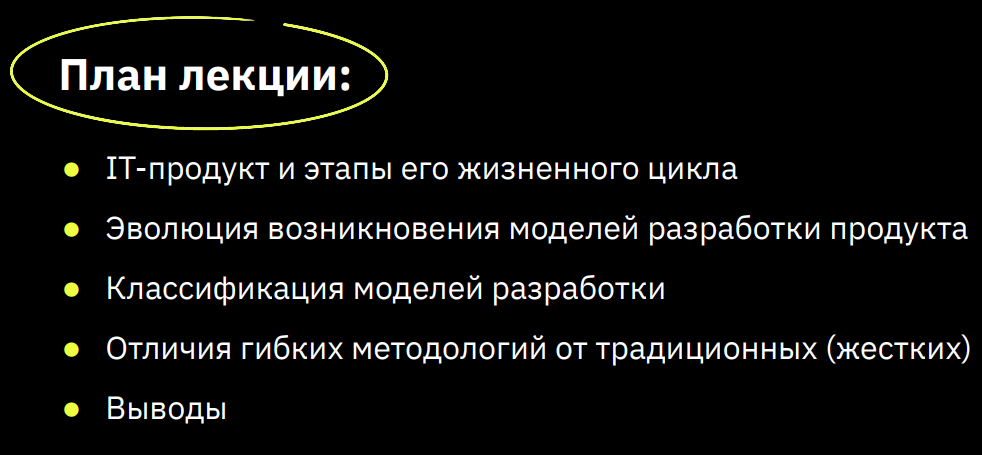
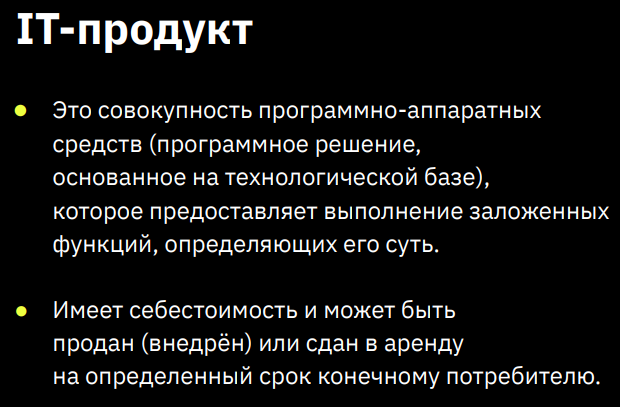
**Урок 1. Гибкие методологии. История возникновения.**

**Здравствуйте, уважаемые студенты? Хочу вас поприветствовать на нашем курсе «гибкие методологии» сегодня у нас будет первый урок. Первый урок у нас посвящен гибким методологиям и истории их возникновения.  
Для начала хотел бы представиться: меня зовут Игорь Зуриев. Я являюсь деканом факультета компании GeekBrance руководитель проектов в строительстве. Помимо этого, в компании** **GeekBrance я являюсь автором-разработчиком и преподавателем в различных курсах на факультете, менеджер проектов в IT.  
Теперь касательно моего опыта, где я работал, что я делал, какие проекты реализовывал? Во-первых, начал я свою карьеру в строительных компаниях. Буквально через определенное количество времени я перешел в компанию «Лукойл», где начал активно принимать участие в реализации таких крупных проектов, как строительство газоперерабатывающего завода, строительство магистрального газопровода и обустройство двух крупных газовых месторождений. Общий объем инвестиций в данный проект составлял порядка 11 млрд. долларов США. Параллельно со стройкой мы активно понимали, что сам газоперерабатывающий завод, магистральных газопровод и вся инфраструктура не может не функционировать без какой-то IT-составляющей. Эти IT-проекты были связаны с технологической основой производства, а также с финансовыми моментами по управлению всем этим большим комплексом. В то время как раз-таки я начал активно интересоваться IT-сферой и плавно участвовать. Начал с реализации IT-проектов в рамках всей этой большой крупной структуры. Так сложилось, что моя работа стала непосредственно связана с этими двумя крупными отраслями эта стройка и IT-технологии, и они постоянно перекликались друг с другом в процессе всей моей рабочей деятельности.  
Касательно моей основной работы сейчас, это аэропорт в Шереметьево. В рамках данной структуры я реализовываю отдельные проекты по автоматизации системы бюджетирования в компании, которые связаны непосредственно со стройкой крупных объектов, таких как терминал C. Новый недавно буквально ввели в эксплуатацию строительство взлетно-посадочной полосы и других различных объектов на территории аэропорта.**

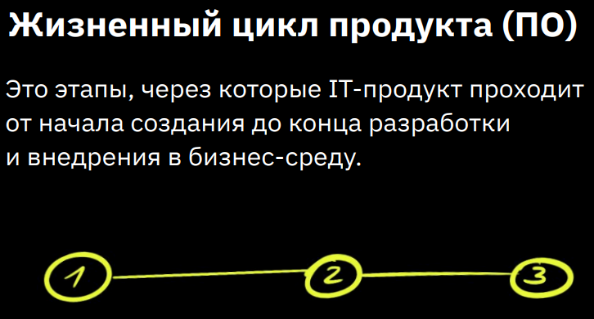
Теперь хотел бы кратко рассказать про план нашего курса, про что мы будем говорить на протяжении наших шести уроков. В самом начале объясню историю возникновения гибких методологий и как вообще человечество пришло к тому, что нужно развивать именно гибкий подход к разработке и любого другого программного обеспечения. На втором уроке мы поговорим про такую методологию, точнее говоря, это больше подход и Agil. На третьем обсудим уже более практическую методологию, точнее, даже говоря, это фреймворк. Это будет Scrun на четвёртом уроке поговорим про бережную разработку Len. На пятом обсудим Canban, а на шестом сделаем выводы, как правильно выбрать гибкую методологию при реализации своего IT-проекта. При этом не просто ее выбрать, а именно не ошибиться с этим выбором и подобрать даже можно сказать, не только одну методологию, с помощью которой вы будете реализовывать весь свой проект, а выработать определенную стратегию по применению той или иной методологии на определенных отрезках жизненного цикла разработки вашего программного обеспечения.

План нашего сегодняшнего урока: во-первых, мы обсудим, что такое IT-продукт, потому что без понимания, что это такое и как в целом происходит разработка, невозможно говорить, зачем нам нужны какие-то методологии и какие-то подходы в реализации IT-продукта. Обсудим, из чего состоит жизненный цикл нашего продукта. Поговорим про эволюцию возникновения в целом абсолютно всех методологий разработки продукта, начиная от твердых жестких так называемых и заканчивая гибкими. Поговорим про классификацию моделей разработки, на что она делится, как люди постепенно пришли к тому пониманию, что нужно все-таки избирать именно гибкий подход реализации IT-проекта. Поговорим про отличия гибких методологий от жестких от традиционных, и сделаем выводы.



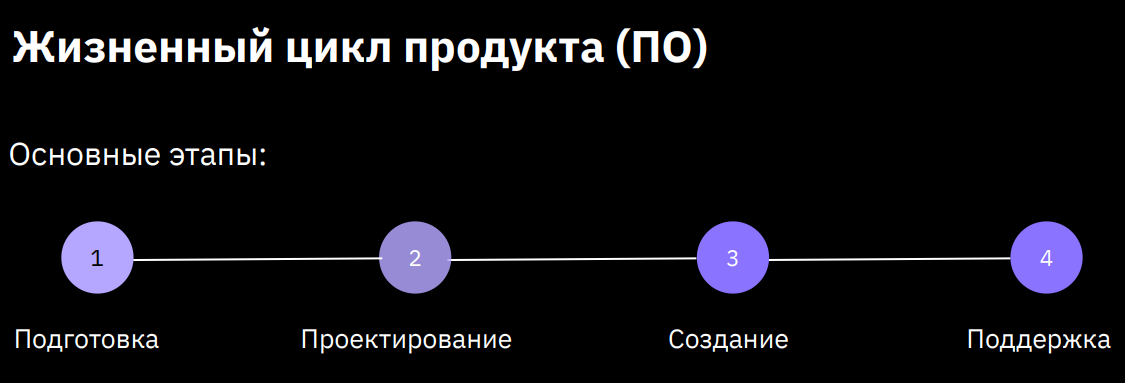
Для начала что такое IT-продукт и в целом, как происходит его разработка. Любой IT-продукт содержит в себе жизненный цикл, то есть начиная от самого нуля и заканчивая тем самым приложением, сайтом, каким-то программным обеспечением, который вы видите в итоге вы и которым вы уже пользуетесь. Для начала обсудим, что такое IT-продукт в целом, если говорить более научным термином, это совокупность программно-аппаратных средств, программное решение, основание на технологической базе, предоставляет выполнение заложенных в нем функций и, соответственно, определяет суть этого самого IT-продукта. Конечно, это определение достаточно сложно для восприятия, особенно если вы новичок в этой сфере. Если говорить проще, то любой продукт имеет себестоимость, может быть продан, внедрен куда-то сдан в аренду на определенный срок конечному потребителю. И в целом это любое программное обеспечение, любое приложение, любой сайт, в общем, все то, чем вы в повседневной жизни очень активно пользуетесь. Любой из вас имеет смартфон, достает, использует какое-то приложение, переводит деньги другу, платят за какие-то товары, услуги в интернете и так далее. Все это результат именно тех самых IT-проектов, которые порождают IT-продукт и несут конечную ценность для потребителя.

