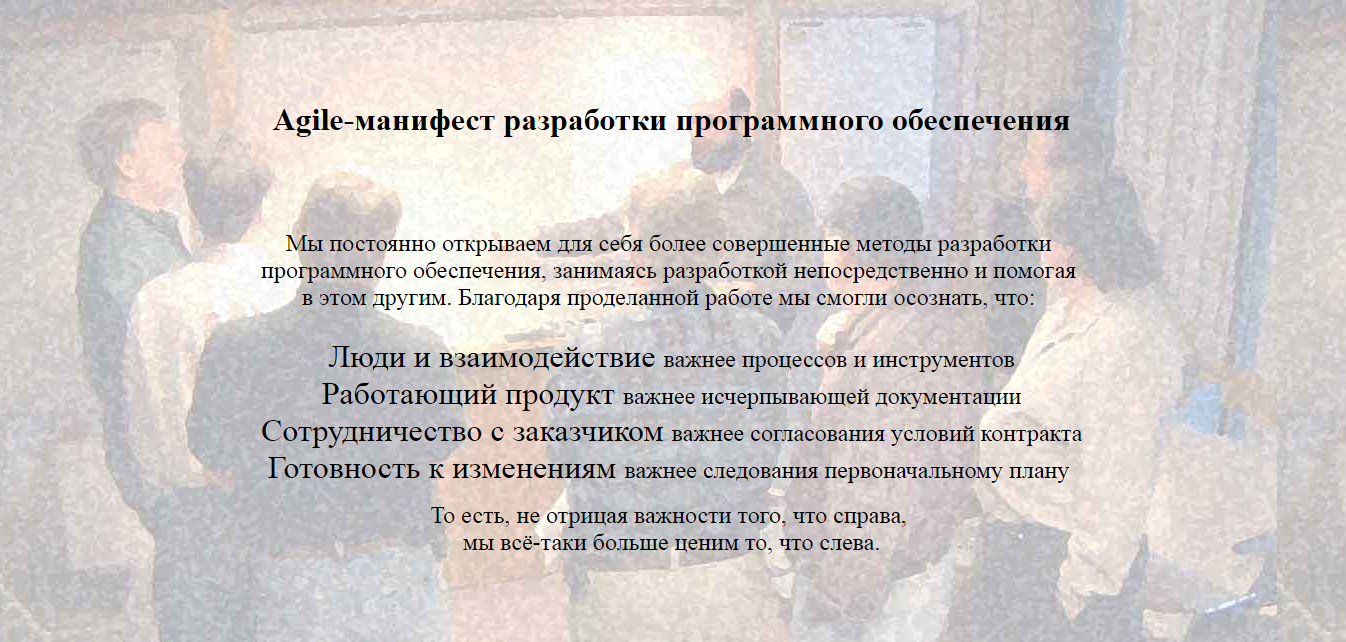
Что такое Agile и Scrum?

Что такое Agile?

В переводе с английского языка **«agile»** означает «живой, подвижный», но переводят его чаще как «гибкий». В отрасли разработки программного обеспечения этот термин появился в начале 2000-х годов, когда в штате Юта был издан «[Манифест гибкой разработки ПО](http://agilemanifesto.org/)». С тех пор под «agile» понимают набор подходов по “гибкой” разработке программного обеспечения.

**

*Agile Manifest*

Суть agile-подхода изложена в “манифесте”, но для заказчика ее можно коротко сформулировать так:

* разработка ведется короткими циклами (итерациями), продолжительностью 1-4 недели;
* в конце каждой итерации заказчик получает ценное для него приложение (или его часть), которое можно использовать в бизнесе;
* команда разработки сотрудничает с Заказчиком в ходе всего проекта;
* изменения в проекте приветствуются и быстро включаются в работу.

В настоящее время agile-принципы используются в работе десятки тысяч команд по всему миру.

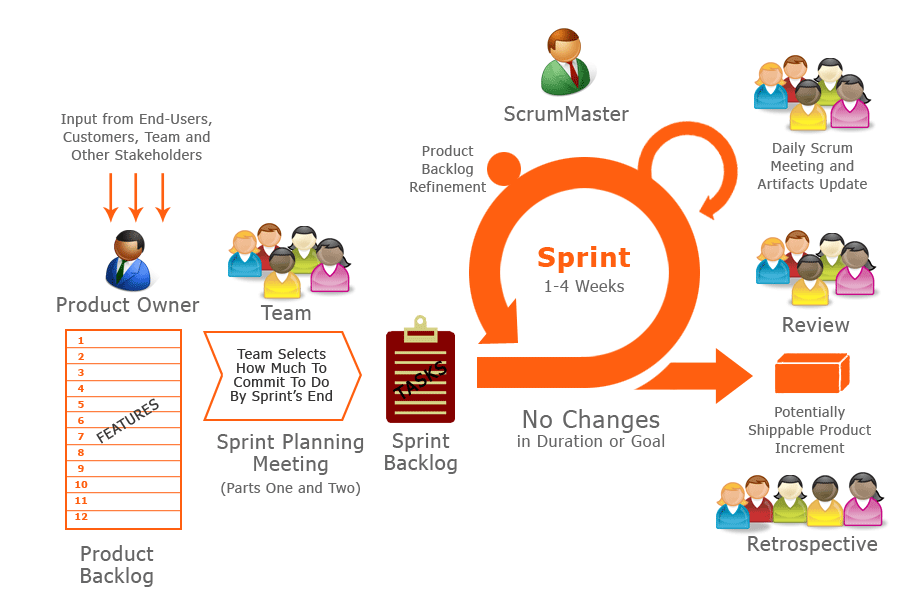
[Основополагающие принципы Agile](https://www.pmoffice.by/wp-content/uploads/2012/04/Osnovopolagayushhie-printsipyi-Agile.pdf).

Краткое видео о том, что такое Scrum (english).

Что такое Scrum?

Scrum – это одна из нескольких методологий гибкой разработки ПО:

* + **Scrum**
  + Lean
  + Feature Driving Development
  + Extreme Programming

**

*Scrum процесс на одном листе*

Scrum – это спортивный термин, который пришел к нам из регби, и представляет собой фигуру, которую образуют игроки перед началом игры.



