#qa_engineer #courses2021 #VadimKsendzov

# Agile. Scrum. Kanban. XP.

Speaker of the report Inna Didenko

# Что такое Agile?

Are you ready?

Agile – это способ мышления и философия, которой соответствует совокупность подходов (Scrum, Kanban, XP) и методов управления, которые помогают:

- концентрировать внимание на целях и потребностях клиентов;
- упрощать организационные процессы;
- разбивать проект на короткие циклы с детализированной проработкой ближайших этапов;
- активно применять обратную связь;
- аргументированно увеличивать полномочия команд;
- интегрировать в повседневный образ жизни рабочие инструменты быстрого реагирования.

# Как работает Agile methodology?

Agile по праву считается гибким подходом. Согласно данной концепции сотрудники имеют право на ошибки, допущенные при планировании и оценивании проекта. Ведь невозможно предусмотреть все. Цель методологии Agile заключается в минимизации недочетов и деталей, которые можно упустить из внимания. Более того, здесь реально предусмотреть потенциальные изменения или дополнения, безболезненно встроив их в рабочий процесс.

Благодаря разбивке проекта на маленькие задачи (их еще называют пользовательскими историями User Stories), легко определить самые важные. Приоритетность является основой метода Agile, позволяя фокусировать внимание на спринте. цели спринта или определенном результате. Это дает высокую вероятность в достижении поставленных бизнес-целей. Ведь рабочая группа точно ориентирована на важность установленной задачи, не размениваясь в потоке требований и запросов.

1. минимизация рисков и потерь
2. работа в коротких циклах с постоянной обратной связью
3. самостоятельные в работе команды

Ключевая особенность

Agile–методологии заключается в создании комфортных условий для максимальной ценности, как результат работы всей команды. При этом концепция подходит как для сферы IT–технологий, так и для ведения всевозможного бизнеса.

# Показатели Agile:

Методология имеет несколько метрик, которые позволяют проанализировать эффективность на разных этапах работы. К ним относят:

1. производительность, то есть количество выполненных задач на определенных стадиях;
2. качество, которое определяется отзывами заинтересованных сторон и временем, потраченным на поддержку продукта;
3. прогноз, определяющий количество часов, доступных в последующем спринте (небольшом цикле поставки пользовательских историй);
4. бизнес ценность – индивидуальный показатель для конкретного продукта.

## Agile – семейство гибких методологий разработки.

| | | | | |
|---|---|---|---|---|
| **Ценности** | Люди и взаимодействие между ними важнее процессов | Рабочий продукт важнее документации | Сотрудничество с заказчиком важнее обсуждения условий договора с ним | Готовность к изменениям важнее следования первоначальному плану |
| **Принципы** | Раняя поставка продукта | Изменения приветствуются | Регулярные поставки | Бизнес и команда работают вместе |
| | Команде доверяют и поддерживают | Общение лицом к лицу | Работающий продукт | Поддержание постоянного темпа |
| | Внимание на качество | Минимизация лишней работы | Самоорганизация команды | Непрерывное совершенствование |

# Agile

Scrum

1

Scrum
Дает конечному пользователю работающее ПО с новыми приоритетными возможностями в жестко фиксированные по времени итерации

XP

2

XP
Это экстремальное программирование, разработка через тестирование, частые релизы, коллективное владение кодом.

Kanban

3

Kanban
Это тот же Scrum, но без привязки к срокам. Наличие визуальной доски.

# Scrum

Сотрудники компании должны извлекать уроки из полученного опыта, осваивать принципы самоорганизации, работая над решением проблемы, и анализировать свои успехи и провалы, чтобы постоянно совершенствоваться. Scrum содействует этому. Участники команды Scrum проводят собрания, используют специальные инструменты и принимают на себя особые роли, чтобы организовать работу и управлять ею.

## Спринты Scrum

При использовании scrum продукт разрабатывается в ходе нескольких итераций с фиксированной продолжительностью, которые называются спринтами и разбивают большие сложные проекты на небольшие задачи. Спринты повышают управляемость проектов, позволяют командам поставлять высококачественные продукты быстрее и чаще, а также обеспечивают большую гибкость при адаптации к изменениям

# Как планировать и выполнять спринты в scrum?

Чтобы запланировать предстоящий спринт, нужно провести собрание по планированию спринта. Планирование спринта — это мероприятие, на котором команда сообща отвечает на два основных вопроса: какую работу можно выполнить в этом спринте и как она будет выполняться?

