**Модели разработки ПО**

Чтобы лучше разобраться в том, как тестирование соотносится с программированием и иными видами проектной деятельности, для начала рассмотрим самые основые — модели разработки ПО (как часть жизненного цикла ПО). При этом сразу подчеркнём, что разработка ПО является лишь частью жизненного цикла ПО, и здесь мы говорим именно о разработке.

Материал данной главы относится, скорее, к дисциплине «управление проектами», потому здесь рассмотрен крайне сжато: пожалуйста, не воспринимайте его как исчерпывающее руководство — здесь едва ли рассмотрена и сотая доля процента соответствующей предметной области.

Выбор модели разработки ПО серьёзно влияет на процесс тестирования, определяя выбор стратегии, расписание, необходимые ресурсы и т.д.

Моделей разработки ПО много, но, в общем случае, классическими можно считать **каскадную, v-образную, итерационную, инкрементальную, спиральную и гибкую.**

Знать и понимать модели разработки ПО необходимо затем, чтобы уже с первых дней работы понимать, что происходит вокруг, что, зачем и почему Вы делаете. Многие начинающие тестировщики отмечают, что ощущение бессмысленности происходящего посещает их, даже если текущие задания интересны. Чем полнее вы будете представлять картину происходящего на проекте, тем яснее Вам будет виден ваш собственный вклад в общее дело и смысл того, чем вы занимаетесь.

Ещё одна важная вещь, которую следует понимать, состоит в том, что никакая модель не является догмой или универсальным решением. Нет идеальной модели. Есть та, которая хуже или лучше подходит для конкретного проекта, конкретной команды, конкретных условий.

**Каскадная (водопадная) модель**сейчас представляет, скорее, исторический интерес, т.к. в современных проектах практически не применима. Она предполагает однократное выполнение каждой из фаз проекта, которые, в свою очередь, строго следуют друг за другом (Рис. 1.2). Очень упрощённо можно сказать, что, в рамках этой модели, в любой момент времени команде «видна» лишь предыдущая и следующая фаза. В реальной же разработке ПО приходится «видеть весь проект целиком» и возвращаться к предыдущим фазам, чтобы исправить недоработки или что-то уточнить.

