его не было и котлет нет, то она бы написала количество "0" и у пользователей бы это отобразилось что товар закончился и они бы не смогли это блюдо заказать.

А с гарниром ещё сложнее, ведь макароны и гречка в штуках не измеряются. С гарниром придётся что-то придумывать. Определить сколько уходит на одну тарелку, чтоб примерно понимать. И тогда уже ввести систему измерения гарнира ТАРЕЛКАМИ )

**10-ый** *слайд*

План разработки приложения. Что нужно будет делать дальше:

1) Написать ТЗ. (Решив, какие функции нужно включить, на каких платформах разрабатывать и т.д.)?

2) На основе требований определить общую архитектуру приложения. Решить, какие технологии, инструменты и языки программирования будут использоваться.

3) Создать макеты и прототипы пользовательского интерфейса для мобильного приложения. Определить, как оно будет выглядеть и взаимодействовать с пользователем.

4) Разработать структуру базы данных для хранения меню, заказов и информации о платежах.

5) Интегрировать систему приема платежей, выяснив, как вообще подключаются мобильные приложения к банковской системе России, чтобы можно было платить онлайн через карточку.

6) Разработать клиентскую и серверную части, реализовав коммуникацию между ними.

7) Провести тестирование отдельных компонентов приложения (UI, базы данных, платежей).

8) Проверить, как работают все компоненты вместе, включая взаимодействие между клиентской и серверной частями.

9) Оцените уровень безопасности приложения и базы данных. Провести тесты на уязвимости.

**11-ый** *слайд*

Неутешительные новости.

Скорее всего, мне не удастся сделать приложение даже за 2 года, ибо для него нужна целая команда разработчиков (программисты, дизайнеры, тестировщики). Вот такие дела... но это реальность; в соло (причём не обладая какими-то знаниями) такое крутое приложение сделать невозможно.

Поэтому я решил, что в качестве своей ВКР я создам прототип приложения. Он будет фокусироваться на frontend-части, (то есть на визуальном интерфейсе и базовой функциональности) без полноценной реализации backend-части и реального взаимодействия с внешними системами, такими как банковские платежные системы.

Если кто-то не знает, что такое прототип. Прототип – это демонстрация концепции и идеи приложения. Он служит для того, чтобы понять, как приложение будет выглядеть и каким образом пользователи будут с ним взаимодействовать.

Создание прототипа обычно требует меньше времени и ресурсов, чем полная разработка. Он разрабатывается, чтобы быстро представить идею и собрать обратную связь.

**12-ый** *слайд*

Если у кого-то наблюдается похожая ситуация с проектом, как и у меня, то вот вам гайд о том, как создаётся прототип.

1) Определите ключевые функции: Выберите наиболее важные функции, которые хотите продемонстрировать в прототипе.

2) Используйте инструменты для прототипирования: Существует множество инструментов для создания прототипов, таких как Figma, Sketch, Adobe XD, InVision и многие другие.

3) Создайте макеты экранов: Используйте выбранный инструмент для создания макетов экранов вашего приложения. Разработайте дизайн интерфейса, включая элементы управления, кнопки и текстовые поля.

4) Создайте интерактивные переходы: Добавьте интерактивность в прототип, чтобы пользователь мог взаимодействовать с элементами интерфейса.

5) Демонстрация функциональности: Прототип должен продемонстрировать, как пользователь сможет пользоваться вашим приложением.

6) Соберите обратную связь: После создания прототипа, показывайте его коллегам, друзьям или даже потенциальным пользователям. Соберите обратную связь и учтите ее при последующей разработке полного приложения.

Вывод:

Создание прототипа поможет вам лучше понять, как будет работать ваше приложение, и даст вам возможность продемонстрировать его концепцию на дипломной защите. После сдачи диплома вы сможете продолжить разработку и реализацию полноценного приложения.

**13-ый** *слайд*

Я вижу своё приложение примерно таким образом.