Выбором подходящих рабочих задач для спринта занимаются совместно владелец продукта, Scrum-мастер и команда разработчиков. Владелец продукта определяет цель спринта и задачи из бэклога продукта, при выполнении которых она будет достигнута.

Затем команда создает план, согласно которому будут выполняться задачи бэклога, чтобы к окончанию спринта вся работа была завершена. Выбранные рабочие задачи и план по их выполнению называется бэклогом спринта. К концу совещания по планированию спринта команда готова приступить к работе. Для этого необходимо просто выбирать задачи из бэклога спринта и менять их статус с «В работе» на «Готово» по мере завершения работы.

Add your idea here
Add your idea here
Add your idea here
Add your idea here

В течение спринта команда собирается на ежедневные Scrum-совещания (стендапы), чтобы обсудить ход работы. Такие совещания нужны, чтобы выявить блокеры и проблемы, которые могут повлиять на достижение цели спринта.
По окончании спринта команда показывает выполненную работу на обзоре итогов спринта. Здесь можно продемонстрировать итоги работы заинтересованным сторонам и другим участникам команды до того, как они попадут в рабочую среду.
Завершите цикл спринтов на собрании — ретроспективе спринта. Здесь команда может определить области, требующие улучшения в следующем спринте. С этими сведениями можно начинать следующий цикл спринта.

# Оценка сложностей

При планировании спринта нужно хотя бы приблизительно оценить сложности работы. Команда должна понять, что можно сделать за спринт, а также сопоставить ожидаемые усилия со своими возможностями.

# Планирование спринта

Необходимо определить продолжительность и цель спринта, а также отправную точку работы. На совещании по планированию спринта определяется круг основных задач и вектор развития спринта:

- «Что». Владелец продукта ставит основную задачу (или цель) спринта и рассказывает, какие задачи из бэклога нужно выполнить, чтобы достичь этой цели. Scrum-команда решает, что удастся выполнить за грядущий спринт и что для этого нужно сделать в течение спринта.
- «Как». Команда разработчиков составляет план работ, которые необходимо выполнить для достижения цели спринта. Итоговый план спринта согласуется между командой разработчиков и владельцем продукта с точки зрения создаваемой ценности и затрачиваемых усилий.
- «Кто». Планирование спринта невозможно без участия владельца продукта и команды разработчиков. Владелец продукта ставит цель, которая зависит от искомой ценности. Команда разработчиков должна понять, может ли она достичь этой цели (и как это сделать). Если одна из этих сторон отсутствует на мероприятии, планировать спринт становится практически невозможно.
- Входные данные. Прекрасной отправной точкой для планирования спринта является бэклог продукта, так как многое из него можно включить в состав текущего спринта. Команда также должна проанализировать текущую работу, которую она выполняет в рамках инкремента, и оценить собственные возможности.
- Результаты. Самое важное, с чем команда может покинуть совещание по планированию спринта, — это понимание того, что нужно достичь за спринт и как начать продвижение к этой цели. Отобразить это можно с помощью бэклога спринта.

# Подготовка к собранию по планированию спринта

Чтобы собрание по планированию спринта прошло успешно, требуется определенная организованность. Владелец продукта должен: освежить в памяти выводы, сделанные при прошлом обзоре итогов спринта, ознакомиться с мнениями заинтересованных сторон и их концепцией продукта.

Чтобы текущая ситуация была прозрачна и понятна, бэклог продукта следует привести в соответствие с современными реалиями и скорректировать. как вариант, если каждый спринт длится две недели, можно провести совещание по улучшению бэклога в середине спринта. Команде полезно отвлечься от спринта и оценить перспективы. Это не только поможет подготовиться к планированию спринта, но и позволит взглянуть на текущую работу со стороны.

# Результат - это главное

Цель спринта показывает, что ожидается от этого спринта в общих чертах, однако нужный результат можно сформулировать и в содержании бэклога. Описать работу с точки зрения клиента можно, например, с помощью пользовательских историй.

Пользовательская история — это описание функциональной возможности ПО простыми, общими словами, составленное с точки зрения конечного пользователя. Она пишется с целью разъяснить, как именно функциональная возможность принесет пользу клиенту.

Когда в пользовательской истории прописаны однозначные измеримые результаты, можно с точностью определить, насколько успешно выполнена работа, и установить для нее критерии «готовности». Заранее необходимо прояснить как можно больше аспектов предстоящей работы, чтобы у команды было четкое представление о работе. Задать вопрос, ответ на который будет найден во время спринта, гораздо лучше, чем промолчать и затем не понимать, что нужно делать..