**Артефакты в Scrum**

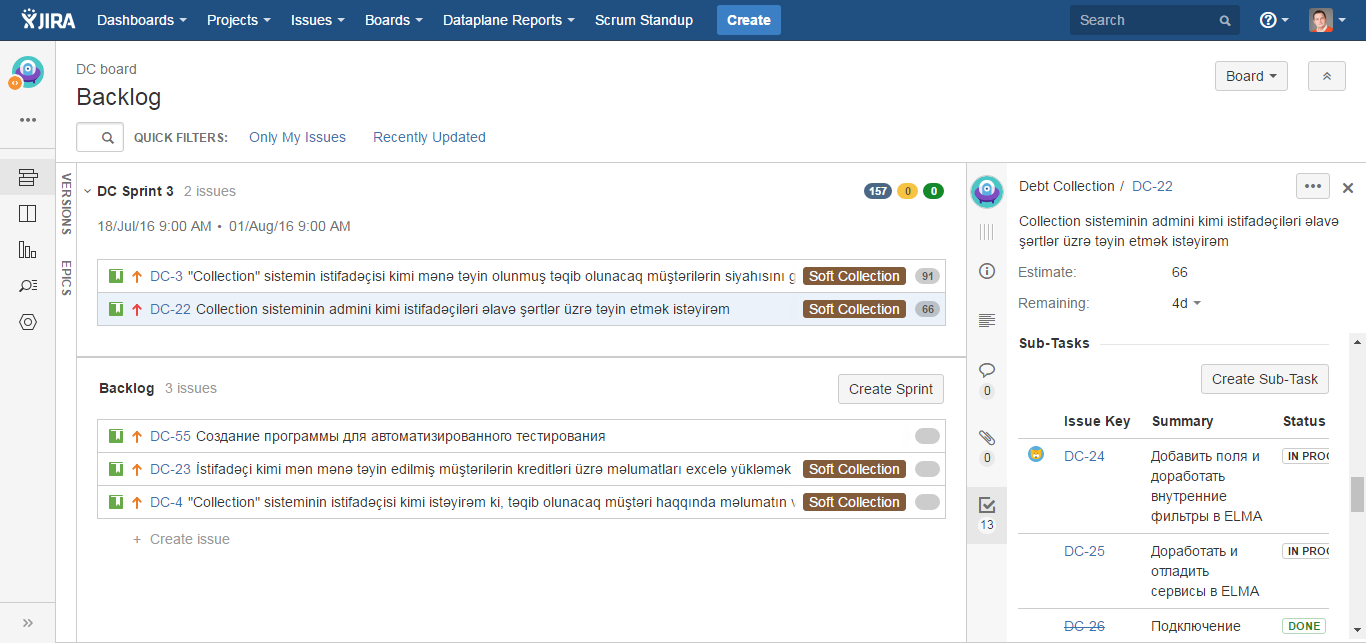
В скрам используется всего четыре артефакта:

* Product Backlog
* Sprint Backlog
* Sprint Goal
* Sprint Burndown Chart.

**Product backlog:**

* Это список всех требований, которые нужно сделать по проекту. Когда в Backlog’e нет требований, проект считается завершенным.
* Все требования описаны по единому шаблону, который называют User Story (пользовательская история).
* Требования составлены так, что очевидно и понятно, какую ценность они представляют для пользователя
* Требования отсортированы по приоритетам, которые пересматриваются каждый спринт.

На снимке ниже представлен Backlog проекта. Команда проекта выбрала 2 требования в Sprint#3.

**

*Project Backlog (JIRA)*

Получить стоимость обучения Agile/Scrum за 60 минут?

Начало формы

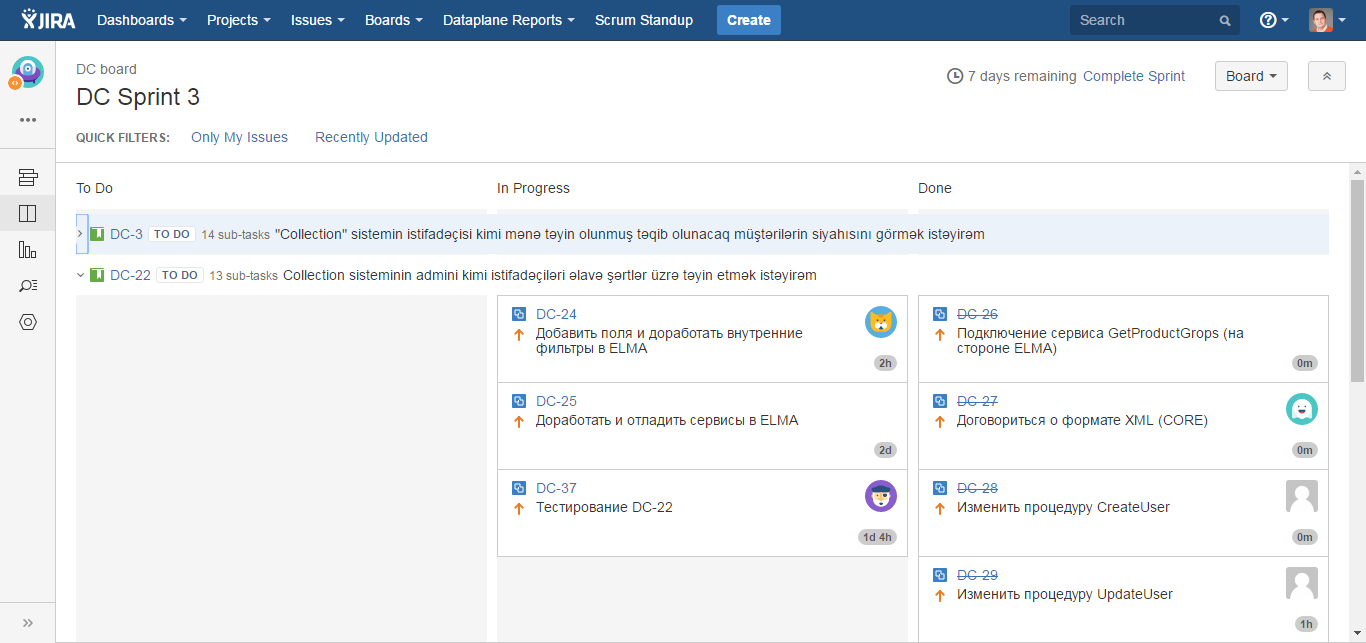
Телефон:Компания:Согласен на обработку данных  


**Конец формы**

**Sprint backlog:**

* Это список всех требований, которые нужно сделать в ближайший спринт.
* В течение спринта, новые требования не могут появится в Sprint backlog.
* Все требования должны быть разделены на задачи и оценены.

Sprint Backlog – это обязательство команды: что они должны выполнить за ближайшие 2 недели. Каждое требование разделено на задачи, которые представлены на Kanban-доске.

**

*Kanban Доска в Спринте*

**Sprint Goal**

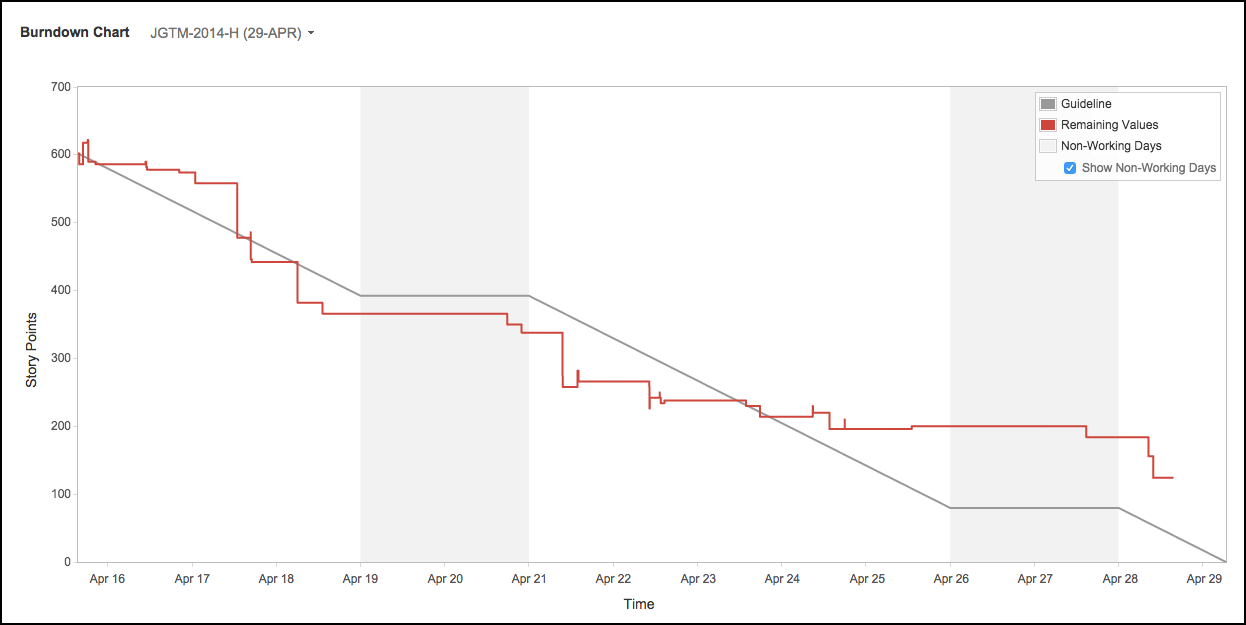
* это краткое описание того, ради чего выполняется данный спринт.
* цель на спринт помогает команде принимать обоснованные решения.