На главной панели при первом заходе пользователя высвечивается окошечко с регистрацией. Можно его закрыть, конечно, и не регистрироваться; но, когда в конце ты будешь вбивать данные банковской карты, они не сохранятся впоследствии и при дальнейшем использовании приложения без регистрации придётся каждый раз вводить данные банковской карточки.

Итак, если пользователь зарегистрирован, или закрыл окно регистрации на главной странице сразу появляется меню продуктов. То есть список категорий: "первые блюда", "вторые блюда", "салаты", "питьё", "дополнительные" и т.д.

Он открывает категорию и видит продукты. На продукты он может нажать, рассмотреть их поближе, посмотреть цену, описание, калорийность, выбрать количество и добавить в корзину.

В правом нижнем углу будет корзина. Нажимаешь на неё, подтверждаешь заказ и оформляешь. Если в первый раз заказываешь, то надо будет вбить данные банковской карточки и будет предложено сохранить данные, чтоб в дальнейшем не вводить каждый раз.

В верхнем углу справа будут 3 полоски. Это элемент, при нажатии на который, выпадает список. В нём будет ячейки:

-- "Профиль", в котором можно будет указать имя и фамилию

-- "Мои заказы", в которых будет отображаться статус заказа и можно будет посмотреть историю заказов

-- "О приложении", там будет ссылка на мой Ютуб-канал =D

и

-- "Выйти из системы"

1) Регистрация и вход:

- При первом заходе пользователя появляется окно с возможностью регистрации.

- Пользователь может закрыть окно и использовать приложение без регистрации.

- Банковская информация НЕ сохраняется при использовании БЕЗ регистрации.

2) Главное меню:

На главной странице отображается меню продуктов с категориями ("1-ые блюда", "2-ые блюда", "салаты", "питьё", "дополнительное" и др.)

3) Просмотр продуктов:

- После выбора категории пользователь может листать список блюд и "нажимать" на них для детального просмотра.

- Для каждого блюда отображаются: цена, описание, калорийность и др.

4) Добавление в корзину:

- Пользователь может выбирать продукты, устанавливать количество и добавлять их в корзину.

5) Корзина:

- В правом нижнем углу есть корзина, при нажатии на которую отображаются выбранные продукты.

- Пользователь может подтвердить заказ и оформить его.

6) Меню "три полоски":

- В верхнем углу справа есть иконка с тремя полосками – это меню с выпадающим списком. В нём присутствуют такие опции, как "Мой профиль", "Мои заказы", "О приложении" и "Выйти из системы".

**14-ый** *слайд*

Как выглядела схема? То есть схема заказа будет выглядеть следующим образом

**15-ый** *слайд*

Рассмотрим прототип, созданный в Figma

**16-ый** *слайд*

Для своего приложения я выбрал **Итеративную и инкрементную модель.** Что это такое, почему именно её и что это вообще за модели такие? Давайте разбираться. Но для начала поговорим о жизненном цикле ПО.

Что такое ЖЦ ПО? 💡 Это ряд взаимосвязанных этапов, через которые проходит продукт: от момента зарождения потребности до завершения работы продукта. Вспомним же структуру ЖЦ ПО. Из каких этапов он состоит?

**17-ый** *слайд*

\*Картинка\* – посмотрите внимательно на слайд. Создание любого ПО можно разделить на шесть этапов (и перечисляю). Теперь поподробнее о каждом.

**18-ый** *слайд*

1. Для начала команда должна сформулировать, что она создаёт и зачем. Для этого необходимо собрать требования у заказчика.

💡 Заказчик — это человек или группа людей, выступающие от лица пользователей ПО. Часто это руководитель отдела или владелец бизнеса.

Заказчик может отлично знать проблемные точки в своих процессах, но при этом не понимать, что с ними делать. Перед стартом работы бизнес-аналитик собирает требования у заказчика: задаёт уточняющие вопросы, анализирует работу конкурентов, изучает уже существующую документацию, анализирует обратную связь от пользователей, наблюдает за процессами — делает всё, чтобы изучить и понять картину продукта в настоящем и то, какое решение поможет учесть пожелания заказчика.