# Планирование спринтов:

**Состав участников:** команда разработчиков, Scrum-мастер, владелец продукта.
**Когда проходит:** в начале спринта.
**Продолжительность:** обычно до двух часов на каждую неделю итерации. То есть двухнедельный спринт нужно начинать с четырехчасового собрания по планированию.
**Agile-методика:** Scrum. (Конечно, команды Kanban тоже занимаются планированием, но они не проводят формальное планирование спринта в соответствии с фиксированным графиком итераций.)
**Назначение:** нужно проработать все мельчайшие детали работы, которую предстоит сделать. Участники команды набросают примерные задания по всем историям, багам и задачам, которые входят в спринт. Необходимо поощрять обсуждения, дабы прийти к единому взгляду на план действий. Эффективное планирование значительно повышает шансы команды успешно выполнить обязательства, которые она взяла на себя в рамках спринта.

# Ежедневный стендап:

**Состав участников:** команда разработчиков, Scrum-мастер, владелец продукта
**Когда проходит:** раз в день, как правило, утром.
**Продолжительность**: не более 15 минут.
**Agile-методика:** Scrum и Kanban.
**Назначение:** стендап нужен для того, чтобы быстро сообщить всем о ситуации в команде. Это не полноценная планерка. Каждый участник команды должен ответить на следующие вопросы.

- «Что мне удалось завершить вчера?»
- «Над чем я буду работать сегодня?»
- «Есть ли препятствия в моей работе?»

Когда отчитываешься о прогрессе за вчерашний день перед коллегами, проявляется личная ответственность.

# Обзор итогов итерации:

**Состав участников:**
**Обязательно:** команда разработчиков, Scrum-мастер, владелец продукта.
**Необязательно:** заинтересованные в проекте стороны.
**Когда проходит:** в конце спринта или по достижении контрольной точки.
**Продолжительность:** обычно 60 минут на каждую неделю итерации. То есть за двухнедельным спринтом должен следовать двухчасовой обзор итогов.
**Agile-методика:** Scrum и Kanban. Как и при планировании, Kanban-команды должны проводить обзор итогов в соответствии с контрольными точками, принятыми в команде, а не с фиксированным графиком.
**Назначение:** на собрании по обзору итогов итерации команда представляет результаты своей работы. Собрание может проходить как в свободной обстановке, так и в более формальной. На нем команда отмечает свои успехи, демонстрирует результаты работы, выполненной за итерацию, и выслушивает мнение заинтересованных в проекте сторон из первых уст. Помните: результат работы можно считать окончательным и готовым к демонстрации в ходе обзора итогов, только если он может быть представлен полностью и отвечает критериям качества команды

# Ретроспектива:

**Состав участников:** команда разработчиков, Scrum-мастер, владелец продукта.
**Когда проходит:** в конце итерации.
**Продолжительность:** обычно 45 минут на каждую неделю итерации. То есть после двухнедельного спринта нужно провести 90-минутную ретроспективу.
**Agile-методика:** Scrum и Kanban. Scrum-команды проводят ретроспективы спринта в соответствии с фиксированным графиком. Kanban-командам тоже может быть полезно время от времени проводить ретроспективы.
**Назначение:** суть Agile — в быстром получении отзывов для улучшения продукта и культуры разработки. С помощью ретроспектив команды понимают, что удалось, а что нет.
Ретроспективы — это не только время для накопленных жалоб. Ретроспективы помогают определить, что работает (и пусть команда продолжит сосредоточиваться на этих аспектах), а что нет (уделите время поиску нестандартных решений и разработке плана действий). Постоянное совершенствование — это опора и движущая сила развития в Agile-команде, и ретроспективы играют в этом ключевую роль.

## Что такое бэклог продукта?

Бэклог продукта — это перечень рабочих задач, расположенных в порядке важности, для команды разработчиков. Наиболее важные задачи расположены в начале бэклога продукта, чтобы команда понимала, какую работу следует выполнить в первую очередь.

Скорость, с которой команда выполняет задачи бэклога, не зависит от желаний владельца продукта, а он, в свою очередь, не оказывает давления на команду. Напротив, команда разработки самостоятельно выбирает задачи из бэклога продукта, когда у нее есть необходимые ресурсы, выполняя их непрерывно (Kanban) или итерациями (Scrum).

Важно регулярно корректировать его по мере выполнения программы. Владельцы продукта должны пересматривать бэклог перед каждым собранием по планированию итерации, чтобы уточнить расстановку приоритетов и внести изменения на основе выводов, сделанных в результате последней итерации.