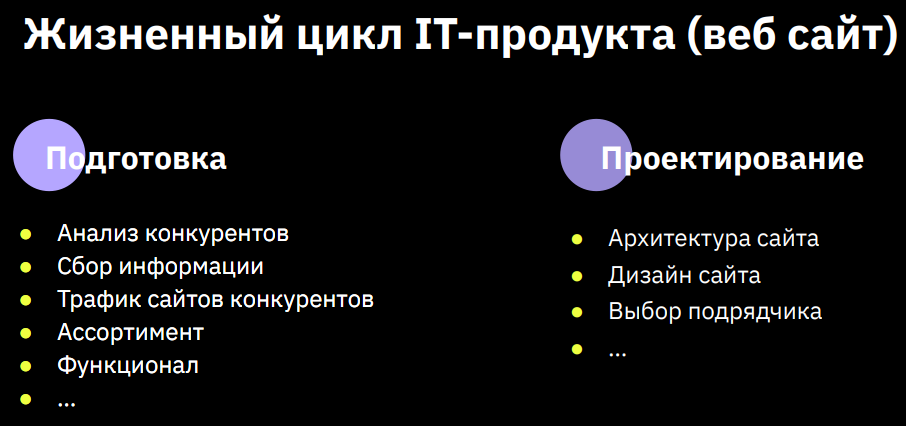
Не будем спешить и забегать вперед, я бы хотел привести пример, яркий пример, который вам позволит понять, для чего в целом нужен IT-продукт и почему важно при его разработке. Например, допустим, у вас есть идея: вы хотите создать сайт интернет-магазина одежды, вам это нравится, вы погружены в эту сферу, вы хотите развиваться, хотите, чтобы росли ваши продажи, хотите продвигать свой товар на рынок, и поэтому вы решаете пойти в интернет, создаете сайт для интернет-магазина одежды. Вам уже представляется, как выглядит этот сайт, что это очень удобный сайт, там много покупателей, много пользователей, у вас постоянно растет статистика посещения вашего сайта и он находится в первых строчках поиска. У вас красивый интерфейс, все очень круто, все замечательно, но по сути дела это всего лишь вершина айсберга. Как вы видите, на данном слайде, все приложения, которыми мы сейчас пользуемся, это та самая вершина айсберга. Под водой находится как раз-таки то, что не видно глазу любого простого пользователя, те самые труды, те самые груды времени, усилий, которые были приложены со стороны разработчиков, со стороны руководителей этих проектов и в целом идейных вдохновителей данных проектов. Так вот, как воплотить вам в жизнь свою идею по созданию интернет-сайта для своего магазины одежды. Что вам нужно будет сделать, естественно, для того чтобы получить конечный продукт, вам нужно будет полностью прогнать его по жизненному циклу, то есть разработать, что такое жизненный цикл продукта. Это этапы, через которые ваш IT-продукт будет проходить от начала создания до конца разработки и непосредственно поставки уже конечного приложения сайта программного обеспечения конечному вашему пользователю.

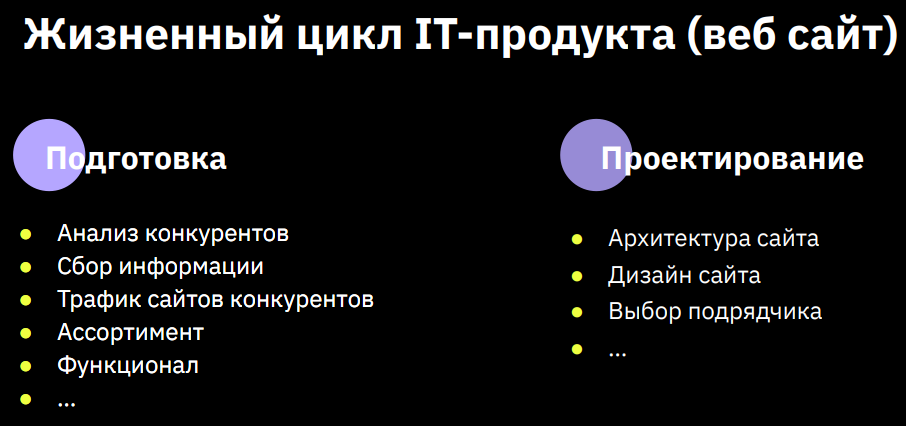


Основные этапы жизненного цикла продукта — это, во-первых, подготовка, когда рождается у вас бизнес-идея, когда вы в целом формируете визуальный образ того, как будет выглядеть это программное обеспечение, ваше приложение, ваш интернет-сайт и так далее. Неважно, что это. Затем вы плавно переходите на процесс проектирования, уже создание, то есть разработка, и у вас получается готовый IT-продукт, который необходимо поддерживать, потому что если вы его запустите, то всё равно будут возникать какие-то баги, какие-то проблемы. Сфера в целом растет, модернизируется специфика продаж, модернизируется IT-сфера, поэтому ваш любой конечный продукт необходимо поддерживать, например, жизненного цикла на вашем продукте в случае, если это веб-сайт для онлайн-магазина вашей одежды.  
Что происходит вы находитесь в самом начале на первом этапе на этапе подготовки. Вы анализируйте конкурентов, вы собираете всю необходимую информацию. Вы полностью мониторите сайт, трафик сайтов ваших конкурентов, смотрите, какой там ассортимент, какой функционал данного сайта. Затем, когда вы уже имеете полную базу всей необходимой исходной информации, как делать, а как делать не нужно, вы переходите на этап проектирования. Вы нанимаете компанию, которая разрабатывает архитектуру сайта, дизайн сайта. Выбирайте подрядчиков специализированных, которые в этом понимают и, грубо говоря, очень хорошо шарят. Затем, когда вам разработали полностью все видение, как будет выглядеть ваш интернет-магазин, вы, естественно, приступайте к его созданию, выбираете компанию-разработчика, которая пишет код, который обрисовывает уже дизайн по готовым материалам. Составляется определенная документация, и вы получаете конечный продукт. Открываете интернет-сайт, и кажется, что все очень круто и все замечательно, все работает, все функционирует. Затем, естественно, вы нанимаете команду программистов, которые будут осуществлять поддержку этого интернет-сайтом, исправлять баги, наполнять его функционалом, расширять ассортимент, учитывать мнение и обратную связь от конечных пользователей, от тех самых покупателей вашей одежды, и, соответственно, все это внедрять в ваш интернет-магазин на ваш сайт грубо говоря. И здесь возникает явный вопрос: как все это связано с нашей темой курса? Ведь мы говорим про методологии, мы не говорим про развитие бизнеса, про развитие какого-то интернет-магазина, как нам продавать одежду, как выходить на рынок, мы говорим про методологии. И так вот, чтобы создать и поддерживать ваш интернет-магазин, ваш IT-продукт, вам необходимо выбрать модель и методологию. Во-первых, для разработки, а во-вторых, для управления этим IT-проектом в дальнейшем, то есть вам в первую очередь необходимо выстроить процессы, чтобы ваше в начале разработка этого интернет-магазина проходил очень гладко, и вы получили именно то, чего вы хотите, именно то, что вы запланировали, а затем уже планово, последовательно поддерживали этот самый IT-продукт. Так вот, для этого есть две интересные составляющие: это модель и методология разработки. Пойдем постепенно, что такое модель? Модель. По сути, она описывает, какие стадии жизненного цикла проходят ваш продукт и что с ним происходит на каждой из этих стадий. Методология — это уже более практически применимый инструмент, то есть это целый набор методов в как разработать, как управлять этой разработкой. То есть те самые правила техники и принципы, которые заложены в определенной методологии и которые позволяют вам достичь конечного результата. Теперь давайте поговорим, как происходила эволюция всех этих самых моделей и методологий разработки программного обеспечения любого IT-продукта, будь то приложение, сайт, неважно что. Но как все это зарождалось? Все это началось в тридцатые годы двадцатого века. Естественно, в то время большими бюджетами обладали две основные отрасли, и все это появлялось, конечно же, в Соединенных Штатах Америки. Это были крупные компании, такие как User Corporation и Action Mobile. В этих компаниях по мере разработки каких-то IT-продуктов программисты понимали, что нужно как-то этот процесс упорядочить, выработать определенные методологии, как всем этим управлять, чтобы получать качественный продукт в сроки установленный бюджет и, соответственно, радовать конечного пользователя. Затем, в конце шестидесятых — начале семидесятых годов появляются первые предпосылки для внедрения принципов управления разработкой. Почему это произошло? Почему все-таки люди в конце семидесятых годов пришли к этому? Возник первый кризис программного обеспечения, а затем поменялись взгляды на разработку софта. Почему? Откуда возник этот самый первый кризис разработки программного обеспечения? На конференции НАТО «Инженерия программного обеспечения» в 1968 году были подняты следующие проблемы: стоимость разработки любой программы начала приближаться к стоимости аппаратного обеспечения, серверов, то есть стало очень сильно расти в цене. Стоимость любых IT-проектов начала превышать бюджет, который изначально был заложен под них. Программное обеспечение имело чрезвычайно низкое качество, то есть присутствовали какие-то баги, проблемы с кодом и так далее. Разработанные программы не очень удовлетворяли конечных пользователей, а именно заказчиков, либо, конечно, уже потребителей, которые уже ими пользовались. Соответственно, наше программное обеспечение не соответствует заявленным требованиям, возникают трудности с поддержкой кода, с поддержкой в целом IT-продукта. Именно поэтому люди решили изменить взгляды на разработку программного обеспечения. Появились отдельные научные области знаний, появились стандарты и регламенты, которые упорядочивают и вбирают в себя лучшие практики по разработке и управлению. Именно так начали зарождаться модели разработки. Теперь давайте посмотрим на их классификации?? Есть следующие модели разработки программного обеспечения: это Waterfall модель каскадная еще ее называют, в-образная модель, спиральная модель, инкрементная, итеративная и гибкие модели и методологии.