2. После сбора информации аналитик перекладывает своё видение на «бумагу». Формируется документ, в нём прописывают: требования к функционалу, который планируется разработать; процессы в их нынешнем состоянии и то, к какому виду они должны прийти после разработки. Также бизнес-аналитик в команде с дизайнером документирует прототип решения для ПО.

💡 Прототип ПО — это частичное, возможное и предварительное воплощение предлагаемого нового продукта.

Прототипы интерфейсов помогают:

* визуализировать предварительное решение заказчику;
* «заложить фундамент» решения, на основании которого будет спроектирована функциональность.

Документирование необходимо для того, чтобы зафиксировать общее понимание и согласие о том, как должен выглядеть и работать будущий продукт.

3. После документирования команда договаривается, как должны формироваться и храниться данные внутри информационной системы. Вся дальнейшая разработка опирается на архитектуру. Без хорошо спроектированной архитектуры сложно создать качественный продукт.

Обычно описание архитектуры — это задача системного аналитика. Но также, эту задачу могут передать и бизнес-аналитику — как универсальному IT-специалисту внутри компании. В этом случае бизнес-аналитик консультируется с разработчиками и архитекторами системы и описывает согласованное решение в документации.

4. Разработать информационную систему значит написать для неё код.

Код пишут разработчики на основании документации, описанной аналитиком. На этом этапе разработчики, опираясь на прототипы, создают интерфейсы, в которых будут работать пользователи, формируют структуру хранения данных, прописывают логику автоматических операций, прокладывают пути обмена данными с другими системами. Обычно разработчики делятся на фронт- и бэк-разработчиков: первые отвечают за разработку пользовательских интерфейсов, вторые — за работу системы «под капотом», то есть части ПО, невидимой пользователю.

💡 Пользовательский интерфейс — экраны системы, с помощью которых пользователь взаимодействует с ПО. Экраны заполняются кнопками, полями для ввода, изображениями, анимацией и прочими атрибутами интерфейса, через которые пользователь даёт команду ПО.

5. Тестирование системы — это комплексная проверка её алгоритмов на:

* корректную работу в разных операционных системах (Mac, Windows, другие), браузерах (например Яндекс Браузер, Mozilla Firefox, Safari, Chrome) и на разных устройствах (смартфоны, планшеты, компьютеры);
* соответствие полученного результата ожиданиям в прописанных требованиях;
* наличие «слепых зон» в логике работы функциональности, которые не были обозначены на этапе аналитики.

В тестировании участвует инженер по тестированию (по-другому — тестировщик). Он проверяет созданный продукт на соответствие требованиям и договорённостям, которые аналитик выявил на этапе сбора и анализа, и зафиксировал на этапе документирования. Все найденные несоответствия или слепые зоны тестировщик подробно описывает и обсуждает с аналитиком. В случае если работа функциональности отличается от указанной логики в требованиях, тестировщик «откатывает» задачу обратно на этап разработки.

6. Внедрение системы — это обучение пользователей, выдача учётных записей и поддержка пользователей в начале работы с системой. Чтобы продукт «ушёл в народ», нужно подружить пользователя с ПО: продемонстрировать ПО в действии, написать инструкцию и ответить на возникшие вопросы. Процесс внедрения завершается переходом продукта в стадию поддержки решения — то есть в сопровождение функциональности и поддержания актуальности её логики. На этом этапе с пользователем взаимодействует служба поддержки, иногда сотрудники службы консультируются по процессам с аналитиком.

Так как IT-продукт — это система, которая живёт и развивается, жизненный цикл может воспроизводиться несколько раз. Например, команда создала продукт, который позволяет заказывать товары через мобильное приложение. Чтобы добавить в продукт систему лояльности, необходимо снова пройти этапы жизненного цикла — собрать требования к системе лояльности, задокументировать их, спроектировать и так далее.

**19-ый** *слайд*

На данном слайде представлены графики степени вовлечения участников команды в проект. Можно заметить, что вовлечение каждого из участников команды меняется в зависимости от этапа.

**20-ый** *слайд*

Рассмотрим 7 основных моделей разработки ПО. Их суть, плюсы и минусы.