Бэклог служит связующим звеном между владельцем продукта и командой разработчиков. Владелец продукта может в любое время поменять приоритеты в работе на основе обратной связи от клиентов, более точных прогнозов и новых требований. И все же следует избегать изменений в ходе работы, потому что они мешают команде разработчиков, негативно влияя на концентрацию, рабочий процесс и моральный дух.

Не стоит путать обзоры итогов спринта с ретроспективами.
На **обзоре итогов** наглядно демонстрируется вся сложность работы всей команды: дизайнеров, разработчиков и владельца продукта. Участники команды собираются за столом и проводят неформальные презентации, с помощью которых они рассказывают, что было сделано за прошедшую итерацию. Во время обзора принято задавать вопросы, испытывать новые функции и давать отзывы.
Сюда можно внести:
Определение критериев готовности (культура выполнения работы, определение критериев готовности для каждой рабочей задачи, критерии приемки, примечания к тестированию).
Необходимо отмечать и всячески поощрять победы команды.

**Ретроспектива** — это любое время, когда команда размышляет о прошлом, чтобы улучшить будущее. Помимо рабочих вопросов в технических и нетехнических командах, ретроспективы можно устраивать практически по любым темам.

**Scrum-мастера** — выступают в качестве коучей для остальной части команды. Руководство по scrum называет их «лидерами». Хорошие scrum-мастера преданы основам и ценностям метода scrum, но остаются гибкими и показывают команде возможности для улучшения рабочего процесса.Он помогает владельцу продукта лучше понимать и выражать ценность, управлять бэклогом, планировать работу с командой и разбивать эту работу для наиболее эффективного обучения. Команде разработчиков Scrum-мастер помогает с самоорганизацией, настраивает участников на достижение конечного результата

# Kanban

**Kanban** — это популярный подход к реализации принципов agile при разработке ПО. Методика предполагает обсуждение производительности в режиме реального времени и полную прозрачность рабочих процессов. Рабочие задачи визуально представлены на доске Kanban, что позволяет участникам команды видеть состояние каждой задачи в любой момент времени.
Визуализация.
**Главный принцип** kanban - это визуализация. Ваша главная задача - сделать визуальную доску, которую вы разобьете по необходимым для вас этапам и расположите задачу по стадии ее развития. По классике жанра есть три направления - планируется, выполняется и сделано.
Количество выполняемых задач. Должно быть понимание, сколько задач должен выполнять сотрудник/отдел за определенный срок (например, неделю/месяц.

**Фокусировка на работе**.
Правила kanban учат принципу, что задачи не нужно постоянно планировать, их нужно делать. Фокусировка на невыполненных задачах - главный приоритет. Подключение дополнительных людей, правильное использование ресурсов - все, что необходимо, чтобы выполнить нерешенные задачи.
**Улучшение**.
В связи с тем, что в kanban идет работа по принципу "минимальными партиями", проблемы в продукции или работе персонала будут видны на ранних этапах работы. Их нужно выявить и устранить. Это и есть один из главных принципов - постоянное улучшение за счет внимания к мелочам.

Add your idea here

Add your idea here

Add your idea here

Add your idea here

# Kanban–доски

## Зачем они нужны?

Работа kanban-команд строится вокруг kanban-доски, которая используется для визуализации и оптимизации рабочего процесса. Хотя некоторые команды предпочитают реальные доски, виртуальные доски давно стали обязательной функцией любого инструмента agile-разработки ПО: с ними проще отследить процессы, организовать совместную работу и доступ из разных мест. Доски нужны, чтобы визуализировать работу команды, стандартизировать процесс, а также найти и устранить блокеры и зависимости. И не важно, в какой форме они представлены — в физической или в цифровой. На стандартной Kanban-доске процесс состоит из трех шагов: «Запланировано», «В работе» и «Сделано». Однако доску можно настроить в соответствии с процессом, принятым в той или иной команде, в зависимости от ее размеров, структуры и целей. Методология Kanban основана на полной прозрачности работы и обсуждении производительности в режиме реального времени. Поэтому доска Kanban должна стать единственным достоверным источником информации о работе команды.

# Kanban–карточки

Зачем отображать работу в виде карточки на Kanban–доске? Благодаря такому наглядному представлению членам команды будет проще и удобнее отслеживать жизненный цикл рабочих задач. На Kanban–карточках отображается важная информация о конкретной рабочей задаче, доступная всей команде: имя ответственного за выполнение задачи, краткое описание выполненной работы, оценка необходимого времени и т. д. На виртуальных Kanban–досках в карточки также часто добавляют снимки экрана и другие важные для исполнителя технические детали. Когда все члены команды видят состояние каждой рабочей задачи в любой момент времени, а также всю связанную с ней информацию, повышается концентрация, обеспечивается полная прозрачность, быстрее выявляются блокеры и зависимости.