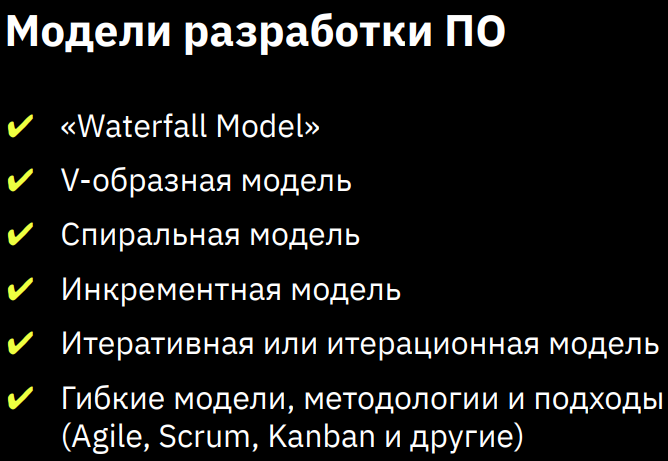
Как вы видите на данном слайде, первые 5 моделей — это больше твердые, жесткие модели управления разработкой. Они, естественно, постоянно видоизменялись, модернизировалиcь, что-то в них дополнялось, что-то дорабатывалось. Но тем не менее получалось все равно не то, к чему стремились люди, к чему стремились именно программисты, разработчики и управленцы, которые взаимодействовали с этим IT-продуктом. Именно поэтому начали появляться гибкие модели. Давайте смотреть по порядку, что это за модели и как люди постепенно приходили к тому пониманию, что нужен гибкий подход. Для начала Waterfall модель — каскадная модель еще ее называют «водопад». Почему именно водопад? Чуть позже мы про это поговорим, потому что она очень сильно похожа на водопад, и вы это поймете. Как появился Waterfall? Впервые данная модель была описана в 1970 году в статье Уинстона Ройса, американский ученый в области информатики. Он был пионером в области разработки любого программного обеспечения, соответственно Waterfall. Так как она появилась самой первой, она является традиционной и самой старейшей моделью управления разработкой программного обеспечения. Что она нам предлагает, во-первых, она предлагает последовательное выполнение всех фаз проекта. Основные фазы проекта — это инициирование, планирование, реализация и завершение проектов. Затем мы уже после завершения проекта передаем наш готовый IT-продукт в поддержку. То есть здесь мы видим, что только в самом конце мы можем увидеть готовый продукт и посмотреть, насколько он работоспособен. Соответственно долгое время данная модель рассматривалась как основной способ регулярной разработки и поставки программного обеспечения. Конечно же, ведь она была самой первой в семидесятых-восьмидесятых годах двадцатого века, и эта модель даже была принята Министерством обороны США как стандарт для разработки программного обеспечения в оборонной отрасли. Затем на основании данной модели был разработан стандарт, который называется Pimbook Project Mangement Bodi of nows, то есть такой своеобразный свод знаний об управлении проектами. Разрабатывал этот стандарт, кстати, Project Mangement также американский институт по управлению проектами, который является такой своеобразной площадкой, где происходит взаимодействие коммуникации между различными профессионалами в области управления разработкой программного обеспечения. Теперь давайте посмотрим, как в целом строится данная модель, Waterfall. Из чего она состоит?





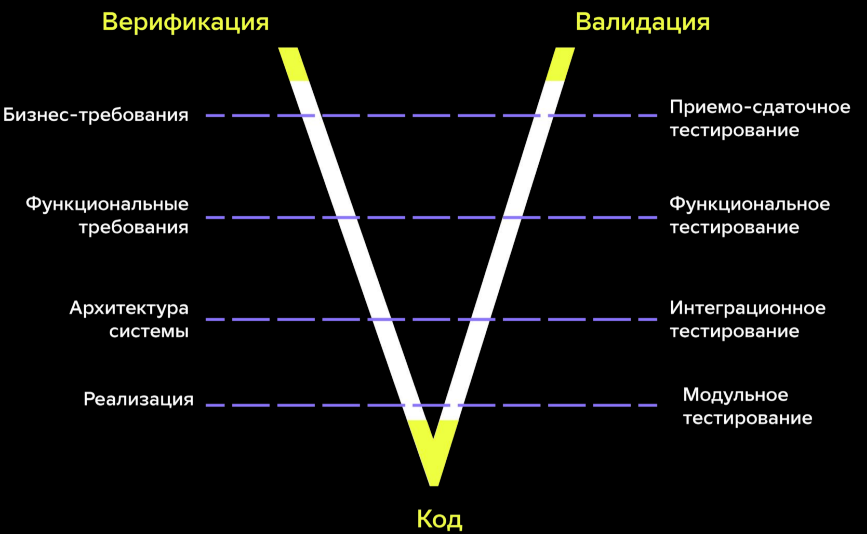


Здесь вы видите на экране цикл разработки программного обеспечения. В самом начале идет создание прототипа, затем мы создаем дизайн. После того как готов дизайн, мы начинаем писать код. После того как готов код, у нас готов программный продукт. Соответственно, мы переходим к стадии тестирования его и после того, если все удачно, все замечательно, мы все проверили, протестировали, исправили необходимые баги, мы отдаем наш продукт в поддержку. Его уже видят конечный пользователи, его можно использовать, и набирается команда программистов, которая осуществляет поддержку этого продукта.

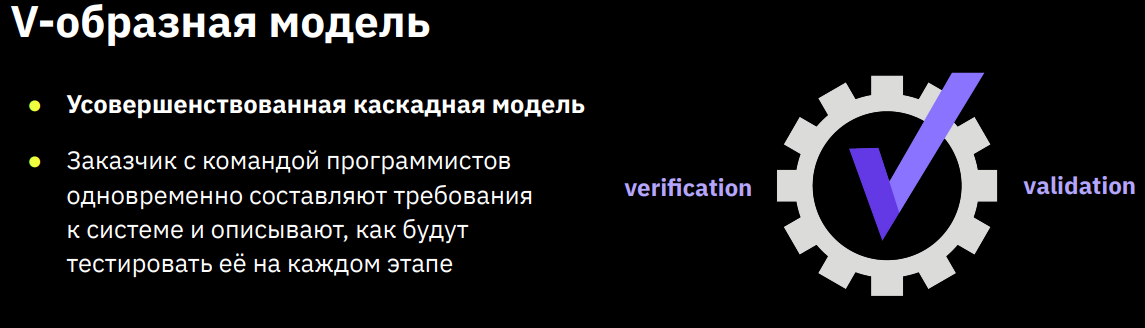


Какие здесь есть преимущества и недостатки? Казалось бы, что все замечательно, вы постепенно проходите весь цикл разработки, получается продукт, тестируете его и понимаете, что это именно то, к чему вы стремились. Но тем не менее, данная модель обладает рядом недостатков и преимуществ. Конечно, она была самая первоначальная. Естественно, невозможно было предусмотреть все именно тогда. Какие есть преимущества у Waterfall? Во-первых, разработку легко контролировать, потому что мы выстраиваем весь цикл управления по стадиям и, соответственно, закончив одну стадию, переходим на следующую. Стоимость разработки определяется в самом начале и, соответственно, у нас есть жесткий утвержденный бюджет проекта, который мы не можем превысить. Всё вроде бы замечательно. Нам не нужно нанимать кучу тестировщиков, которые будут обладать серьезной технической подготовкой, потому что мы продукт тестируем в самом конце, то постоянно его разрабатываем, дорабатываем. Но протестировать можем только тогда, когда получается готовый IT-продукт, а это происходит практически на самых финальных стадиях разработки. Соответственно, исходя из этого у нас появляются следующие недостатки данной модели? Первый недостаток — это то, что тестирование начинается на последних этапах разработки. Значит, наш заказчик видит готовый пойти продукт только в самом конце разработки. Только тогда может дать нам какую-то обратную связь. Насколько мы правильно сделали то, что он изначально прописал в техническом задании. Разработчики пишут очень много технической документации в самом начале, то есть достаточно скрупулезно работают над техническим заданием для разработки. Это сильно задерживают сроки. Плюс надо понимать, что IT-продукт, любое приложение, сайт и так далее, это своеобразный живой продукт, который невозможно изначально предсказать и понять, как он должен выглядеть в конце. В самом начале мы формируем только лишь образ и вот именно данная методология, данная модель разработки программного обеспечения Waterfall нам позволяет гибко двигаться по процессу. Мы должны в самом начале уже четко понимать, что у нас должно получиться в конце, то есть мы должны достаточно подробно скрупулезно проработать техническое задание, что, к сожалению, сделать в самом начале не всегда представляется возможным. Соответственно, Waterfall— это не совсем плохая методология, не совсем плохая модель для разработки IT-продукта. Нет ни в коем случае это не так? Почему. Потому что, во-первых, Waterfall нам дает отличный результат только в проектах, где есть четко заранее определенные требования. На самом деле таких отраслей очень много, и чуть позже в качестве примера мы про них поговорим. Соответственно, в Waterfall в противовес первому его преимуществу, невозможно сделать шаг назад, то есть невозможно откатиться и проверить свою ошибку, потому что тестирование начинается только после того, когда у нас уже готов продукт, то есть готов сайт, готовое приложение, и только тогда мы можем, протестировать, увидеть свои ошибки, а значит, ошибки в требованиях начинают сильно возрастать в цене, потому что у нас уже готов полностью весь код. У нас готовый продукт, и чтобы откатиться назад и что-то исправить, нам придется перелопачивать большую часть данного кода переделывать большую часть нашего продукта. Нельзя скорректировать список требований к продукту в процессе его разработки, потому что элементарно у нас отсутствует промежуточное какое-то тестирование. Соответственно, когда же можно применять эту самую модель в разработке. Во-первых, когда у нас известны требования, когда они понятны, четко зафиксированы, и мы не можем от них отступать, когда нет противоречивых требований к функционалу продукта. То есть у нас есть заказчик, который четко понимает, что он хочет, он уже видит в голове образ нашего продукта. У нас нет проблем с доступностью программистов, нужной квалификации, их очень много, у нас есть достаточный бюджет, чтобы их привлекать по поэтому мы можем позволять себе допускать такие ошибки. И Waterfall в основном применяется в относительно небольших проектах. Возникает вопрос, в каких отраслях можно применить Waterfall? Три ярких примера таких отраслей — это медицинская, космической отрасль и строительство. Касательно говоря, медицинской отрасли, там есть четкие регламенты, четкое понимание, четкие требования к конечному IT-продукту, то есть определенный ряд законодательных актов, которые полностью все это регламентируют. Именно эти законодательные акты, регламенты позволяют нам создать правильный продукт и не допускать каких-то ошибок, потому что у нас есть уже изначально прописанные четкие технические требования. Если мы говорим про космическую отрасль, она в целом очень похожа на медицинскую. Там допускать ошибку невозможно. Там есть четкие регламенты, научные разработки, и это все уже известно, практически. Если мы говорим про стройку, естественно, построить какое-то здание или сооружение, не применяя гибкую методологию, нельзя. То есть невозможно какими-то итерациями небольшими возрастить большое офисное здание. Мы должны сначала вырыть котлован, полностью сделать фундамент, потом нарастить коробку, потом ее отделать, потом завершить крышу. И тог далее, то есть у нас происходит водопадная модель разработки продукта. Казалось бы, отрасли то очень схожие практически, потому что и там, и там применяется эта самая модель разработки. Главная проблема — недостаток Waterfall, в чем она заключается? В самом начале на старте разработки мы должны достаточно четко и подробно прописать все требования к данной разработке продукта. На этапе тестирования, когда у нас уже получится готовый продукт, у нас не должно возникнуть то обстоятельство, что мы допустили где-то ошибку, иначе это будет очень сильно бить по нашему бюджету проекта. Соответственно, подводя итоги по модели Waterfall, можно отметить самые ярко выраженные недостатки и преимущества данной модели. Из недостатков, во-первых, это сильный рост стоимости незапланированных изменений, то есть ошибка очень дорого нам будет стоить в конце соответственно, исходя из этого у нас происходит рост количества проблем в процессе проектирования по мере того, как будет расти наш проект потому что мы должны все достаточно четко, детально и структурно прописать в самом начале. Тем не менее, Waterfallобладает рядом преимуществ. Например, старт реализации проекта должен быть отложен до выяснения всех технических моментов проекта до выяснения всех целей задач, то есть пока мы не получим действительно качественное техническое задание, мы не можем дать старт проекту. Да, это преимущество, потому что мы уже в самом начале можем четко говорить о том, что в конце получит заказчик. Разработка проекта, точнее говоря, продукта проекта должна быть очень хорошо скоординирована, должна подчиняться разумному планированию и управлению и в целом управлением проектом по методологии Waterfall— достаточно легкий процесс по сравнению, допустим, с гибкими методологиями. Именно поэтому зачастую неопытные управленцы, неопытные руководители разработкой, выбирают эту модель, потому что она очень понятна она позволяет по процессу по стадийно выстроить разработку и получить продукт. Но они забывают о том, что ошибка в конце. Если она будет выявлена, она будет очень и очень дорого стоить нам. Теперь давайте перейдем к следующей модели, это в-образная модель, то есть мы поговорили про Waterfall мы обсудили его основные преимущества и недостатки. Тем не менее, эти отрасли не стоят на месте. Люди понимают, что Waterfall обладает одним из серьезных недостатков — это отсутствие тестирования в процессе разработки. Именно поэтому принимают решение о том, чтобы модернизировать эту модель и разработать следующую. В-образную модель, которая по сути дела называется «разработкой через тестирование». Что это такое? Это усовершенствованная каскадная модель, водопадная. Заказчик с командой программистов одновременно составляют требования к системе, при этом описывают, как будет тестироваться данная система на каждом этапе разработки. То есть главный принцип — это верификейшн и валидейшн.