**21-ый** (Waterfall model)

Одна из самых старых, подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. В модели Waterfall легко управлять проектом. Благодаря её жесткости разработка проходит быстро, стоимость и срок заранее определены. Но это палка о двух концах. Каскадная модель будет давать отличный результат только в проектах с четко и заранее определенными требованиями и способами их реализации. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочёты, о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Исправление осознанных в процессе создания недостатков возможно, и требует от одного до трех дополнительных соглашений к контракту с небольшим ТЗ.

**Когда использовать каскадную методологию?**

* Только тогда, когда требования известны, понятны и зафиксированы. Противоречивых требований не имеется.
* Нет проблем с доступностью программистов нужной квалификации.
* В относительно небольших проектах.

**22-oй** (V-model)

Унаследовала структуру «шаг за шагом» от каскадной модели. V-образная модель применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и так далее. Особенностью модели можно считать то, что она направлена на тщательную проверку и [тестирование продукта](http://www.edsd.ru/ru/uslugi/testirovanie_po), находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время кодирования пишутся модульные тесты.  
  
Пример работы на основе V-методологии — мобильное приложение для европейского сотового оператора, который экономит расходы на роуминг во время путешествий. Проект выполняется по четкому ТЗ, но в него включен значительный этап тестирования: удобства интерфейса, функционального, нагрузочного и в том числе интеграционного, которое должно подтверждать, что несколько компонентов от различных производителей вместе работают стабильно, невозможна кража денег и кредитов.

**Когда использовать V-модель?**

* Если требуется тщательное тестирование продукта, то V-модель оправдает заложенную в себя идею: validation and verification.
* Для малых и средних проектов, где требования четко определены и фиксированы.
* В условиях доступности инженеров необходимой квалификации, особенно тестировщиков.

**23-ий** (Инкрементная)

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система.

Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы. Например, её применяли для создания читалки DefView, а следом и сети электронных библиотек Vivaldi.  
  
В качестве примера, рассмотрим cуть одного инкремента. [Сеть электронных библиотек Vivaldi](http://habrahabr.ru/company/edison/blog/232033/) пришла на смену DefView. DefView подключалась к одному серверу документов, а теперь может подключаться ко многим. На площадку учреждения, желающего транслировать свой контент определенной аудитории, устанавливается сервер хранения, который напрямую обращается к документам и преобразует их в нужный формат. Появился корневой элемент архитектуры — центральный сервер Vivaldi, выступающий в роли единой поисковой системы по всем серверам хранения, установленным в различных учреждениях.

**Когда использовать инкрементную модель?**

* Когда основные требования к системе четко определены и понятны. В то же время некоторые детали могут дорабатываться с течением времени.
* Требуется ранний вывод продукта на рынок.
* Есть несколько рисковых фич или целей.

**24-ый** (RAD-модель)

RAD-модель — разновидность инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.

**Модель быстрой разработки приложений включает следующие фазы:**

* Бизнес-моделирование: определение списка информационных потоков между различными подразделениями.
* Моделирование данных: информация, собранная на предыдущем этапе, используется для определения объектов и иных сущностей, необходимых для циркуляции информации.
* Моделирование процесса: информационные потоки связывают объекты для достижения целей разработки.
* Сборка приложения: используются средства автоматической сборки для преобразования моделей системы автоматического проектирования в код.
* Тестирование: тестируются новые компоненты и интерфейсы.

**Когда используется RAD-модель?**  
Может использоваться только при наличии высококвалифицированных и узкоспециализированных архитекторов. Бюджет проекта большой, чтобы оплатить этих специалистов вместе со стоимостью готовых инструментов автоматизированной сборки. RAD-модель может быть выбрана при уверенном знании целевого бизнеса и необходимости срочного производства системы в течение 2-3 месяцев.

**25-ый** (Итеративная модель)

Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.