# Преимущества Kanban

**Гибкость планирования**

Kanban-команда концентрируется только на текущей работе. По завершении рабочей задачи команда забирает следующую задачу с верха бэклога. Владелец продукта может менять приоритет задач в бэклоге, не мешая работе команды, поскольку изменения происходят за пределами текущих рабочих задач.

**Сокращение времени цикла**

Продолжительность цикла — ключевой показатель для Kanban-команд. Под продолжительностью цикла понимается время прохождения рабочей задачей жизненного цикла, от начала работы над задачей до ее поставки. Оптимизировав продолжительность цикла, в будущем команда сможет с уверенностью предсказывать срок поставки задач. В методологии Kanban все члены команды отвечают за то, чтобы рабочие процессы протекали без сучка, без задоринки.

# Преимущества Kanban

Add your idea here

**Меньше узких мест**

Многозадачность убивает эффективность. Чем больше незавершенных задач, тем чаще приходится переключаться между ними, а это сказывается на сроках их завершения. Поэтому ключевой принцип Kanban состоит в ограничении объема незавершенной работы (WIP). Лимиты незавершенной работы позволяют быстро находить в работе команды узкие и проблемные места, вызванные нехваткой внимания, людей или навыков.

**Непрерывная поставка**

Непрерывная поставка (CD) предполагает частую поставку релизов продукта клиентам. Непрерывная интеграция (CI) — это практика инкрементной автоматизированной сборки и тестирования кода в течение дня. Вместе они образуют конвейер CI/CD, без которого сложно обойтись командам разработчиков (особенно командам DevOps), если они хотят быстрее выпускать качественное ПО.

Add your idea here

**Наглядность**

Одна из основных ценностей — предельное внимание к повышению эффективности команды с каждой рабочей итерацией. Графики — это визуальное средство, позволяющее командам не останавливаться на достигнутом. Если у всей команды есть доступ к данным, проще заметить (и устранить) узкие места в процессе. Kanban-команды часто используют два общих отчета: графики управления и совокупного потока.

На графике управления отображается продолжительность цикла выполнения каждой задачи, а также скользящее среднее для всей команды.

**Цель команды** — сократить время прохождения задачи по этапам рабочего процесса. Если среднее время цикла на контрольном графике снижается, то команда на верном пути.

# Сравнение досок Kanban и Scrum

Различия между Kanban и Scrum довольно незначительны. Многие сходятся во мнении, что команды Scrum используют доску Kanban, но с процессами, артефактами и ролями, принятыми в Scrum. И все же в некоторых аспектах эти две методики разительно отличаются.

- У спринтов в Scrum есть дата начала и дата окончания, в то время как в Kanban работа ведется без перерыва.
- В команде Scrum четко разграничены роли (владелец продукта, команда разработчиков и Scrum-мастер), а в Kanban формальные роли отсутствуют. Обе методики требуют от команд навыков самоорганизации.
- Доска Kanban используется на протяжении всего жизненного цикла проекта, а доска Scrum обнуляется и обновляется после каждого спринта.
- Доска Scrum содержит определенное количество заданий, которые нужно выполнить к заданному сроку.
- Доски Kanban дают больше свободы в том, что касается заданий и времени их выполнения. В зависимости от потребностей можно менять приоритеты, людей, ответственных за выполнение заданий, и содержание заданий.

# Сравнение Kanban и Scrum

Хотя методологии различаются, их принципы в целом одинаковы. Обе помогают улучшить качество продуктов и избавляют от множества проблем.
Методики Kanban предполагают непрерывность и большую подвижность, а Scrum основывается на коротких структурированных спринтах работы.

Суть **Kanban** в визуализации работы, ограничении объема незавершенной работы и достижении максимальной эффективности (или скорости). Kanban-команды стремятся максимально сократить время, которое уходит на выполнение проекта (или пользовательской истории) от начала до конца. Для этого они используют доску Kanban и непрерывно совершенствуют свой рабочий процесс.

Задача команд **Scrum** — поставить работающее ПО за ряд промежутков времени, которые называются спринтами. Они стремятся создавать циклы обучения для быстрого сбора и учета отзывов клиентов. Scrum-команды используют особые роли, создают специальные артефакты и проводят регулярные собрания, чтобы работа шла в нужном русле.