Этот артефакт необходим для того, чтобы команда проекта могла самостоятельно принимать решение в случае появления альтернативных путей решения задачи. Чтобы решения команды были осознанными, Product Owner определяет цель спринта.

**Sprint Burndown Chart**

* дословно “диаграмма сгорания”
* в качестве “сгорающих” элементов выступают человеко-часы или идеальные единицы (Story Points).
* диаграмма обновляется каждый раз, когда завершается какая-либо задача.

Внешний вид диаграммы на рисунке ниже. На практике такая диаграмма очень наглядна: каждый день можно быстро узнать, насколько команда продвинулась вперед.

**

*Burndown диаграмма в Jira*

**Роли в Scrum**

В скрам используется всего три роли:

* Product Owner
* Scrum Master
* Team.

Роль Product Owner

* формулирует требования
* приоритезирует требования
* корректирует приоритеты на каждом спринте
* несет персональную ответственность за ценность требований для рынка/пользователей
* отвечает за взаимодействие с рынком
* только один человек

Product Owner – это представитель подразделения, которое владеет разрабатываемым продуктом. Например в банке это может быть Департамент карточных продуктов. Правильно определить Product Ownera не просто, т.к. эта роль требует сочетания следующих качеств:

* иметь личную вовлеченность в проект и его результаты;
* хорошо владеть навыком написания требований.

В некоторых случаях допустимо назначить более одного человека на роль Product Owner. Но в этом случае необходимо назначить среди них “главного”, который будет авторизовать требования в Bcaklog’e и лично расставлять приоритеты.

Роль Scrum Master

* следит за корректным применением принципов Agile и процессов (ритуалов) Scrum
* организует работу команды и обеспечивает её всем необходимым
* защищает команду, несёт ответственность за её эффективность
* только один человек.

Очень сложная роль. В классическом project management есть Руководитель проекта. В Scrum такая роль не предусмотрена. Лучшим синонимом роли Scrum Master будет “администратор”. Скрам Мастер организовывает работу команды проекта, но не вмешивается в её работу.

* Скрам мастер не назначает людей на задачи – это делает сама команда;
* Мастер не заставляет людей делать работу – это ответственность команды;
* Мастер не указывает Product Owner какие требования он должен написать – это работа владельца продукта.

Тем не менее, если скрам-процесс проходит с нарушениями (кто-либо из команды опаздывает на daily-meeting), то мастер должен вмешаться и исправить ситуацию.

Функции Scrum Master’a существенно шире, но чтобы пояснить их все нужна отдельная статья. Пишите в комментариях, если таковая нужна.

Team (команда проекта)

* кросс-функциональная
* взаимозаменяемая
* самоорганизующаяся
* с фиксированным составом (в ходе спринта)
* 4-10 человек.

Команда отвечает за разработку продукта итерациями (спринтами). Команда определяет самостоятельно:

* продолжительность спринта
* емкость (capacity) команды
* размер её фокус фактора (коэффициент слаженности)
* трудоемкость требований, которые будут реализованы в спринте
* очередность выполнения задач и много другое.

Команда НЕ принимает решений:

* какие требования являются приоритетными – это делает Product Owner.

На снимке ниже команда проекта проводит обязательный “ритуал” – Daily Meeting (см. ниже).

**

*Команда проводит “ритуал” Daily Meeting*

Ритуалы (процессы в Scrum)

В скрам есть несколько процессов, которые принято называть ритуалами. Каждый ритуал выполняется неукоснительно и в строгом соответствии с подходом. На практике такие процессы стараются немного адаптировать, но ключевые принципы не изменяют.

Ритуалы в скрам это:

* Sprint Planning Meeting
* Daily Meeting
* Sprint Review
* Retrospective

Sprint Planning Meeting (встреча по планированию спринта)

* выполняется всей командой перед началом спринта
* команда выбирает требования из Product Backlog и формирует Sprint Backlog
* если требуется учесть взаимосвязи между операциями, то это делается здесь
* команда декомпозирует требования на задачи (tasks)
* каждая задача проходит оценку в трудозатратах или универсальных единицах
* во время встречи Product Owner отвечает на вопросы команды.

Встреча, которая проводится перед началом каждого спринта. Структура встречи:

* представление и пояснение Product Owner’ом списка требований
* вопросы со стороны команды
* /рекомендуется перерыв/
* декомпозиция требований на задачи (tasks)
* оценка задач по методу Planning Poker.

Встреча простая по сути, но крайне сложная по содержанию. В начале проекта может занимать 5-6 часов. И только после 3-4 спринта встреча становится более оперативной и длится 2-3 часа. Крепитесь.

Daily Meeting (ежедневная встреча команды).

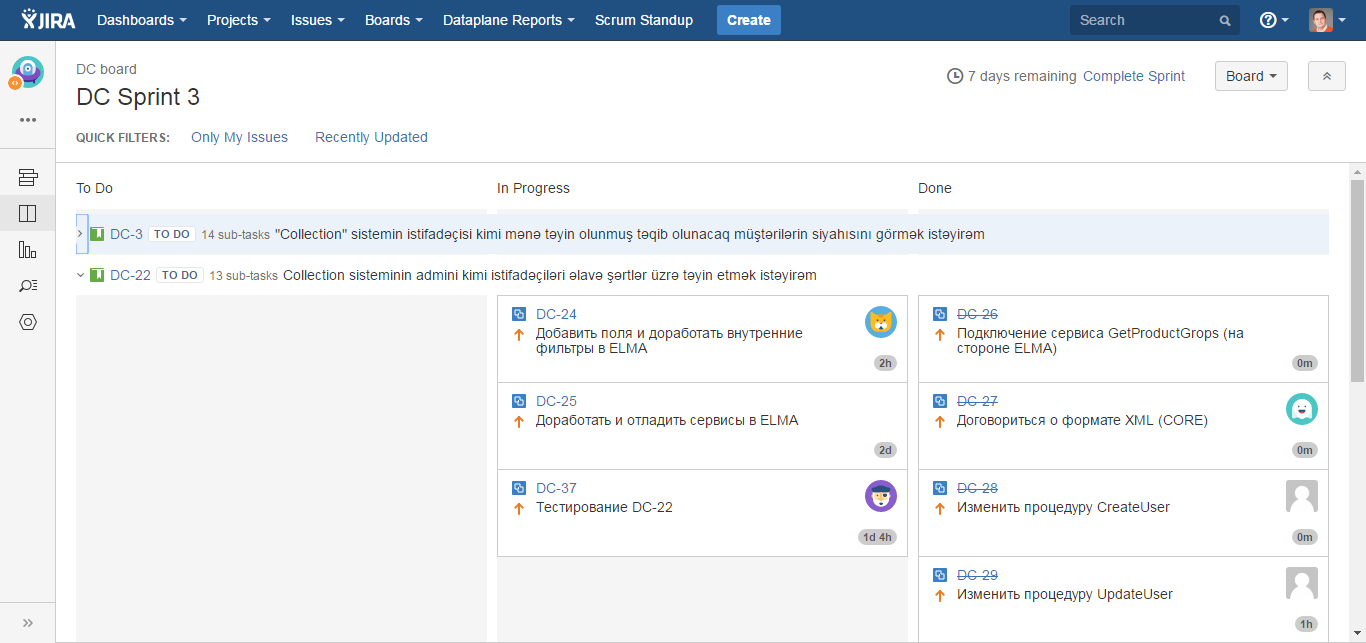
Из названия понятно, что встреча проводится ежедневно. Основные принципы:

* проходит ежедневно и только в одно и то же время;
* встреча проходит только стоя;
* поэтому длительность встречи не более 15 минут;
* чтобы успеть каждый должен ответить всего на 3 вопроса: что я делал вчера, чем я занимаюсь сегодня, какие есть проблемы?