На диаграмме показана итерационная «разработка» Мона Лизы. Как видно, в первой итерации есть лишь набросок Джоконды, во второй — появляются цвета, а третья итерация добавляет деталей, насыщенности и завершает процесс. В инкрементной же модели функционал продукта наращивается по кусочкам, продукт составляется из частей. В отличие от итерационной модели, каждый кусочек представляет собой целостный элемент.  
  
Примером итерационной разработки может служить распознавание голоса. Первые исследования и подготовка научного аппарата начались давно, в начале — в мыслях, затем — на бумаге. С каждой новой итерацией качество распознавания улучшалось. Тем не менее, идеальное распознавание еще не достигнуто, следовательно, задача еще не решена полностью.  
  
Когда оптимально использовать итеративную модель?

* Требования к конечной системе заранее четко определены и понятны.
* Проект большой или очень большой.
* Основная задача должна быть определена, но детали реализации могут эволюционировать с течением времени.

**26-ой** (Agile -модель)

В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике.

В основе такого типа — непродолжительные ежедневные встречи — «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают:

* отчёт о проделанной работе с момента последнего Scrum’a;
* список задач, которые сотрудник должен выполнить до следующего собрания;
* затруднения, возникшие в ходе работы.

Методология подходит для больших или нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка. Соответственно, в процессе реализации требования изменяются. Стоит вспомнить класс творческих людей, которым свойственно генерировать, выдавать и опробовать новые идеи еженедельно или даже ежедневно. Гибкая разработка лучше всего подходит для этого психотипа руководителей. Примером клиентских проектов является [Электронная Система Медицинских Осмотров](http://www.kvzrm.ru/), созданная для проведения массовых медосмотров в считанные минуты.

Когда использовать Agile?

* Когда потребности пользователей постоянно меняются в динамическом бизнесе.
* Изменения на Agile реализуются за меньшую цену из-за частых инкрементов.
* В отличие от модели водопада, в гибкой модели для старта проекта достаточно лишь небольшого планирования.

**27-ой** (Spiral -модель)

«Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.  
  
Спиральная модель предполагает 4 этапа для каждого витка:

1. планирование;
2. анализ рисков;
3. конструирование;
4. оценка результата и при удовлетворительном качестве переход к новому витку.

Эта модель не подойдет для малых проектов, она резонна для сложных и дорогих, например, таких, как разработка системы документооборота для банка, когда каждый следующий шаг требует большего анализа для оценки последствий, чем программирование. На проекте по [разработке СЭД для ОДУ Сибири СО ЕЭС](http://edsd.biz/odu-sibiri) два совещания об изменении кодификации разделов электронного архива занимают в 10 раз больше времени, чем объединение двух папок программистом. Государственные проекты, начинаются с подготовки экспертным сообществом дорогостоящей концепции, которая отнюдь не всегда бесполезна, поскольку окупается в масштабах страны.

**28-ой** (Просто читай слайд из презентации)

**29-ый** (Просто читай слайд из презентации)

**30-ый** *слайд*

Подытожим. Концепции моделей:

1. **Каскадная модель (Waterfall)**:

Процесс разработки линейный и последовательный, разделен на фазы (такие как анализ, проектирование, разработка, тестирование и сопровождение), и каждая фаза начинается после завершения предыдущей.

1. **Итеративная и инкрементальная модель (Iterative and Incremental)**:

Разработка происходит через серию итераций или добавлением новых компонентов (инкрементов) к предыдущим версиям приложения.

1. **Spiral (Спиральная) модель**:

Разработка происходит в циклах, каждый из которых включает в себя планирование, риск-анализ, инженерное выполнение и оценку. Развитие приложения по спирали.

1. **Модель V (V-Model)**:

Отражает параллельную связь между фазами разработки и фазами тестирования. Каждой фазе разработки соответствует фаза тестирования.

1. **RAD (Rapid Application Development)**:

Основана на быстром создании прототипов с последующим итеративным улучшением. Акцент на быстром завершении проекта.

1. **Agile (Гибкая) модель разработки**:

Основана на коллаборации и гибкости. Разработка разбита на короткие циклы (итерации) с акцентом на постоянной адаптации к изменяющимся требованиям.