# Сравнение Scrum и Kanban

| | Scrum | Kanban |
|---|---|---|
| График | Регулярные спринты с фиксированной продолжительностью (например, 2 недели) | Непрерывный процесс |
| Подходы к релизу | В конце каждого спринта | Непрерывная поставка |
| Роли | Владелец продукта, scrum-мастер, команда разработчиков | Обязательных ролей нет |
| Ключевые показатели | Скорость | Время выполнения, время цикла, объем незавершенной работы (WIP) |
| Отношение к изменениям | В ходе спринта команды не должны вносить изменения. | Изменение может произойти в любой момент |

| SCRUM | | Канбан |
|---|---|---|
| Скрам-мастер, владелец продукта и члены команды составляют команду Скрам. | Роли в команде | Набор ролей не определен. Не обязательно, чтобы роли были кросс-функциональными. |
| Столбцы помечены, чтобы отразить периоды рабочего процесса от начала до определения, которое группа сделала для этого. | Рабочие доски | Столбцы также помечены, чтобы отображать состояния рабочего процесса, но также публикуют максимальное количество историй, разрешенных в столбце одновременно. |
| В процессах Scrum большое внимание уделяется расписанию с упорядоченным списком сюжетных моментов. Этот итеративный процесс позволяет точно оценивать рабочий процесс и эффективно управлять несколькими проектами. | Планирование / Каденция | Нет обязательных временных рамок или итераций. Хотя метод Канбан является итеративным по своей природе, ожидается, что постоянное улучшение будет происходить эволюционным образом, поскольку работа постоянно завершается. |

**Давайте проанализируем каждую конкретную тему Scrum и отличия от Канбана:**

- В Scrum разделяеют свою организацию на небольшие, функциональные, самоорганизующиеся команды. В Канбане нет жесткого требования про функциональные команды
- В Scrum есть особенные роли, а для Канбана подойдут обычные
- В Scrum ежедневное стендап-собрание — это сердечный ритм проекта. В Канбане планерки проводить жестко не требуются
- В Scrum вы должны разделить свою работу на список небольших конкретных результатов. В Канбане достаточно разделить работу на части, написать каждую часть на стикер и повесить его на стену. Они не должны быть результатами.
- В Scrum вы должны разбить время на короткие итерации фиксированной длины (1-4 недели), с разными версиями минимально-работоспособного продукта, которые вы будете демонстрировать после каждой итерации. В Канбане же работа непрерывная, а не итеративная.
- В Scrum вы должны отсортировать список поставляемых результатов по приоритету и оценить необходимые усилия для каждого элемента. В Канбане нет жесткого требования оценивать работу.
- В Scrum, основываясь на информации, полученной на проверке релиза после каждой итерации, оптимизируется план выпуска и обновляются приоритеты вместе с заказчиком. В Канбане этого не происходит.
- В Scrum у вас есть фиксированные события, такие как начало, планирование, обзор и ретроспектива. В Канбане же есть задающие ритм каденции.
- В Scrum рабочий процесс всегда должен оптимизироваться после каждой проведенной ретроспективы. В Канбане же это не обязательно.
- В Scrum должен быть список невыполненных работ по продукту и график сгорания задач, чего в Канбане нет.

**Итак, что же находится в Канбане? Давайте посмотрим на контраст со Scrum:**

- Канбан фокусируется на представлении рабочего процесса команды, давая им возможность визуализировать его и улучшить как можно скорее. Scrum имеет фиксированный процесс и церемонии.
- Канбан позволяет использовать любые именованные столбцы в вашей доске, чтобы проиллюстрировать, где находится каждый элемент, продукт или услуга в рабочем процессе. Scrum фокусируется на результатах с конкретными столбцами: “бэклог”, “бэклог спринта”, “работа в процессе” и “выполненная работа”.
- Канбан ограничивает “работу в процессе» WIP-лимитом. В Канбане необходимо установить ограничения на количество работ, которые могут выполняться в каждом столбце рабочего процесса. В Scrum нет никаких правил на этот счет.
- Одна из самых важных вещей в Канбане — это измерение среднего времени выполнения одного элемента, называемое “временем цикла”. Это очень важно, потому что это дает вам возможность оптимизировать процесс, чтобы сделать работу как можно короче и предсказуемее.
- В Канбан можно вносить изменения по мере необходимости. В Scrum изменения не должны прерывать спринт.
-

**Итог:**

И Scrum, и Kanban позволяют разбивать большие и сложные задачи на части и эффективно выполнять. Оба придают большое значение постоянному совершенствованию, оптимизации работы и процесса. И оба разделяют очень похожую направленность на хорошо заметный рабочий процесс, который держит всех членов команды в курсе о незавершенной работе и о том, что будет дальше.