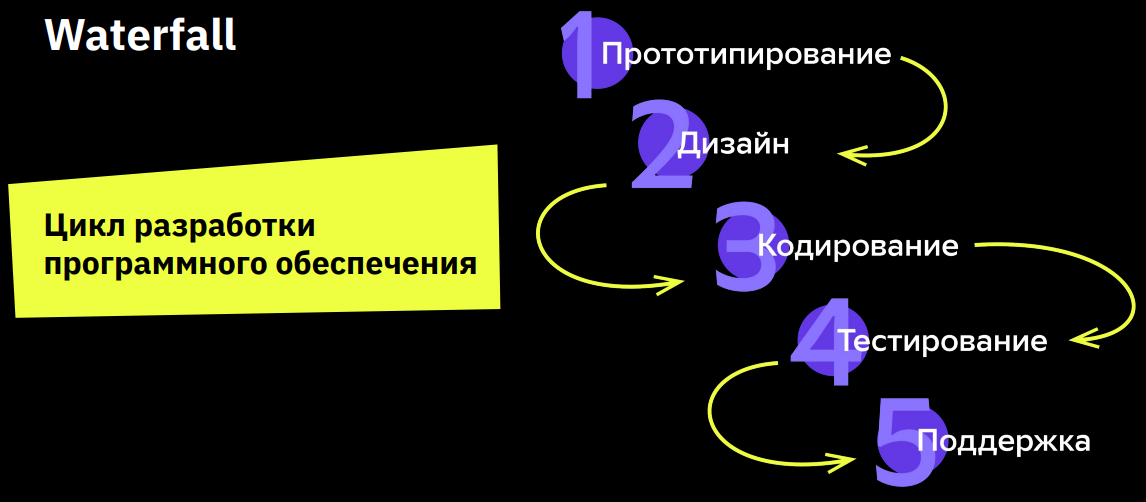
Здесь вы видите, это, можно сказать, наглядный рисунок, наглядность — схематическое представление того, как выглядит в-образная модель разработки. Давайте поговорим подробнее, что включает в себя данная в-образная модель. Какие особенности? Во-первых, в-образная модель направлена на тщательную проверку и тестирование продукта, которая находится уже на самых первоначальных стадиях проектирования. То есть мы разрабатываем постепенно и тестируем также постепенно. Не ждем, пока закончится наша разработка, как пока мы получим конечный продукт и начнем тестировать. Стадия тестирования проводится одновременно соответствующей стадией разработки. Что очень серьезно помогает избежать каких-то ошибок. Во время кодирования сразу же пишутся модульные тесты, при этом в-образная модель тоже не является панацеей. Она в основном применима к системам, которым особенно важно бесперебойное функционирование. Есть два ярких примера таких систем: это прикладные программы в клиниках для наблюдения за пациентами, потому что там допустить какую-то ошибку в разработке кода сможет стоить жизни пациентов, а также интегрированное программное обеспечение для механизмов управления аварийными подушками безопасности в наших автомобилях, потому что там такая ошибка может очень и очень дорого стоить любому автовладельцу. Именно поэтому при разработке IT-продуктов, связанных именно с этими двумя примерами можно применять в-образную модель.  
Давайте посмотрим, какими преимуществами и недостатками она обладает? Преимущество количество ошибок в архитектуре программного обеспечения у нас сводится к минимуму, потому что мы постепенно по мере разработки, тестируем то, что сделали, пишем модульные тесты. Конечно же, она вбирает в себя ряд преимуществ, про которые мы говорили, когда обсуждали методологию Waterfall. При этом есть ряд недостатков: если при разработке архитектуры была допущена ошибка, то вернуться и исправить ее будет все равно стоит также дорого, как Waterfall. Потому что наш продукт не разделяем на какие-то части, мы делаем общий большой продукт. По мере того, как пишем какой-то код, мы его просто тестируем и не смотрим на то, как этот код будет взаимодействовать с другим программным обеспечением, с другими кодами. Мы просто тестируем отдельно взятую разработку, которую мы делаем здесь и сейчас. И это именно и стоит этих денег, и стоит дорого. Когда же можно все-таки использовать в-образную модель, если требуется тщательное тестирование продукта, то есть верификация и валидация главные принципы в модели применяется в модель для малых и средних проектов, где требования четко определены, фиксированы.



Как вы видите, она очень похожа на Waterfall. Ее лишь всего лишь добавили. Да вот момент тестирования в условиях доступности инженеров необходимой квалификации, особенно тестировщиков. То есть, по сути, в-модель — это та же каскадная модель с элементом тестирования кода, она все равно еще не настолько совершенна, как бы этого нам хотелось видеть. Соответственно, люди понимают, что этого недостаточно. Прогресс идет вперед, мы двигаемся вперед, и поэтому нам нужно вырабатывать какие-то новые подходы, новые модели методологии для разработки этих продуктов. Следующая модель, про которую хотелось бы рассказать, это спиральная модель. По сути дела, она возникает уже после того, как люди обнаружили недостатки в-образной модели. Что происходит здесь? Продукт разрабатывается версией.



При этом сейчас на слайде вы видите, как выглядит спиральная модель, как выглядит в целом разработка по этой модели. Давайте смотреть подробнее поэтапно. Во-первых, история ее возникновения, впервые она была описана в 1988 году Барри Боймом, американским инженером-программистом, профессором программной инженерии. Описал он ее в одной из своих статей в американском научном журнале. Что она в себя включает? Что он вообще подразумевал, когда говорил про спиральную модель? Во-первых, работы над проектом представляются как цикл, как спираль. То, что вы видите сейчас в правом углу нашей презентации. Каждый виток спирали, по сути своей является водопадной моделью. То есть все равно мы все еще крутимся вокруг Waterfall, мы еще никак не можем от него отойти, мы его модернизируем, смотрим, какие-недостатки, улучшаем их, но все же пока еще около Waterfall. Как происходит разработка по спиральной модели? Во-первых, у нас формируется требование, затем происходит анализ рисков этих самых требований, затем конструирования, проектирования, кодирования данной системы, и после этого мы уже запускаем ее в эксплуатацию. Так сказать, происходит оценка заказчикам того, что у нас получилось. Вроде бы выглядит все хорошо. У нас есть первая версия продукта, вторая версия, третья версия. Но тем не менее у нас при этом есть большой риск застрять на этапе анализа рисков, анализа требований к нашему IT-продукту. Давайте обсудим особенности спиральной модели и, соответственно, нам будет видно, почему возникает такой риск? Цикл разработки начинается с этапа сбора требований к предполагаемым изменениям в рамках реализации нашего продукта и завершается уже непосредственно стадией реализации. То есть решается основная проблема традиционных моделей о том, что невозможно изменить какие-то требования к продукту. Заказчик и команда разработки очень серьезно анализирует риски проекта. То есть в самом начале нужно понять, действительно ли нужен будет конечный продукт, какие могут возникнуть сложности при его реализации при его разработке, кодированию, и только после этого уже приступить к разработке. Именно поэтому они выполняют данный проект итерации, делают определенные версии продуктов поэтапно. Последующая стадия основывается на предыдущей, то есть только после того, как мы завершили допустим, разработку первой версии, мы должны проанализировать и сделать выводы по своим ошибкам, исправить их, затем проанализировать риски старта второй версии и только после этого приступить к ее разработке. Соответственно, в конце каждого витка нашей спирали принимается решение о том, стоит ли продолжать проект или лучше его закончить.