Scrum Master следит за ходом встречи, побуждает участников высказываться полностью и слушать говорящего.

На ежедневной встрече команда обменивается опытом. Также становится понятно, кто и над какими задачами будет сегодня трудиться. Важно, чтобы команда делала этот ритуал самостоятельно. Я вообще рекомендую Scrum Masters не вмешиваться в ход встречи до тех пор, пока соблюдаются все требования к этому ритуалу.

Встреча команды эффективно проводить напротив Kanban доски, на которой отражены все задачи спринта.

**

*Kanban Board во время спринта*

Sprint Review – сдача спринта Product Owner

По завершению каждого спринта команда обязана провести демонстрацию полученного результат. Ценность этого ритуала я поясню отдельно.

Ценность Scrum для обычного заказчика во многом состоит в том, что результат работ (плохой или отличный, не важно) будет продемонстрирован в любом случае. Это знает и команда и Product Owner и другие заинтересованные лица. Если команда не проводит демонстрацию (иное название Sprint Review), то это дискредитирует все преимущества гибких процессов.

Структура встречи:

* команда зачитывает требования из Sprint Backlog
* по каждому критерию приемки происходит демонстрация полученных результатов
* каждый вопрос со стороны Product Owner’а записывается, чтобы иметь возможность ответить на них позже
* каждое новое требование Product Owner’a выписывается, чтобы позже включить его в Product Backlog.

На встрече могут присутствовать любые сотрудники организации или просто заинтересованные лица. Важно, чтобы право голоса имели только участники Scrum процесса (Produt Owner, Team, Scrum Master).

Никаких презентаций в PowerPoint на встрече, если вы правильно меня поняли!

Retrospective

Ритуал, который направлен на обмен опытом внутри команды. Встреча проводится после Sprint Review. На встрече присутствует вся команда и Scrum Master. На встрече *может присутствовать*Produt Owner, если считает нужным.

Методика проведения встречи варьируется в зависимости от проекта, его команды и просто традиций в коллективе. Тем не менее, должны быть озвучены ответы на следующие вопросы:

* какие решения должна принять команда, чтобы сделать процесс более предсказуемым?
* какие проблемы мешают команде выполнять взятые на себя обязательства?
* как улучшить взаимодействие с Product Owner’ом?
* какие ошибки совершает команда и почему.

Решения должны быть записаны на отдельной доске. После всеобщего голосования решения принимаются к исполнению со следующего спринта. Scrum Master контролирует ход встречи и следит за её регламентом.

**Почему появился Agile?**

Теперь немного слов о том, как и зачем появился этот подход? История возникновения этого подхода стала ответом на запросы отрасли:

1. Заказчик не может сформировать четкие требования к ПО;
2. Новые технологии усилили конкуренцию и потребовали оперативного применения в бизнесе;
3. Заказчики и разработчики ПО не удовлетворены процессом взаимодействия.

**#1 Заказчик не может сформировать четкие требования к ПО**

В начальной фазе проекта заказчик не может сформулировать исчерпывающие требования к продукту. Этому есть несколько причин:

* у Заказчика существует только идея приложения и он не представляет всю его функциональность;
* у группы проекта есть разный взгляд на функциональность приложения;
* команда не может договориться, как же будет удобнее/разумнее реализовать ту или иную часть функциональности приложения.

*Один из принципов Agile гласит «Используйте прототипы и делайте поставки продукта как можно чаще». Это позволит снять неопределенность в требованиях и проверить, как с ней будут работать реальные пользователи.*

В традиционных «водопадных» моделях руководитель проекта минимизирует изменения в проекте, используя для этого отдельные процессы – управление изменениями. Но если требования будут меняться раз в месяц, то управление изменениями становится трудоемким и замедляет ход проекта.

*В Agile реакция на изменения важнее, чем следование плану. В Agile приветствуется, когда заказчик и пользователи вносят новые требования, чтобы сделать продукт более конкурентноспособным.*

**#2 Новые технологии усилили конкуренцию и потребовали оперативного применения в бизнесе**

К середине 90-х разрабатываемое ПО было в основном десктопным и его требовалось устанавливать на каждый отдельный компьютер (например, MS Word). С появлением веб-приложений внедрение новой функциональности стало происходить быстрее: требовалось развернуть приложение только на сервере и все пользователи получали к нему доступ. Эта инновация серьезно усилила конкуренцию между компаниями: тот, кто применил новую технологию раньше других – выигрывает рынок и клиентов.

*Подход Agile предоставил бизнесу главное преимущество – быстрые поставки новой функциональности. Это позволило каждый месяц выпускать продукт и оперативно получать обратную связь от пользователей.*

**#3 Заказчики и разработчики не удовлетворены процессом взаимодействия**

При жестком сроке и в условиях постоянных изменений разработчики вынуждены формализовывать процессы взаимодействия с Заказчиком. Разработчики закладывают в бюджет работы по созданию детальных требований и спецификаций, а также риски на возможные их изменения. При этом Заказчик вынужден оплачивать документы, которые не несут реальной ценности для бизнеса.

*Основная идея agile – сотрудничество с заказчиком важнее, чем контрактные обязательства. И поэтому agile-методы стремятся к уменьшению объема документации. Это позволяет Заказчику платить только за результат, имеющий ценность для бизнеса.*

При этом agile не отказывается от формулирования требований. Заказчик (в agile – владелец продукта, product owner) предъявляет требования в упрощенном виде и на сценариях работы пользователей.

**Резюме**

Agile-философия проста. Agile-принципы разумны. Но переход к реальному применению agile – это серьезный вызов для каждой команды. Требуется не только освоить новый подход к управлению проектами, но также подобрать людей, способных работать в agile режиме.

Как работает скрам-команда? В чем отличие между «владельцем продукта» и «скрам-мастером»? Как составить бэклог продукта и грамотно провести ежедневный скрам?

**1. Владелец продукта (Product Owner)** – отвечает за итоговый продукт, знает, в чем его востребованность для целевой аудитории, клиента. Владелец продукта задает направление движения для всей команды. Важно, что он не руководит процессом, а лишь расставляет приоритеты.

«Он не ставит задачи программистам на языке разработчиков, он в этом не компетентен, но этот человек знает, каким должен быть продукт. Знаю прецедент, когда владельца продукта назначали, но из этого ничего не вышло. Это важная роль и далеко не каждый может с ней справиться. Нужно «гореть» продуктом, быть примером для команды, оценивать риски и чувствовать динамику команды». Артем Молчанов

Например, в it-подразделении банка «Альфа-лаборатории», чтобы найти человека на роль владельца продукта специалисты проводят собеседования и ищут конкретный тип людей: с предпринимательским и организаторским опытом. А после отбора подходящего человека компания вкладывается в его обучение.