# Экстремальное программирование (XP)

Экстремальное программирование или XP, eXtreme Programming — гибкая методология разработки программного обеспечения. Как и у других agile-методологий, у нее есть особенные инструменты, процессы и роли. Хотя автор XP не придумал ничего нового, а взял лучшие практики гибкой разработки и усилил до максимума. Поэтому программирование и зовется экстремальным.

Цель методики XP — справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки. Поэтому XP хорошо подходит для сложных и неопределенных проектов.

Методология XP строится вокруг четырех процессов: кодирования, тестирования, дизайна и слушания. Кроме того, экстремальное программирование имеет ценности: простоту, коммуникацию, обратную связь, смелость и уважение.

**Идея** состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменениен кода к регрессии, то есть к появлению ошибок в уже протестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

**Планирование релиза:**

Планирование в XP проводят в два этапа — планирование релиза и планирование итераций.

На планировании релиза команда программистов встречается с заказчиком, чтобы выяснить, какую функциональность он хочет получить к следующему релизу, то есть через 2-6 месяцев. Так как требования заказчика часто размытые, разработчики конкретизируют их и дробят на части, реализация которых занимает не более одного дня. Важно, чтобы заказчик разбирался в операционной среде, в которой будет работать продукт. Задачи записываются на карточки, а заказчик определяет их приоритет. Дальше разработчики оценивают, сколько времени уйдет на каждую задачу. Когда задачи описаны и оценены, заказчик просматривает документацию и дает добро на начало работы. Для успеха проекта критично, чтобы заказчик и команда программистов играли на одном поле: заказчик выбирает действительно необходимую функциональность в рамках бюджета, а программисты адекватно сопоставляют требования заказчика со своими возможностями.

В XP если команда не успевает выполнить все задачи к дате релиза, то релиз не отодвигается, а режется часть функционала, наименее важная для заказчика.

**Планирование итераций:**Планирование итераций проводится каждые две недели, иногда чаще или реже. Заказчик обязательно присутствует: он определяет функциональность на следующую итерацию и вносит изменения в требования к продукту.

**Частые релизы версий:**

В XP версии выпускаются часто, но с небольшим функционалом. Во-первых, маленький объем функциональности легко тестировать и сохранять работоспособность всей системы. Во-вторых, каждую итерацию заказчик получает часть функционала, несущую бизнес-ценность.

**Пользовательские тесты:**

Заказчик сам определяет автоматизированные приемочные тесты, чтобы проверить работоспособность очередной функции продукта. Команда пишет эти тесты и использует их для тестирования готового кода.

**Коллективное владение кодом:**

В XP любой разработчик может править любой кусок кода, т.к. код не закреплен за своим автором. Кодом владеет вся команда.

**Непрерывная интеграция кода:**

Это значит, что новые части кода сразу же встраиваются в систему — команды XP заливают новый билд каждые несколько часов и чаще. Во-первых, сразу видно, как последние изменения влияют на систему. Если новый кусок кода что-то сломал, то ошибку найти и исправить в разы проще, чем спустя неделю. Во-вторых, команда всегда работает с последней версией системы.

**Стандарты кодирования:**

Когда кодом владеют все, важно принять единые стандарты оформления, чтобы код выглядел так, как будто он написан одним профессионалом. Можно выработать свои стандарты или принять готовые.

**Метафора системы:**

Метафора системы — это ее сравнение с чем-то знакомым, чтобы сформировать у команды общее видение. Обычно метафору системы продумывает тот, кто разрабатывает архитектуру и представляет систему целиком.

**Устойчивый темп:**

XP команды работают на максимуме продуктивности, сохраняя устойчивый темп. При этом экстремальное программирование негативно относится к переработкам и пропагандирует 40-часовую рабочую неделю.

**Разработка, основанная на тестировании:**

Одна из самых трудных практик методологии. В XP тесты пишутся самими программистами, причем ДО написания кода, который нужно протестировать. При таком подходе каждый кусок функционала будет покрыт тестами на 100%. Когда пара программистов заливают код в репозиторий, сразу запускаются модульные тесты. И ВСЕ они должны сработать. Тогда разработчики будут уверены, что движутся в правильном направлении.

**Простой дизайн:**

Простой дизайн в XP означает делать только то, что нужно сейчас, не пытаясь угадать будущую функциональность. Простой дизайн и непрерывный рефакторинг дают синергетический эффект — когда код простой, его легко оптимизировать.