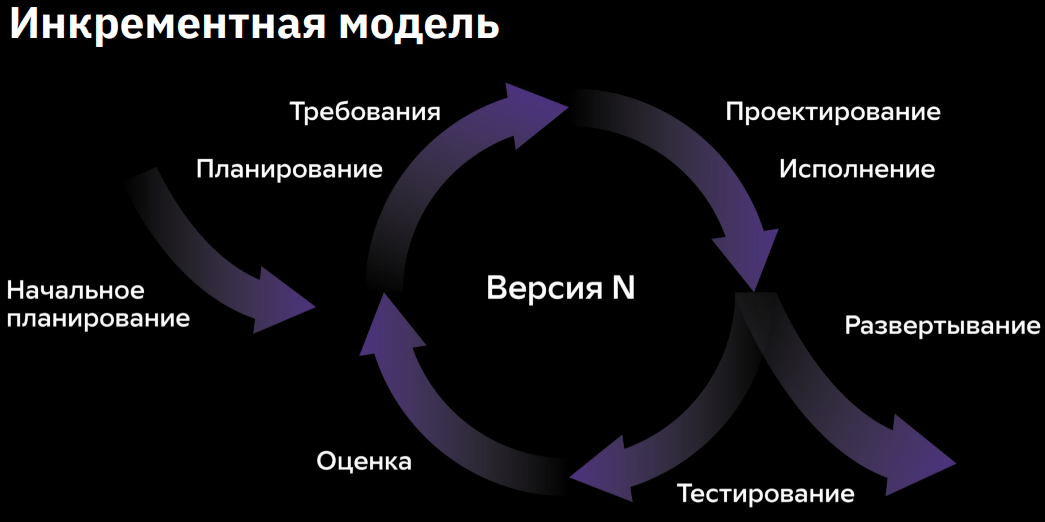
Здесь хотелось бы привести яркий пример спиральной модели, это «умный дом». Допустим, вы заказчик, который захотел разработать систему «умный дом». Что вы делаете? Вы, как заказчик, заказывайте программистам реализовать управление чайником с телефона, одна из частей вашего «умного дома». Программисты начинают действовать по модели «водопад». Они выслушали вашу идею, они проводят анализ предложенных на рынке каких-то инструментов по реализации данного решения, обсуждают с вами как с заказчиком архитектуру системы, решают, как будут ее реализовывать, разработали, написали коды, протестировали и выкатили вам конечный продукт. Затем заказчик у нас оценивает результат, оценивает риски необходимости следующей версии продукта. И решает создать следующую версию, потому что риски оценил, все замечательно, и ему кажется, что все будет супер, и следующая версия тоже выстроена. Что он хочет сделать? Он хочет управлять уже с телефона телевизором. Снова происходит расчет сроков бюджета, снова он заказывает разработку. Программисты начинают использовать снова каскадную модель и представляют заказчику более сложный продукт, разработанный на базе первого. Заказчик принимает продукт, понимает, что ему еще необходимо третью версию продукта. У него возникает такое желание, и он уже хочет создать управление с телефона холодильником с помощью wi-fi. Казалось бы, очень круто звучит. У нас уже третий виток нашей спирали. На каждом витке, не забывайте, применяется каскадная методология, и вот здесь заказчик начинает анализировать риски. Он начинает изучать те самые предложения, которые есть на рынке по техническому внедрению, понимает, что в холодильник очень сложно встроить вай-фай модуль, и производители в целом, с кем он разговаривал, не заинтересованы в сотрудничестве. Они не хотят это делать, им это невыгодно, грубо говоря, тут мы понимаем, что риски для заказчика превысили потенциальную выгоду от разработки третьей версии, третьего витка спирали в его IT-продукте, и он принимает решение приостановить разработку и уже совершенствовать то, что у него есть те самые версии, которые были сделаны до этого. На следующий на третий виток он не переходит, потому что, все слишком долго оценивал риски, слишком много рисков увидел и все-таки принял решение не рисковать, потому что зачем разрабатывать уже систему, которая есть сейчас.



Исходя из этого яркого примера того, что мы говорили про спиральную модель, можно выделить следующие преимущества и недостатки этой модели. Во-первых, преимущество уделяется большое внимание проработке рисков, а значит, что мы анализируем все решения, все ошибки и возможность допустить ошибку у нас сводится практически к минимуму. Тем не менее она все равно присутствует, потому что каждый виток спирали делается по модели каскад, но есть ряд и недостатков. Есть большой риск застрять в самом начале, как это в нашем примере было с нашим заказчиком, то есть он долго анализировал риски, к нему никто не шел навстречу, из производителей холодильника не внедряли вай-фай модуль, и поэтому он все-таки решил не идти дальше. Разработка длится достаточно долго, стоит она также дорого, потому что у нас все еще тот же подход к каскадной модели. Мы все также постепенно поэтапно разрабатываем продукт.



Следующей после в-образной модели в своей эволюции была инкрементная модель. Давайте посмотрим, что это такое и какие основные параметры включают в себя данная модель разработки программного обеспечения. Во-первых, полные требования к программному обеспечению делятся на различные сборки, поэтапная сборка ПО. Что мы видим, у нас появляется несколько циклов разработки, которые вместе составляют жизненный цикл мульти-водопад, то есть все равно мы еще недалеко уходим от нашей каскадной модели и цикл разделяется на более мелкие, легко создаваемые модули. Каждый модуль у нас проходит через фазы определения требований, проектирования, кодирования, то есть реализация программного обеспечения, внедрения и его тестирования. Данная модель предполагает, что на первом большом этапе разработки продукта у нас уже появляется базовый функционал, то есть у нас есть какая-то небольшая работоспособная версия, которая обладает базовым функционалом работы. Затем уже происходит последовательное добавление новых функций. Они еще называются «инкременты». Соответственно, процесс разработки продолжается до тех пор, пока не будет создана полная система программного обеспечения.



Здесь сейчас на слайде вы видите графическое представление того, как работает инкрементная модель, то есть происходит начальное планирование, вырабатываются определенные требования к нашему программному обеспечению, происходит стадия проектирования исполнения, и затем уже тестирование, и развертывание данной системы в рамках бизнес-среды, в которой предполагается создать это программное обеспечение.

Теперь посмотрим, например, потому что теория — это хорошо, но все же лучше смотреть с точки зрения практического применения и посмотреть, как это работает, как же это будет наглядно выглядеть? Но возьмем в качестве примера проект по созданию социальной сети своеобразный IT-продукт. Что у нас происходит? Заказчик пишет достаточно детальное подробное техническое задание, как он представляет себе свою новую социальную сеть, нанимает разработчиков-программистов, которые, видя всю сложность данного проекта, предлагают заказчику реализовать основные функции данной социальной сети, затем протестировать эти функции на пользователях, посмотреть, взлетит или не взлетит, и только после этого уже ее дорабатывать. Соответственно, программисты разрабатывают ту самую минимальную версию нашего продукта, с ее базовым функционалом запускают ее на рынок, если пользователям соцсеть нравится, то происходит дальнейшая доработка ее, то есть работа над ней продолжается как над целым проектом, но уже по частям. То есть наши разработчики-программисты постепенно создают функциональность для загрузки фотографий, обмена документами, прослушивание музыки, добавление этой музыки в свой аудиоплеер и различные другие действия, которые позволят достаточно полноценно использовать данную социальную сеть. Здесь мы видим, что наш продукт совершенствуется постепенно, то есть инкремент за инкрементом и в целом приближается к изначально прописанному достаточно скрупулезно техническому заданию заказчика. Соответственно, данное техническое задание в данной модели инкрементной может претерпевать какие-то изменения. То есть здесь мы видим, ошибки все-таки возможны, и цена этих ошибок уже снижается по стоимости, что очень важный момент и показывает отличительную особенность инкрементной модели от всех предыдущих и, в частности, от каскадной модели. Еще хотелось бы привести пример, который я приводил чуть раньше, с использованием другой модели разработки программного обеспечения. Это все тот же самый умный дом. Помните, у нас заказчик захотел разработать такой пойти продукт, чтобы управлять с телефона своим домом. Соответственно, как бы это происходило, если бы использовали инкрементную модель, а не спиральную, как в предыдущем примере. Заказчик пишет достаточно подробное техническое задание к своему новому IT-продукту. Все замечательно, он прописывает весь функционал, программисты смотрят это техническое задание и предлагают ему реализовать основные функции. То есть сначала чтобы можно было управлять чайником с помощью телефона, затем запустить это, так сказать, в какой-нибудь магазин приложений, чтобы пользователи могли его скачать, протестировать, посмотреть и дать какой-то фидбэк по данному IT-продукту, посмотреть, таким образом, взлетит это или нет, стоит ли двигаться дальше или нет? Здесь мы видим, что в самом начале отсутствует какой-то процесс по достаточно детальному анализу всех рисков как, мы это видели в спиральной модели, а соответственно, мы постепенно прощупываем почву и смотрим, подходит пользователям наш IT-продукт или нет, продукт разработан, запускается на рынок. Допустим, что пользователям система нравится, в таком случае продолжается работа над этой системой, но уже по частям. Программисты параллельно создают функционал для управления телевизором с телефона, холодильником, кондиционером, системой отопления, системой охлаждения какими-то другими бытовыми приборами, которые находятся внутри вашего дома. Соответственно, здесь ярко видно, что наш продукт дорабатывается, инкремент за инкрементом, приближаясь к нашему изначально созданному техническому заданию. При этом у отсутствует тот момент, что ввиду достаточно детального обзора, анализа всех рисков заказчик может отказаться от своей идеи, от своей задумки. То есть он идет на рынок, тестирует постепенно инкремент за инкрементом, общий свой IT-продукт, работает это или нет, нравится это пользователям или нет. Давайте отметим основные преимущества и недостатки такой инкрементной модели? Во-первых, преимущество в самом начале нам не требуется вложение достаточно большие не требуется вкладывать много денег в разработку, потому что мы идем постепенно. Можно получить сразу же фидбэк от конечных пользователей и при необходимости скорректировать техническое задание. Все замечательно, у нас снижается риск создать продукт, который никому не нужен и в данном случае, когда мы применяем инкрементную модель, наша ошибка стоит гораздо дешевле. Тем не менее вроде все замечательно, но инкрементная модель также обладает рядом недостатков. Что это такое? Учитывая, что мы действуем постепенно разрабатываем инкремент за инкрементом, над продуктом работают несколько команд разработчиков, соответственно, каждый из них видит свою задачу и в целом продукт по-своему, а значит, может произойти рассинхронизация этих людей, разработчики зачастую оттягивают доработку основной функциональности. Они пилят какие-то небольшие коды, какую-то мелочевку, которая в целом не представляет, той самой ценности для завершения данного инкремента. Соответственно, инкрементная модель подходит для проектов, в которых достаточно точное условие уже прописано на старте. То есть все равно мы видим, что заказчик в самом начале должен видеть, что он получит, в конце, а уже опытные программисты, разработчики, руководители данных проектов, они его структурируют на инструменты. При этом мы видим, что наш продукт достаточно быстро выходит на рынок, то есть первая работоспособная версия с минимальным базовым функционалом уже появляется у конечных пользователей. Как бороться с этими недостатками? Здесь важно отметить заслуги именно опытных управленцев разработкой и, соответственно, управленцев реализации данного IT-проекта. Не всегда получается найти таких опытных людей, которые могут достаточно хорошо организовать работу над продуктом с помощью инкрементной модели. Следующая наша модель, которая уже постепенно двигается в сторону гибкого подхода.