**2. Скрам-мастер (Scrum Master)** – этот человек следит за тем, чтобы принципы скрама были понятны всем участникам. Он должен суметь организовать эффективную работу команды, общаться с каждым сотрудником, наблюдать за всем процессом изнутри.

**3. Скрам-команда (Scrum Team)** – многофункциональная команда из специалистов, которая работает над продуктом с начала и до конца. В ней есть люди разных профессий: у каждого свой взгляд на продукт. Никакой иерархии: команда сама себе ставит задачи, оценивая свои возможности. Задачи не могут быть «спущены» от владельца продукта.

В команде каждый помогает друг другу, и все вместе отвечают за результат. Большие коллективы следует поделить на несколько команд, в каждой из которых от трех до девяти человек.

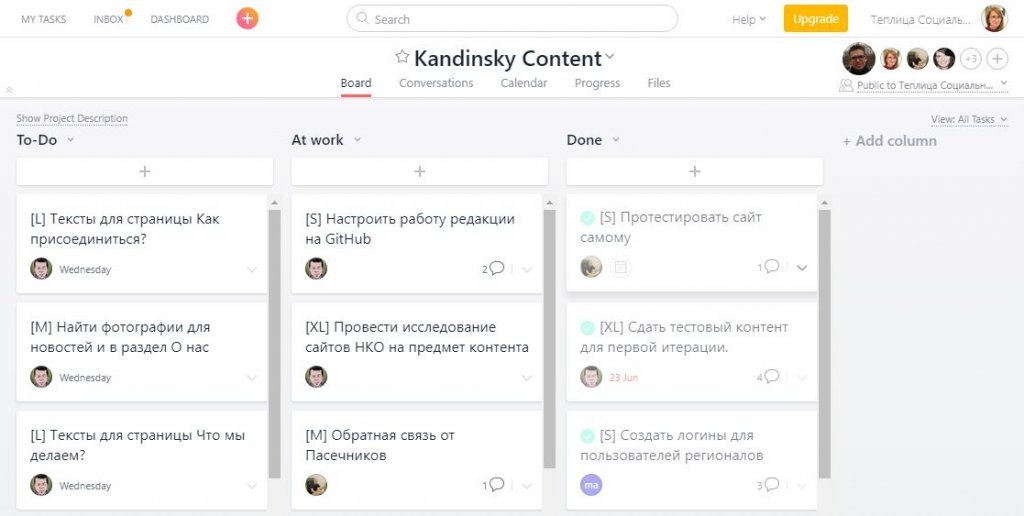
**4. Пользовательская история (User Story)** – описание требований к продукту с точки зрения запроса пользователей. Требования должны отвечать на вопрос: какие функции, дизайн должен быть у продукта, чтобы его было удобно использовать?

**5. Бэклог продукта (Product Backlog)** – это приоритизированный набор пользовательских историй. В процессе этот список может обновляться.

**6. Спринт (Sprint)** – это циклы выполнения задач. Их длительность может быть от одной до четырех недель. За это время команда должна создать потенциально готовый к работе продукт, чтобы пользователи могли его протестировать и поделиться впечатлениями. Спринт состоит из нескольких этапов: планирование, работа, презентация результата и анализ. Команда сама оценивает свои силы на один спринт и формирует столько задач, сколько сможет выполнить.

«Весь спринт владелец продукта проживает с командой. Он, как глава семьи, мотивирует участников каждый день, общается с ними. Если команда говорит: «Чувак, это нереально сделать», он спросит: «А что в ваших силах?». Тогда команда предлагает новый вариант. Как правило, взрослые люди, профессионалы выполняют свои обещания. Это уже на уровне культуры». Артем Молчанов

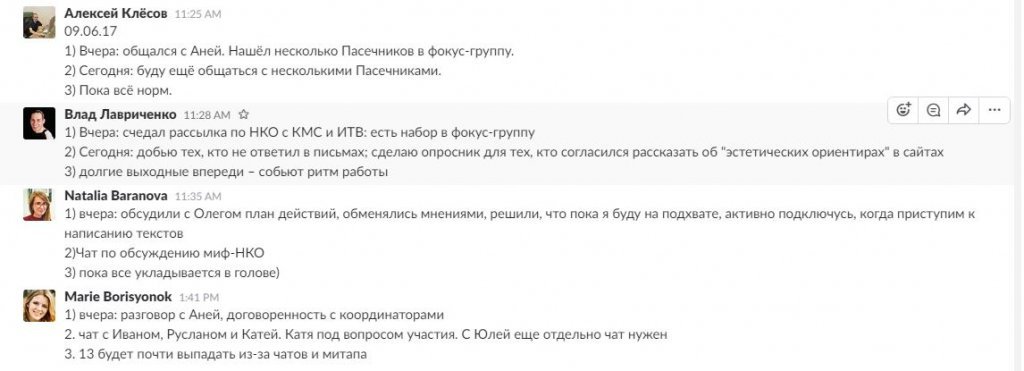
**7. Скрам-доска – визуальное отображение рабочего процесса.** Все задачи стоит прописывать: можно на флипчарте или создать виртуальный рабочий стол в вашем корпоративном веб-сервисе. Необходимо сделать три столбца с такими названиями: «сделать», «в работе» и «сделано». По мере выполнения задачи перемещаются в другие столбцы.



ТАК ВЫГЛЯДИТ СКРАМ-ДОСКА ТЕПЛИЦЫ СОЦИАЛЬНЫХ ТЕХНОЛОГИЙ.

**8. Бэклог спринта (Sprint Backlog)** – это список задач и объем работы, которую нужно сделать за один спринт. К примеру, за одну или две недели.

**9. Ежедневный скрам (Daily Scrum)** – так называют каждодневную встречу команды. Если ваша команда работает удаленно, отчет можно предоставлять в вашем рабочем чате. Поставьте напоминание на каждый день, например, 14:00, – в это время каждый сотрудник должен прислать свой небольшой отчет. Еще вариант – устраивать собрание команды на 10-15 минут.

ЕЖЕДНЕВНЫЙ СКРАМ КОМАНДЫ ТЕПЛИЦЫ СОЦИАЛЬНЫХ ТЕХНОЛОГИЙ.

На таких встречах участники команды отвечают на три вопроса: что они сделали вчера, что будут делать сегодня и с какими трудностями столкнулись. С помощью таких ежедневных планерок каждый сотрудник знает, чем занимались его коллеги. Все в курсе событий, изменений и трудностей проекта.

**10. Инкремент продукта (Product increment)** – ощутимый результат работы одного спринта. Например, внедрение новой функции на сайт, прототип мобильного приложения. Команда должна показать целевой аудитории, что она сделала за это время. Это нужно для получения обратной связи и планирования следующих задач по улучшению продукта.

**11. Ретроспектива спринта (Sprint Retrospective)** – собрание после завершения спринта. «Вся команда, скрам-мастер и владелец продукта обсуждают, что было сделано на «отлично», а что пошло не так, и что нужно сделать для того, чтобы в следующий раз быть успешнее. И это общая цель, потому что если команда не успела что-либо сделать, клиенты просто не увидят, например, нескольких полезных и более усовершенствованных функций продукта», – говорит Артем Молчанов.