**Парное программирование:**

Представьте двух разработчиков за одним компьютером, работающих над одним куском функциональности продукта. Это и есть парное программирование, самая спорная практика XP. Старая поговорка «одна голова хорошо, а две лучше» отлично иллюстрирует суть подхода. Из двух вариантов решения проблемы выбирается лучший, код оптимизируется сразу же, ошибки отлавливаются еще до их совершения. В итоге имеем чистый код, в котором хорошо разбираются сразу двое разработчиков.

**Рефакторинг:**

Рефакторинг — это процесс постоянного улучшения дизайна системы, чтобы привести его в соответствие новым требованиям. Рефакторинг включает удаление дублей кода, повышение связности и снижение сопряжения. XP предполагает постоянные рефакторинги, поэтому дизайн кода всегда остается простым.

**Тестирование**.
Команда создает тесты ПЕРЕД написанием нового кода и постепенно перерабатывает старый код. Для старого кода тесты пишутся по мере необходимости: когда нужно добавить новую функциональность, исправить ошибку или переработать часть старого кода.
**Проектирование**.
Команда постепенно рефакторит старый код, обычно перед добавлением новой функциональности. Как и в случае с тестированием, рефакторинг старого кода проводят только по необходимости. При этом команде стоит сформулировать долгосрочные цели переработки кода и постепенно достигать их.
**Планирование**.
Команда должна перейти на тесное взаимодействие с заказчиком. На этом этапе важно донести до него преимущества работы в одной команде с разработчиками и интегрировать его в команду.
**Менеджмент**.
Роль менеджеров при переходе на XP — контролировать, чтобы все члены команды работали по новым правилам. Менеджер проекта принимает решение, когда расстаться с членом команды, который не справляется с работой в новых условиях, или найти нового и правильно интегрировать его в работу.
**Разработка**.
Преобразования в разработке начинаются с организации рабочих мест для программирования парами. Следующая задача — программировать парами большую часть рабочего времени, как бы тяжело это не давалось разработчикам.

# Преимущества и недостатки XP

## Методология XP вызывает много споров и критики со стороны тех, кто так и не смог ее внедрить в своей команде.

Преимущества экстремального программирования имеют смысл, когда команда полноценно использует хотя бы одну из практик XP. Итак, ради чего стоит стараться:

- заказчик получает именно тот продукт, который ему нужен, даже если в начале разработки сам точно не представляет его конечный вид
- команда быстро вносит изменения в код и добавляет новую функциональность за счет простого дизайна кода, частого планирования и релизов
- код всегда работает за счет постоянного тестирования и непрерывной интеграции
- команда легко поддерживает код, т.к. он написан по единому стандарту и постоянно рефакторится
- быстрый темп разработки за счет парного программирования, отсутствия переработок, присутствия заказчика в команде
- высокое качество кода
- снижаются риски, связанные с разработкой, т.к. ответственность за проект распределяется равномерно и уход/приход члена команды не разрушит процесс
- затраты на разработку ниже, т.к. команда ориентирована на код, а не на документацию и собрания

Несмотря на все плюсы, XP не всегда работает и имеет ряд слабых мест. Итак, экстремальное программирование — недостатки:

- успех проекта зависит от вовлеченности заказчика, которой не так просто добиться
- трудно предугадать затраты времени на проект, т.к. в начале никто не знает полного списка требований
- успех XP сильно зависит от уровня программистов, методология работает только с senior специалистами
- менеджмент негативно относится к парному программированию, не понимая, почему он должен оплачивать двух программистов вместо одного
- регулярные встречи с программистами дорого обходятся заказчикам
- требует слишком сильных культурных изменений, чтобы не контролировать каждую задачу
- из-за недостатка структуры и документации не подходит для крупных проектов
- т.к. гибкие методологии функционально-ориентированные, нефункциональные требования к качеству продукта сложно описать в виде пользовательских историй.

Экстремальное программирование — гибкая методология, в центре которой качественный работоспособный код с простой архитектурой. Ее предназначение — снизить уровень неопределенности в проектах и по-настоящему гибко реагировать на изменения требований к продукту.

Ссылки на информацию для доклада:

https://brainrain.com.ua/chto-takoe-agile/

https://www.cprime.com/resources/blog/3-differences-between-scrum-and-kanban-you-need-to-know/

https://habr.com/ru/post/464861/

https://resources.kanban.university/wp-content/uploads/2021/09/The-Official-Kanban-Guide_Russian_A4.pdf

https://www.atlassian.com/ru/agile/kanban

https://scrumtrek.ru/blog/agile-scrum/4029/metodologiya-agile/

https://www.atlassian.com/ru/agile

https://worksection.com/blog/extreme-programming.html