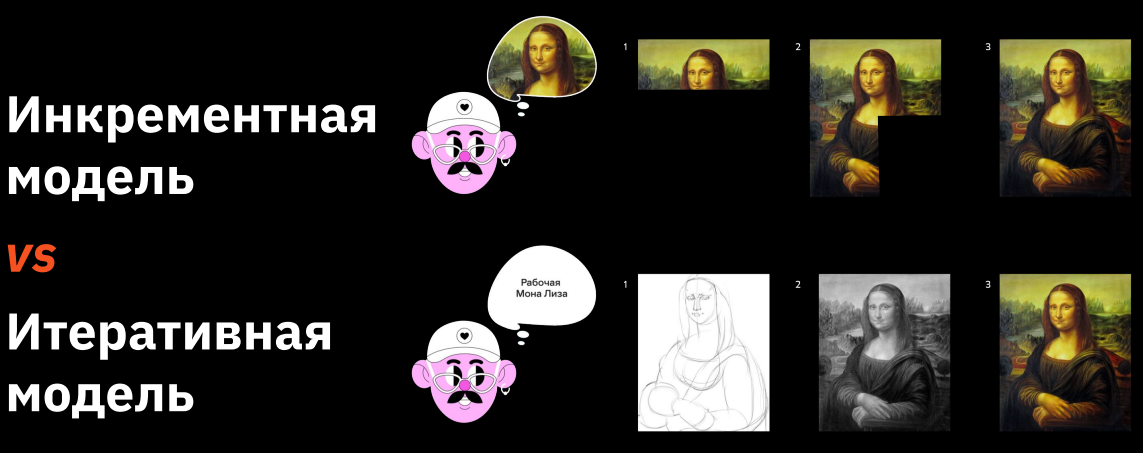


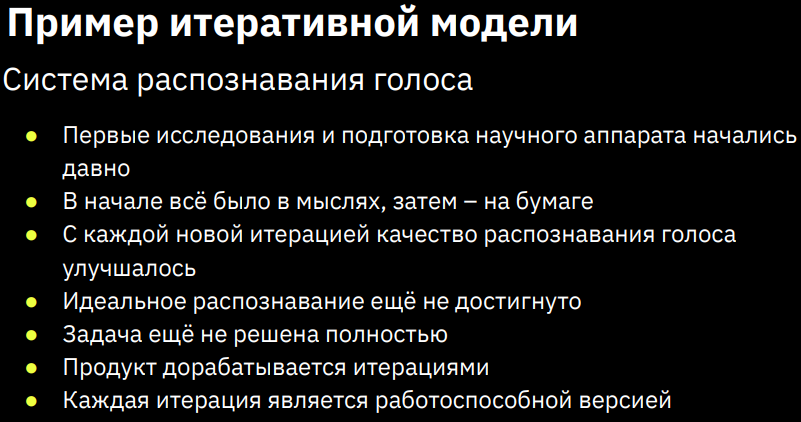
Мы видим, что инкрементная модель все равно не такая универсальная, она обладает рядом недостатков, рядом преимуществ. Но тем не менее IT-отрасль не стоит на месте. Мы все тихо эволюционируем и двигаемся в сторону итеративной модели. Что это такое? Итеративная модель, еще и вызывает итерационная, по своей сути, наша разработка программного обеспечения представляется как последовательность итераций.

Здесь вы видите рисунок, то есть сначала мы рисуем квадратик, потом добавляем кругляшок, потом хобот, потом его штрихуем. То есть постепенно сначала делаем основные начертания, набрасываем эскиз, потом его дорабатываем, и у нас уже потихоньку создается представление того, что мы в конце хотим увидеть. Это в данном случае слон. Каждая итерация сама по себе является небольшим проектом. В рамках общей нашей глобальной задачи каждая итерация предполагает создание измеримого прироста ценности продукта по завершению итерации. По своей сути данная модель дает основу и является началом для применения гибких методологий. То есть после того, как люди начали применять итеративную модель, они уже начали смотреть в сторону гибкого подхода разработки. Они видят, что не обязательно раскладывать полностью весь свой проект по каскадной модели, выстраивать определенный цикл разработки программного обеспечения, а можно действовать постепенно, то есть итерация за итерацией, доработка за доработкой. Соответственно, в данном случае фокус уже начинает смещаться с момента, когда мы хотели обеспечить полноту требований к нашему IT-продукту в самом начале в сторону формирования процессов, слаженной работы команд. Здесь уже больше уклон идет в сторону людей, в сторону тех самых специалистов, которые разрабатывают этот IT-продукт, потому что в конце все зависит именно от них, от их хардскелов, от их softskillof от того, как они взаимодействуют, как они доносят ту самую ценность до своего заказчика. Но как же люди пришли к этой итеративной модели, как она начала зарождаться, откуда это все появилось? И здесь, хотелось бы отметить, что еще в тридцатых годах в своих статьях специалист по проблемам качества продукции Уолтер Шевард из компании Bell Labs начал потихоньку описывать данный подход. Показательный пример эффективности данной итеративной модели считают реализованный в пятидесятые годы проект сверхзвукового самолета x 15.0. Конечно, по своей сути данный проект вобрал не только IT-технологии, но и технологии самолетостроения, и в целом управление продуктом и управление проектом в другой сфере. Не будем углубляться, как они это сделали, как они реализовали данный самолет, потому что это немножко отходит от нашей тематики вопроса. При этом хотелось бы отметить, что участники этого проекта считают именно итерационный подход основным показателем успеха реализации их проекта. Соответственно, начинается, так сказать, эра постепенного перехода к гибкому подходу разработки.



Почему всем так понравилась и итеративная модель, почему люди начали к ней присматриваться. Во-первых, она не требует в самом начале полной спецификации требований, то есть заказчику не обязательно сразу же видеть, как будет выглядеть его продукт. Создание начинается с части какого-то функционала, то есть мы обеспечиваем базовую функциональность, запускаем, получаем фидбэк и начинаем уже работать над ней дальше. Это часть функционала, которую мы сделали в самом начале. Она становится основой и базой для определения наших последующих требований к следующим версиям продукта. То здесь мы видим в предыдущих моделях мы сразу должны были иметь четкое представление того, что мы хотим получить в конце, как будет выглядеть наш эти продукт. А здесь нам этого не нужно, мы запускаем базовый функционал, тестируем его и начинаем уже от него отталкиваться. Что мы будем дорабатывать и как мы это будем делать? Соответственно, самая первоначальная версия, она не должна быть идеальной, она может быть с какими-то проблемами, ошибками, но самое главное, чтобы она работала, чтобы ее уже можно было где-то использовать. Понимая конечную цель, мы начинаем стремиться к ней так, чтобы каждый шаг за шагом был результативнее, и при следующей операции мы получали уже усовершенствованную нашу, так сказать, минимальную модель, которую разработали в самом начале. И соответственно, каждая следующая итерация должна нам порождать работоспособную версию продуктов с дополнительным каким-то функционалом.