**12. Самоорганизация (Self-organization)** – ключевой критерий работы заключается в том, что команда действует по принципу самоорганизации. Она не ждет поставленных сверху задач, а работает автономно.

«Работая в команде, сотрудники горят продуктом, вместе расстраиваются, когда что-то не получается. Люди, которые равнодушно относятся к делу, постепенно уходят», – уточнил эксперт.

По словам ***Артема Молчанова***, важно не допустить демотивации сотрудников. «Бывает, что продукт не получился, владелец продукта не отработал обратную связь, а команда не понимает, что делать. Отсюда и негатив, всем непонятно, что делать дальше. Поэтому всегда стоит налаживать коммуникацию с сотрудниками, базовый уровень мотивации есть у всех, конечно, не все спринты будут успешными, но чтобы достичь высокого результата, нужно каждый день трудиться над этим», – посоветовал Артем Молчанов.

Существуют модели разработки ПО. И существуют методологии. В интернете много противоречивой информации о том, что есть что и как их отличать. Начинающему специалисту бывает сложно в этом разобраться. В этой статье мы расставим все точки над i.

**Этапы жизненного цикла ПО**

У любого программного обеспечения есть жизненный цикл — этапы, через которые оно проходит с начала создания до конца разработки и внедрения. Чаще всего это подготовка, проектирование, создание и поддержка. Этапы могут называться по-разному и дробиться на более мелкие стадии.



Рассмотрим эти этапы на примере жизненного цикла интернет-магазина.

**Подготовка.** Иван решил запустить книжный интернет-магазин и начал анализировать, какие подобные сайты уже представлены в сети. Собрал информацию об их трафике, функциональности.

**Проектирование.** Иван выбрал компанию-подрядчика и обсудил с её специалистами архитектуру и дизайн будущего интернет-магазина.

**Создание.** Иван заключил с разработчиками договор. Они начали писать код, отрисовывать дизайн, составлять документацию.

**Поддержка.** Иван подписал акт сдачи-приёмки, и подрядчик разместил интернет-магазин на «боевых» серверах. Пользователи начали его посещать и сообщать о замеченных ошибках в поддержку, а программисты — оперативно всё исправлять.

*Модель*разработки программного обеспечения описывает, какие стадии жизненного цикла оно проходит и что происходит на каждой из них.

А*методология* включает в себя набор методов по управлению разработкой: это правила, техники и принципы, которые делают её более эффективной.

Основные модели разработки ПО

* Code and fix — модель кодирования и устранения ошибок;
* Waterfall Model — каскадная модель, или «водопад»;
* V-model — V-образная модель, разработка через тестирование;
* Incremental Model — инкрементная модель;
* Iterative Model — итеративная (или итерационная) модель;
* Spiral Model — спиральная модель;
* Chaos model — модель хаоса;
* Prototype Model — прототипная модель.

Из этих моделей наиболее популярны пять основных: каскадная, V-образная, инкрементная, итерационная и спиральная. Разберём их подробнее.

Waterfall (каскадная модель, или «водопад»)

В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. Если всё делать правильно, «водопад» будет наиболее быстрой и простой моделью. Применяется уже почти полвека, с 1970-х.



**Преимущества «водопада»**

* *Разработку просто контролировать.* Заказчик всегда знает, чем сейчас заняты программисты, может управлять сроками и стоимостью.
* *Стоимость проекта определяется на начальном этапе.* Все шаги запланированы уже на этапе согласования договора, ПО пишется непрерывно «от и до».
* *Не нужно нанимать тестировщиков с серьёзной технической подготовкой.* Тестировщики смогут опираться на подробную техническую документацию.

**Недостатки каскадной модели**

* *Тестирование начинается на последних этапах разработки.* Если в требованиях к продукту была допущена ошибка, то исправить её будет стоить дорого. Тестировщики обнаружат её, когда разработчик уже написал код, а технические писатели — документацию.
* *Заказчик видит готовый продукт в конце разработки и только тогда может дать обратную связь.*Велика вероятность, что результат его не устроит.
* *Разработчики пишут много технической документации, что задерживает работы.* Чем обширнее документация у проекта, тем больше изменений нужно вносить и дольше их согласовывать.

«Водопад» подходит для разработки проектов в *медицинской и космической отрасли, где уже сформирована обширная база документов*(СНиПов и спецификаций), на основе которых можно написать требования к новому ПО.

При работе с каскадной моделью основная задача — написать подробные требования к разработке. На этапе тестирования не должно выясниться, что в них есть ошибка, которая влияет на весь продукт.

V-образная модель (разработка через тестирование)

Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. История этой модели начинается в 1980-х.



**Преимущества V-образной модели**

* Количество ошибок в архитектуре ПО сводится к минимуму.

**Недостатки V-образной модели**

* Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде».

V-модель подходит *для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках.*

**Incremental Model (инкрементная модель)**

Это модель разработки по частям (increment в переводе с англ. — приращение) уходит корнями в 1930-е. Рассмотрим её на примере создания социальной сети.

1. Заказчик решил, что хочет запустить соцсеть, и написал подробное техническое задание. Программисты предложили реализовать основные функции — страницу с личной информацией и чат. А затем протестировать на пользователях, «взлетит или нет».
2. Команда разработки показывает продукт заказчику и выпускает его на рынок. Если и заказчику, и пользователям социальная сеть нравится, работа над ней продолжается, но уже по частям.
3. Программисты параллельно создают функциональность для загрузки фотографий, обмена документами, прослушивания музыки и других действий, согласованных с заказчиком. Инкремент за инкрементом они совершенствуют продукт, приближаясь к описанному в техническом задании.



**Преимущества инкрементной модели**

* *Не нужно вкладывать много денег на начальном этапе.* Заказчик оплачивает создание основных функций, получает продукт, «выкатывает» его на рынок — и по итогам обратной связи решает, продолжать ли разработку.
* *Можно быстро получить фидбэк от пользователей и оперативно обновить техническое задание.* Так снижается риск создать продукт, который никому не нужен.
* *Ошибка обходится дешевле.*Если при разработке архитектуры была допущена ошибка, то исправить её будет стоить не так дорого, как в «водопаде» или V-образной модели.

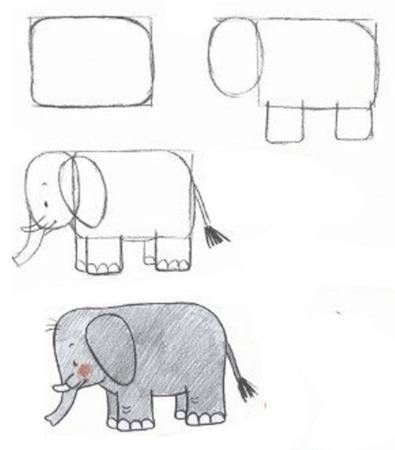
**Недостатки инкрементной модели**

* *Каждая команда программистов разрабатывает свою функциональность и может реализовать интерфейс продукта по-своему.* Чтобы этого не произошло, важно на этапе обсуждения техзадания объяснить, каким он будет, чтобы у всех участников проекта сложилось единое понимание.
* *Разработчики будут оттягивать доработку основной функциональности и «пилить мелочёвку».* Чтобы этого не случилось, менеджер проекта должен контролировать, чем занимается каждая команда.

Инкрементная модель подходит для *проектов, в которых точное техзадание прописано уже на старте, а продукт должен быстро выйти на рынок.*

**Iterative Model (итеративная модель)**

Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.



Рассмотрим на примере создания мессенджера, как эта модель работает.