  
Хотелось бы привести очень яркий пример, который вам позволит оценить различия инкрементной модели и итеративной, которые, по сути, возможно, очень похожи друг на друга. Если мы посмотрим на инкрементную, помощью инкрементной модели нарисуем портрет Моны Лиза, он будет выглядеть так, то есть в самом начале заказчик уже должен представлять, как будут выглядеть черты лица Мона Лизы, во что она будет одета, какой будет пейзаж, какой будет фон, на котором она будет сидеть. Затем у нас возникает полностью детализированное техническое требование: что это должно быть? Это должен быть большой портрет, она должна быть видна вплоть до нижней части тела, то есть должны быть видны ее руки, голова, грудь и так далее? У нас инкремент за инкрементом создается портрет Мона Лизы. Сначала первый инкремент. Мы обрисовываем верхнюю часть портрета и голову ее представления, как она смотрит, взгляд и так далее. Затем рисуем нижний левый угол и потом уже правый левый угол. Как мы видим, здесь у нас могут возникнуть какие-то ошибки в начертании данного портрета. И если эта ошибка возникнет, чтобы ее исправить, нам нужно будет заново создавать инкремент за инкрементом. То есть ошибка будет очень дорого стоить, нам придется откатываться назад. Если мы говорим про создание портреты портрета Мона Лизы с помощью итеративной модели как это происходит: приходит заказчик и говорит: я хочу увидеть портрет Мона Лиза, рабочую модель, так сказать, собирается пул программистов-разработчиков предлагают выполнить данный проект с помощью оперативной модели, то есть итерации. За итерацией они обрисовывают портрет, берется чистый лист бумаги, и сразу же производится начертание — общее начертание, как выглядит эта женщина на портрете. Здесь мы видим: первая итерация — это карандашный набросок Мона Лизы на чистом белом листе бумаги. Затем начинают переводить ее в черно-белый цвет, то есть заштриховывать. И третья итерация — это уже в цвете, то есть у нас получается готовый работоспособный продукт, портрет Мона Лизы, если его можно так назвать, что хотелось бы отметить, почему здесь цена ошибки будет гораздо ниже, чем эта цена ошибки в инкрементной модели. Потому что, когда у нас рождается первая итерация в виде карандашного наброска Лиза, мы можем где-то что-то стереть поправить, потому что у нас еще не отрисован весь портрет, как если мы это делаем с помощью инкрементной модели. Затем мы это все штрихуем. Если опять же заказчику что-то не нравится, мы можем с легкостью это поправить, исправить карандашный набросок, и только после того, когда вот этот карандашный набросок в черно-белом цвете уже полностью удовлетворяет требованиям заказчика. Мы его переводим в цвет, и у нас получается готовый красивый портрет. Соответственно, по мере того как мы его разрабатываем, обрисовываем, мы можем что-то менять, наносить какие-то корректировки, исправлять свои ошибки, потому что они свойственны абсолютно всем. Это не значит, что это плохие специалисты работали над проектом, потому что не ошибается только тот, кто ничего не делает. Именно поэтому итеративная модель так стало активно в разработке IT-продуктов, в разработке программного обеспечения и в целом в управлении IT-проектами. Давайте посмотрим еще один пример и итеративной модели, уже с применением больше IT-сфере. Если там мы говорили про начертание портрета, то теперь давайте поговорим про разработку программного обеспечения. Очень ярким примером итеративной модели является создание системы распознавания голоса. Первые исследования и в целом подготовка научно-исследовательской базы научного аппарата для этой разработки началась очень давно. Вначале все это было только в мыслях, в задумках, а затем уже переходила на бумагу. При этом с каждой новой итерацией, когда разрабатывалась эта система, ее функционал улучшался. То есть мы получили сначала первую минимальную версию продукта, которая была работоспособной, которая могла распознать голос человека. Но дело она это все недостаточно хорошо. Соответственно, мы видим, что идеальное распознавание голоса еще не достигнуто, а значит, у нас еще есть много работы. Впереди итерация за итерацией мы должны улучшить наш продукт, задача у нас не решена еще полностью, и продукт дорабатывается, и даже на сегодняшний день невозможно сказать, что этот продукт в виде системы распознавания голоса идеален. Соответственно, каждая итерация является работоспособной версией этой системы по распознаванию голоса. В конце концов, когда-то мы сделаем эту систему абсолютно совершенной, и она сможет распознавать абсолютно все голоса и все языки, на которых говорят в нашем мире. Давайте посмотрим на ряд преимуществ и недостатков итеративной модели. Сначала преимущество: во-первых, мы сможем достаточно быстро выпустить MVP — минимальную версию продукта, про которую вы наверняка слышали много, где читали, смотрели различные источники и так далее. Мы сможем с помощью итеративной модели получить быструю обратную связь от заказчика нашего продукта и от конечных пользователей. Фокусировка происходит именно на наиболее важных функциональных особенностях нашего программного обеспечения, и, соответственно, по мере выполнения каждой итерации эта функциональность улучшается и дорабатывается. Происходит постоянное тестирование конечными пользователями, и это нам позволяет быстро обнаружить какие-то ошибки и устранить их. Соответственно, мы выкатываем следующую итеративную версию нашего продукта.



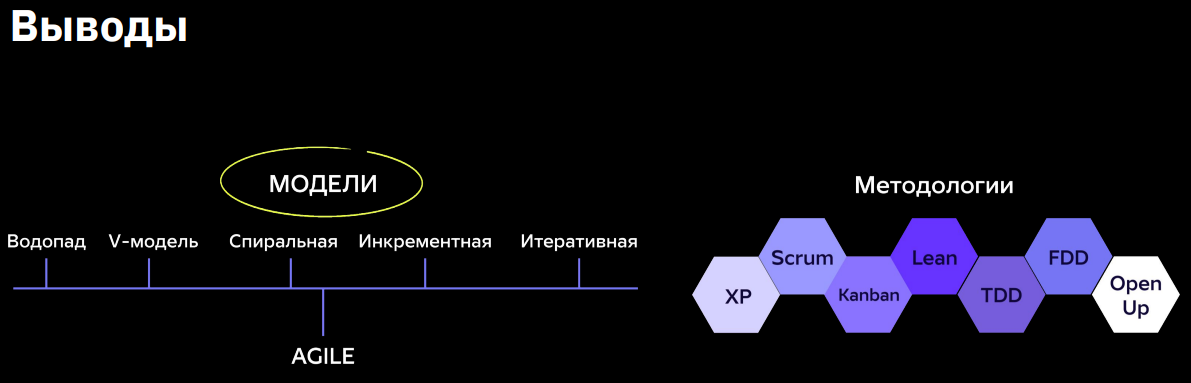
Тем не менее, есть ряд недостатков данной инкрементной модели. Во-первых, в самом начале, когда мы разрабатываем продукт, мы используем очень много баз данных, используем сервера, что это означает, базы данных сложно масштабировать. Сервера бывает, что не выдерживают нагрузку, соответственно, случаются моменты, когда те самые минимальные версии продукта, созданные в самом начале, приходится перезаписывать перекодировать большую часть этого приложения. Да, это серьезный недостаток, то при этом есть еще второй важный недостаток — это то, что у нас нет фиксированного бюджета на разработку и нет фиксированных сроков, потому что заказчик в самом начале еще не имеет представления того четкого представления, что должно получиться в конце, соответственно, и не может достаточно детально просчитать бюджет и сроки на реализацию. Какие есть особенности итеративной модели, подходит данная и итеративная модель для работы над большими проектами, когда мы еще только в самом начале пути и не можем осознать и увидеть, как должен выглядеть конечный наш продукт? Когда мы не можем в самом начале определить все технологические, технические и другие требования к нашему продукту, мы только имеем идею, и мы представляем, как может эта идея реализоваться, войти продукт в его минимальной версии самым небольшим базовым функционалом, а затем мы его запускаем, получаем фидбэк и дорабатываем. Данная и итеративная модель подходит для задач с инновационным подходом, когда заказчик еще не уверен в результате. Это своеобразные стартапы, которые мы видим на сегодняшний день, очень активно выстреливают. Развиваются и начинаются они именно создания MVP — малой версии продукта. Плавно мы подходим к тому моменту, что люди начинают понимать, что и декоративная модель — это все круто, но нужно ее еще больше модернизировать, и, соответственно, появляются гибкие методологии. Почему это происходит? Замена традиционных моделей и методологий подхода к разработке она неизбежна, потому что происходит рост этих технологий, растет в целом рынок, растет количество практических проблем в процессе разработки. И если в самом начале в тридцатые-семидесятые- восьмидесятые годы IT-продукты разрабатывались в основном для оборонки, для нефтегазовой сферы, где были большие бюджеты, где внедрялись новые инновационные технологии, то сейчас в современном мире IT проникла во все сферы нашей жизнедеятельности, и уже те подходы, которые применялись тогда, не подходят сейчас. Важно двигаться в ногу со временем и, соответственно, внедрять новые подходы в разработке.

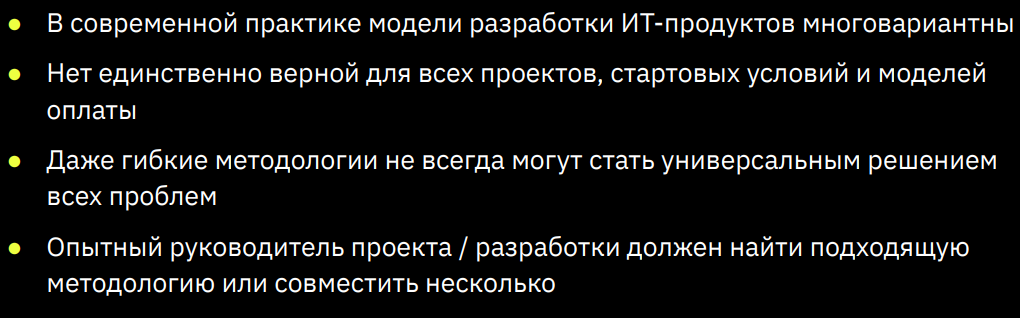
Одним из таких подходов. Одной из таких философий является и Agile. Это, так сказать, основа гибкого подхода гибких методологий для разработки этих продуктов. По сути, Agile была создана на основе итеративной модели, которая, как я уже говорил, являлась, так сказать, прародительницей гибких методологий. По сути, и Agile — это не модель, это больше подход — «философский подход к разработке» с английского переводится как гибкий, включая в себя определенные практики методологии, подходы, которые помогают эффективно качественно разрабатывать эти продукты и поставлять уже конечную ценность этого продукта заказчику и, конечно, потребителю. По сути, так как Agile — это целый философский подход, включает в себя набор методологий и практических инструментов, как эту философию реализовать в деле. Это такие подходы, как экстремальное программирование, бережливая разработка программного обеспечения Lean она еще называется, Scrum который является, по сути целым фреймворком для управления IT-проектами, разработка, управляемая функциональностью FDD, разработка через тестирование; методология «чистые комнаты», итеративный инкрементальный подход к разработке; методология разработки Microsoft Solutions Framework и так далее. То есть, по сути, очень и очень много различных методологий, которые входят в философию Agile. Соответственно, каждую из этих методологий где-то очень подробно, потому что она достаточно часто и ярко применяется при эти разработке, а какой-то более свернута, мы будем рассматривать чуть дальше.

Хотелось бы остановиться на основных таких методологиях и фреймворках, которые применяются в рамках гибкого подхода к разработке. Во-первых, это Scrum. По сути, это фреймворк. Это более сформированная методология с практическими инструментами, ценностями, подходами и в целом описанием того, как должна работать и взаимодействовать команда разработчиков, руководителей проекта и владельца продукта. По сути дела, Scrum четко прописывает все роли и процессы, которые происходят при разработке IT-продукта. Следующая интересная методология в рамках IT-философии — это Canban. Сегодня это, можно сказать, такой брат Scrum. Это одна из наиболее популярных методологий, которая применяется при разработке и при управлении этой самой разработкой. Включает в себя определенные принципы. Команда ведет разработку с помощью визуализированной виртуальной доски, которая разбита на определенные этапы проекта. Каждый участник, соответственно, видит, какие задачи находятся у него в работе, какие застряли на каком-то этапе, и требует, чтобы их подвинули. Также важно отметить, что основное отличие Canban от Scrum заключается в том, что в Canban срочные задачи разработчик может взять сразу, не дожидаясь, пока закончится какой-то определенный спринт, как это сделано в Scrum. Конечно, сразу представить, как все это функционирует, достаточно сложно. Именно поэтому сейчас мы углубляться в эти самые гибкие методологии, такие как Scrum, Canban, Lean и другие, которые я перечислил в рамках Agile, не будем. Давайте двигаться постепенно, эволюционно, как это происходило развитие наших методологий и моделей управления IT-продуктом. Сейчас хотелось бы отметить основные отличия гибких методологий от наших традиционных, про которые мы сегодня говорили. Таким образом, подвести итоги нашей сегодняшней лекции.



Здесь вы видите такую сравнительную таблицу, которая включает в себя основные характеристики IT-проекта по разработке и сравнение как это видится в традиционных методологиях и моделях и в гибких давайте смотреть постепенно. Во-первых, подход к разработке традиционной модели, он прогнозирующий, то есть в самом начале мы должны четко видеть, что мы хотим получить. Мы делаем определенный прогноз, и все достаточно четко детально планирую. Гибкие методологии предполагают адаптивный подход к разработке, то есть мы видим маленькую небольшую часть как будет функционировать наше программное обеспечение, разрабатываем ее и затем постепенно начинаем доработку.





Критерии успеха если мы смотрим с точки зрения традиционного подхода к критериям успеха является следование четкому плану. Если же смотреть с точки зрения Agile гибкая методология, то здесь больше смещается фокус в сторону ценности для бизнеса. То есть мы хотим поставить действительно качественный IT-продукт, которым сможет воспользоваться бизнес-конечный наш пользователь. Если мы говорим про риски, учитывая, что в традиционном подходе делается достаточно углубленное изучение и анализ всех рисков в самом начале. Именно поэтому при таком подходе риски у нас уже изначально определены. Да, это неплохо, где-то в каких-то отраслях это хорошо, где-то наоборот, неприменимо. Соответственно, гибкие методологии предполагают то, что в самом начале наши риски недостаточно проработаны. Они возникают по мере того, как мы начинаем разработку, как мы дорабатываем наш продукт. Контроль? В традиционном подходе очень легко контролировать разработку, потому что там все идет поэтапно. Методологии, грубо говоря, каскада водопада. Все начинается с самого первого этапа, после того как он завершается, переходится на второй этап и так далее. То есть мы нигде никуда не прыгаем по мере разработки IT-продукта. В гибких методологиях, здесь уже контроль зависит от профессионального уровня команды, от профессионального уровня руководителя проекта, специалистов, которые работают над вашим проектом. И в целом всей команды: это дизайнеры, программисты, scrum-мастера, про которых мы будем подробнее говорить, чуть дальше — руководители проектов, владелец продукта и так далее. Здесь уже более такой адаптивный подход, где достаточно сложно контролировать в целом разработку, и это под силу только профессионалам. Заказчики? Если мы говорим про традиционный подход, там достаточно низкая вовлеченность заказчиков. Почему? Потому что они в самом начале все достаточно детально проработали, выкатили определенные требования к конечному IT-продукту и, соответственно, команде разработчиков и удалились от процесса разработки. Если мы говорим про Agile-подход в гибких методологиях, у нас высокая вовлеченность, потому что разработчики делают какую-то итерацию и выполняют определенные работы и предоставляют данную версию продукта на тест своему заказчику. Тот смотрит и говорит, что исправить, как исправить, и чтобы он хотел видеть в следующей версии продукта. Таким образом, мы максимально вовлекаем заказчика в процесс разработки. Документация — следующий основной параметр реализации любого проекта. В традиционных подходах документация достаточно детальная и очень много, и она формируется с самого начала проекта, что серьезно бюрократизирует в целом весь процесс разработки. В гибких методологиях, документация формируется по мере развития проекта и очень серьезно делается уклон именно практически применимой документации. То есть стараются уходить от ненужных бюрократических бумажек в сторону того, чтобы создавать действительно практически применимый документ, который будет работать который будет являться настольной книгой, настольным документом для управленца, для разработчика и так далее. То есть это не просто документ ради документа, а документ ради того, чтобы зафиксировать определенные требования, понять, куда двигаться и в целом что вы в конце получите. То есть в гибких методологиях документ у нас практически применимый инструмент. А в традиционном подходе документ — это больше бюрократическая процедура. Касательно требований в традиционном подходе требования нам известны заранее. Помните, я приводил пример, когда говорил про «ватерфолл», что он очень хорошо применим в строительной сфере, сфере медицины, в сфере космических каких-то технологий, там, где есть четкие жесткие регламенты, требования, различные кодексы Российской Федерации, которые выставляют нам эти требования уже в самом начале, и мы их нарушать не можем. Если мы говорим про гибкие методологии, здесь требования не всегда известны заранее. К примеру, тот же самый стартап, как мы можем знать заранее, как он должен функционировать таким образом, чтобы донести действительно конечную ценность до своего конечного пользователя. Естественно, в самом начале этого сделать невозможно. Поэтому эти самые требования они легко изменяются по мере того, как выходит на рынок определенная версия продукта, дорабатывается, изменяется и вносится корректировки в то самое первоначальное техническое задание, которое составляет заказчик. Опять же можно вернуться и посмотреть, что происходит высокая вовлеченность заказчика, потому что он может менять свои требования в зависимости от того, какой фидбэк он получает от конечных пользователей. Следующая важная характеристика проекта — это команда проектов, потому что без людей мы ничего не сможем сделать. Но если мы посмотрим на традиционный подход, то здесь включение новых специалистов на любом этапе разработки оно происходит достаточно гладко, и оно возможно. И да, это серьезный плюс, потому что людям свойственно находить места получше, уходите из одного проекта в другой и так далее. Традиционный подход нам позволяет это делать. Если мы говорим про гибкие методологии, здесь нам нужен стабильный состав, включает в себя опытных специалистов, которые, соответственно, могут выстроить этот самый гибкий подход к разработке и получить действительно ценный продукт. Поэтому команда для реализации проекта с помощью гибкой методологии она не должна превышать порядка 15-20 человек. Это, так сказать, самый максимум, которым уже становится очень сложно управлять в процессе разработки.

Следующая характеристика — это рефакторинг, то есть изменение кода. Если мы хотим исправить какую-то ошибку, которую мы видим на этапе разработки в традиционном подходе, там это сделать достаточно дорого и сложно, потому что нам нужно откатиться назад, исправить абсолютно все и перелопатить весь код. Если мы говорим про гибкую методологию, то так как там происходит и итеративная разработка, итерации за итерации, исправить ошибку стоит недорого, но ошибки, возможно, их исправлять нужно, но гибкой методологии при использовании гибкой методологии это сделать достаточно проще, чем если делать в традиционном порядке в классическом подходе к разработке. Что же хотелось под итожить и какие сделать выводы? На самом деле в современной практике модели разработки IT-продуктов они многовариантны. Нельзя сказать, что только один какой-то единый подход дает получения той самой цели, которую мы ставим перед собой самом начале. Нельзя говорить, что только гибкие модели и методологии позволяют нам получить действительно качественно идти продукт. Мне где-то надо применить и каскад, где-то и в-образную модель, где-то и спираль подойдет, а где-то важны именно гибкие подходы к разработке, особенно если это касается каких-то стартапов. Здесь важно отметить, что, то самое бремя по выбору этой самой методологии для разработки IT-продуктов ложится по большей части на руководителя разработки, то есть руководителя IT-проекта. Опытный руководитель должен в самом начале определить, какую модель, какую методологию выбрать для разработки либо всего IT-продукта, либо только определенных ее частей.  
В завершение нашей лекции хотелось бы сделать следующие основные выводы: в самом начале люди, конечно же, использовали традиционные подходы к разработке программного обеспечения. Затем по мере того, как развивалась IT-индустрия, различные другие сферы, двигался прогресс. Естественно, пришло понимание к тому, что невозможно все в начале предусмотреть, распланировать и выставить жесткие четкие требования, потому что это приведет лишь к жесткому четкому следованию плана. Цель создания любого IT-продукта – это ценность для бизнеса, ценность для конечного пользователя, который будет уже использовать ваш IT-продукт, ваше приложение, программное обеспечение, ваш интернет-сайт и так далее, в общем, любую систему, которую вы создадите. Именно поэтому люди поняли, что нужно переходить больше гибкому подходу разработки. Это та самая Agile-философия, которая уже взяла свое начало от итеративной модели и затем получились различные ответвления, такие как Scrum, Lean, Canban, FDD, XP и FDD. Про все эти гибкие подходы, методология к разработке IT-продуктов, более подробно, последовательно с примерами и как все это происходит, как это применяется на практике, мы будем говорить уже на последующих уроках.