1. Заказчик решил, что хочет создать мессенджер. Разработчики сделали приложение, в котором можно добавить друга и запустить чат на двоих.
2. Мессенджер «выкатили» в магазин приложений, пользователи начали его скачивать и активно использовать. Заказчик понял, что продукт пользуется популярностью, и решил его доработать.
3. Программисты добавили в мессенджер возможность просмотра видео, загрузки фотографий, записи аудиосообщений. Они постепенно улучшают функциональность приложения, адаптируют его к требованиям рынка.

**Преимущества итеративной модели**

* *Быстрый выпуск минимального продукта*даёт возможность оперативно получать обратную связь от заказчика и пользователей. А значит, фокусироваться на наиболее важных функциях ПО и улучшать их в соответствии с требованиями рынка и пожеланиями клиента.
* *Постоянное тестирование пользователями* позволяет быстро обнаруживать и устранять ошибки.

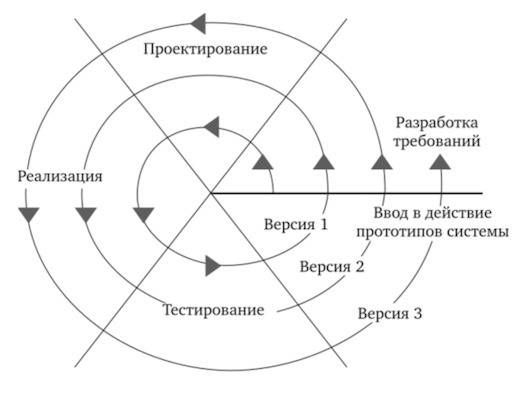
**Недостатки итеративной модели**

* *Использование на начальном этапе баз данных или серверов*— первые сложно масштабировать, а вторые не выдерживают нагрузку. Возможно, придётся переписывать большую часть приложения.
* *Отсутствие фиксированного бюджета и сроков.* Заказчик не знает, как выглядит конечная цель и когда закончится разработка.

Итеративная модель подходит для работы над *большими проектами с неопределёнными требованиями*, либо для задач с *инновационным подходом,*когда заказчик не уверен в результате.

**Spiral Model (спиральная модель)**

Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение, продолжать ли проект. Эту модель начали использовать в 1988 году.



Рассмотрим, как функционирует эта модель, на примере разработки системы «Умный дом».

1. Заказчик решил, что хочет сделать такую систему, и заказал программистам реализовать управление чайником с телефона. Они начали действовать по модели «водопад»: выслушали идею, провели анализ предложений на рынке, обсудили с заказчиком архитектуру системы, решили, как будут её реализовывать, разработали, протестировали и «выкатили» конечный продукт.
2. Заказчик оценил результат и риски: насколько нужна пользователям следующая версия продукта — уже с управлением телевизором. Рассчитал сроки, бюджет и заказал разработку. Программисты действовали по каскадной модели и представили заказчику более сложный продукт, разработанный на базе первого.
3. Заказчик подумал, что пора создать функциональность для управления холодильником с телефона. Но, анализируя риски, понял, что в холодильник сложно встроить Wi-Fi-модуль, да и производители не заинтересованы в сотрудничестве по этому вопросу. Следовательно, риски превышают потенциальную выгоду. На основе полученных данных заказчик решил прекратить разработку и совершенствовать имеющуюся функциональность, чтобы со временем понять, как развивать систему «Умный дом».

Спиральная модель похожа на инкрементную, но здесь гораздо больше времени уделяется оценке рисков. С каждым новым витком спирали процесс усложняется. Эта модель часто используется в *исследовательских проектах и там, где высоки риски.*

**Преимущества спиральной модели**

* *Большое внимание уделяется проработке рисков.*

**Недостатки спиральной модели**

* *Есть риск застрять на начальном этапе*— бесконечно совершенствовать первую версию продукта и не продвинуться к следующим.
* *Разработка длится долго и стоит дорого.*

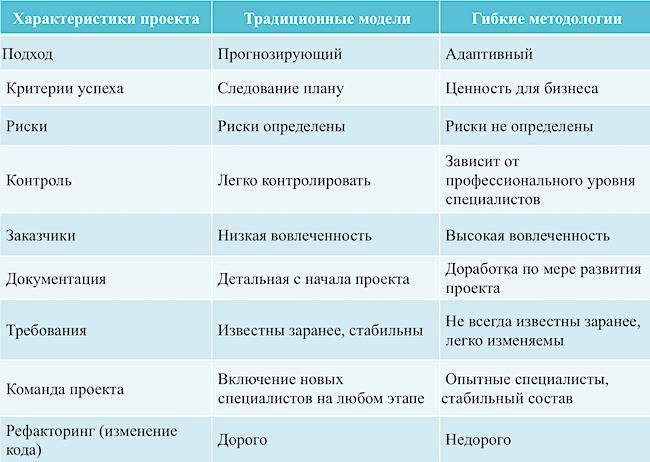
На основе итеративной модели была создана Agile — не модель и не методология, а скорее подход к разработке.

Что такое Agile?

Agile («эджайл») переводится с английского как «гибкий». Включает в себя практики, подходы и методологии, которые помогают создавать продукт более эффективно:

* экстремальное программирование (Extreme Programming, XP);
* бережливую разработку программного обеспечения (Lean);
* фреймворк для управления проектами Scrum;
* разработку, управляемую функциональностью (Feature-driven development, FDD);
* разработку через тестирование (Test-driven development, TDD);
* методологию «чистой комнаты» (Cleanroom Software Engineering);
* итеративно-инкрементальный метод разработки (OpenUP);
* методологию разработки Microsoft Solutions Framework (MSF);
* метод разработки динамических систем (Dynamic Systems Development Method, DSDM);
* метод управления разработкой Kanban.

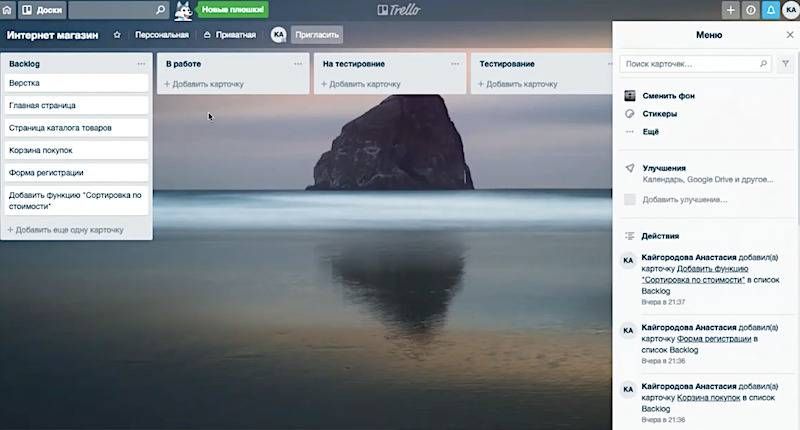
Различия между Agile и традиционным подходом к разработке мы свели в таблице:



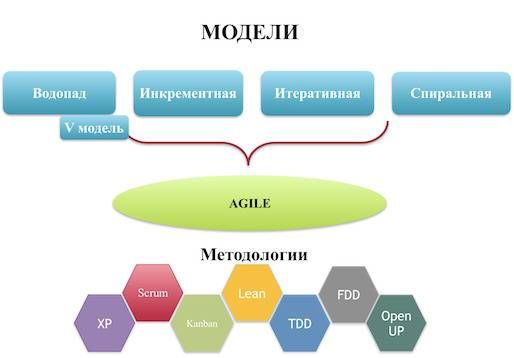
Не всё перечисленное в списке — методологии. Например, Scrum чаще называют не методологией, а фреймворком. В чём разница? Фреймворк — это более сформированная методология со строгими правилами. В скраме все роли и процессы чётко прописаны. Помимо Scrum, часто используют Kanban.

**Kanban**

Сегодня это одна из наиболее популярных методологий разработки ПО. Команда ведёт работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания.



В отличие от скрама, в канбане можно взять срочные задачи в разработку сразу, не дожидаясь начала следующего спринта. Канбан удобно использовать не только в работе, но и в личных целях — распределять собственные планы или задачи семьи на выходные, наглядно отслеживать прогресс.



# **МЕТОДОЛОГИЯ МЕНЕДЖМЕНТ**

## Методология

 – это способ разработки программного обеспечения, включающий определенные этапы и имеющий определенные особенности и идеи. Это целая система, которая определяет процесс и стиль создания продукта. Как известно, создание любой программы следует определенной методологии.

В современной разработке существует огромное количество методологий, среди которых есть наиболее распространенные. Давайте их и рассмотрим:

## RUP

RUP (Rational Unified Process) – методология разработки ПО, созданная компанией Rational Software. Читать про [RUP](http://qaevolution.ru/metodologiya-menedzhment/rup/)

## Waterfall

**Каскадная модель** (англ. waterfall model, иногда переводят, как модель «Водопад») — модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки. Читать про [Waterfall](http://qaevolution.ru/metodologiya-menedzhment/waterfall/)

## XP (Extreme Programming)

**Экстремальное программирование**(англ. Extreme Programming, XP) — одна из гибких методологий разработки программного обеспечения. Читать про [XP](http://qaevolution.ru/metodologiya-menedzhment/xp/)

## Agile

**Гибкая методология разработки** (англ. Agile software development, agile-методы) — серия подходов к разработке программного обеспечения, ориентированных на использование *интерактивной разработки*, *динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп*, состоящих из специалистов различного профиля. Читать про [Agile](http://qaevolution.ru/metodologiya-menedzhment/agile/)

## Scrum

Скрам является одной из гибких методологий разработки ПО. Основополагающим принципом данной методологии является качественный контроль разработки ПО. Читать про [Scrum](http://qaevolution.ru/metodologiya-menedzhment/scrum/)

## Canban

Канбан также является одной из гибких методологий разработки ПО.  Он во многом схож со скрамом, но тем не менее отличается от него по ряду характеристик.

Canban  – это методология разработки ПО, главным образом направленная на уменьшение выполняющейся в данный момент работы.

[](http://qaevolution.ru/wp-content/uploads/2016/01/4.jpg)

Разница между Канбан и [SCRUM](http://qaevolution.ru/metodologiya-menedzhment/scrum/):

— нет ограничений по времени ни на задачи, ни на спринты;

— задачи больше и их меньше;

— сроки на выполнение задач ставить нет прямой необходимости, но замерять их для общего представления все-таки нужно.

Стоит четко понимать, что главное отличие от SCRUM состоит в ориентации на задачи, в отличие от SCRUM, где главное —  успешное выполнение спринтов. Здесь нет никаких спринтов, команда просто работает над задачей. Когда задача выполнена – тогда она и реализуется.

Но возникает логичный вопрос: неужели команда работает без контроля? Э нет, не все так просто

В канбане существует такое понятие как Канбан доска, на которой распределяются задачи в разные столбцы.  Утрированное изображение вы можете увидеть ниже:

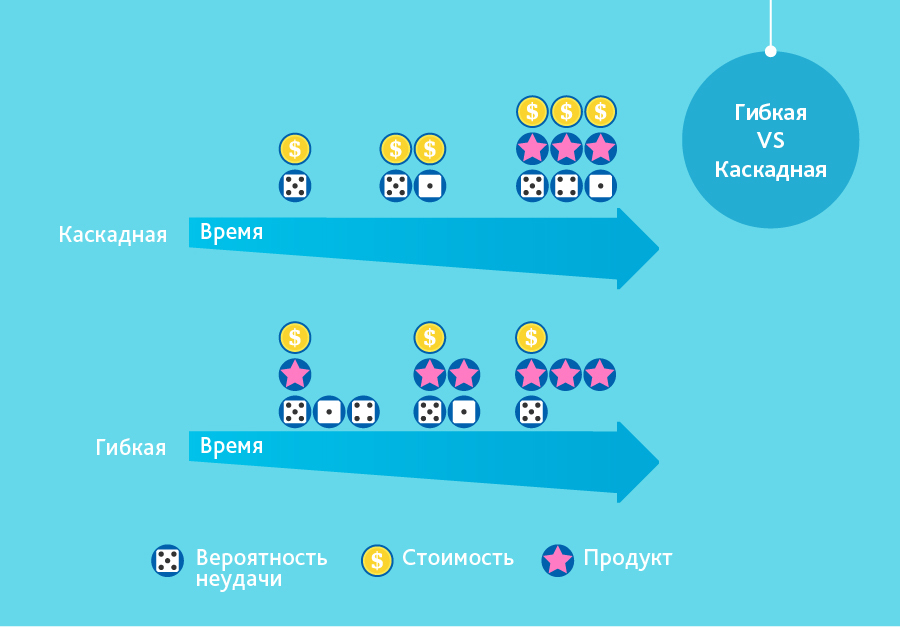
Как вы поняли, задачи постепенно переходят из одного столбца в другой. Таким образом отлично отслеживается прогресс работы.

Т.е to do – это фактически бэклог, in progress – то, что в разработке, Done – то, что выполнено.

Под каждым из столбцов можно поставить цифру, которая будет определять количество задач, которые могут быть выполнены одновременно. Данное число зависит от количества людей, задействованных в проекте, если программистов много – соответственно и задач можно на них повесть побольше (но не переусердствуйте, они же тоже люди ).

Итак, можно определить несколько ключевых моментов данной методологии:

1. Визуализация – лучше схематично лицезреть весь процесс на какой-нибудь доске или в какой-нибудь программе
2. Ограничение работы на каждом из этапов проекта, таким образом мы гарантируем выполнимость поставленных задач
3. Стоит прибегать к измерениям временных затрат на один цикл, чтобы иметь общее представление о силах команды в определенных циклах данного проекта.

[](http://qaevolution.ru/wp-content/uploads/2016/01/8edc96a71de8427bacbfa2f38a2fd98a.jpg)**Методология и различия**

В современной практике модели разработки программного обеспечения многовариантны. Нет единственно верной для всех проектов, стартовых условий и моделей оплаты. Даже столь любимая всеми нами Agile не может применяться повсеместно из-за неготовности некоторых заказчиков или невозможности гибкого финансирования. Методологии частично пересекаются в средствах и отчасти похожи друг на друга. Некоторые другие концепции использовались лишь для пропаганды собственных компиляторов и не привносили в практику ничего нового.

Следует отметить, что гибкие методологии изначально проектировались для продуктов, требования к разработке которых изменяются в течение всего процесса их создания. В этом и заключается их ключевое отличие от классических методологий, в частности, каскадной модели, для которых характерны долгие согласования по поводу малейших изменений в проекте.

Если говорить о достоинствах и недостатках гибких методологий в целом, то главным минусом станет «плавающая» оценка сроков разработки и бюджета, постоянно изменяющихся параллельно корректировке требований. К плюсам же agile-методов, безусловно, стоит отнести низкие сроки производства продукта и отсутствие простоев на время согласования проектной документации