**Каскадная модель (waterfall)**

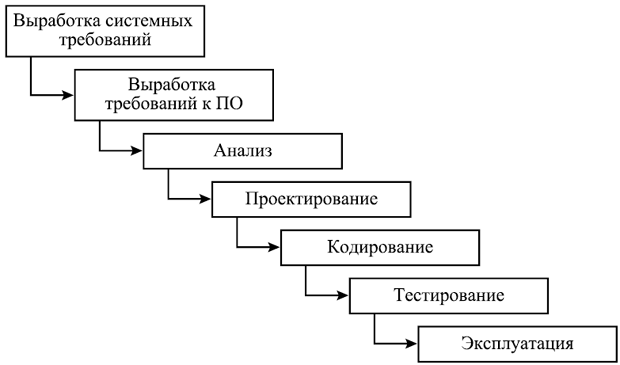


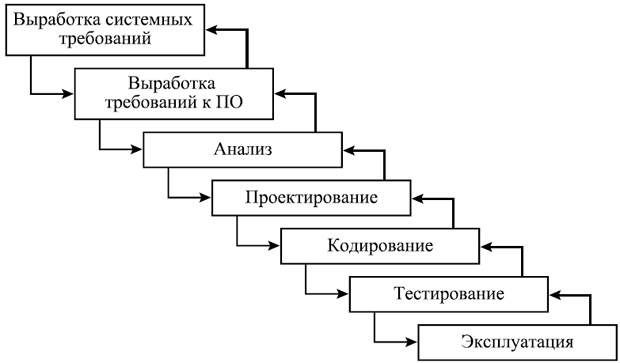
Рис. 1.2. Каскадная (водопадная) модель

**Особенности каскадной модели:**

— высокий уровень формализации процессов;  
— большое количество документации;  
— жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.  
**Минусы:**  
• Waterfall-проект должен постоянно иметь актуальную документацию. Обязательная актуализация проектной документации. Избыточная документация.  
• Очень не гибкая методология.  
• Может создать ошибочное впечатление о работе над проектом (например, фраза «45% выполнено» не несёт за собой никакой полезной информации, а является всего лишь инструментов для менеджера проекта).  
• У заказчика нет возможности ознакомиться с системой заранее и даже с «Пилотом» системы.  
• У пользователя нет возможности привыкать к продукту постепенно.  
• Все требования должны быть известны в начале жизненного цикла проекта.  
• Возникает необходимость в жёстком управлении и регулярном контроле, иначе проект быстро выбьется из графиков.  
• Отсутствует возможность учесть переделку, весь проект делается за один раз.  
**Плюсы:**  
• Высокая прозрачность разработки и фаз проекта.  
• Чёткая последовательность.  
• Стабильность требований.  
• Строгий контроль менеджмента проекта.  
• Облегчает работу по составлению плана проекта и сбора команды проекта.  
• Хорошо определяет процедуру по контролю качества.

**«Водоворот» или каскадная модель с промежуточным контролем**

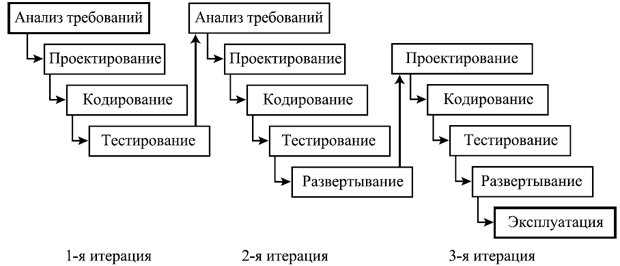
В этой модели предусмотрен промежуточный контроль за счет обратных связей. Но это достоинство порождает и недостатки. Затраты на реализацию проекта при таком подходе возрастают практически в 10 раз. Эта модель, как Вы уже поняли, является незначительной модификацией предыдущей и относится к первой группе.



При реальной работе, в соответствии с моделью, допускающей движение только в одну сторону, обычно возникают проблемы при обнаружении недоработок и ошибок, сделанных на ранних этапах. Но еще более тяжело иметь дело с изменениями окружения, в котором разрабатывается ПО (это могут быть изменения требований, смена подрядчиков, изменение политики разрабатывающей или эксплуатирующей организации, изменения отраслевых стандартов, появление конкурирующих продуктов и пр.).

**Итеративная модель**

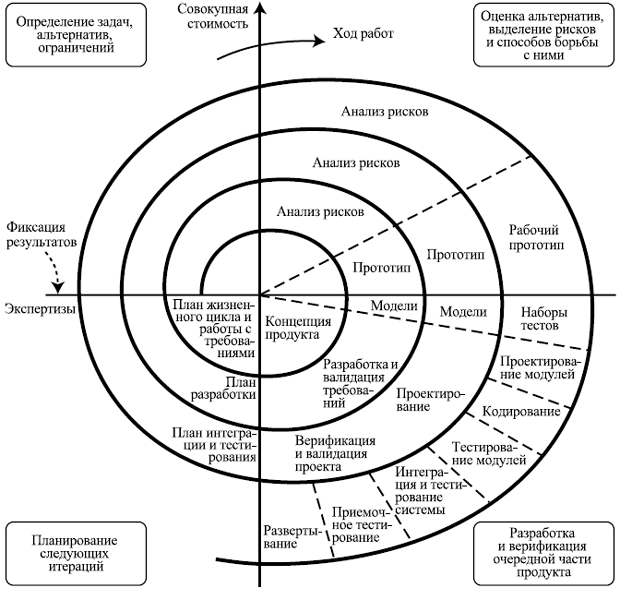
**Итеративные или инкрементальные модели** (известно несколько таких моделей) предполагают разбиение создаваемой системы на набор кусков, которые разрабатываются с помощью нескольких последовательных проходов всех работ или их части.



Каскадная модель с возможностью возвращения на предшествующий шаг, при необходимости пересмотреть его результаты, становится итеративной.  
Итеративный процесс предполагает, что разные виды деятельности не привязаны намертво к определенным этапам разработки, а выполняются по мере необходимости, иногда повторяются, до тех пор, пока не будет получен нужный результат.  
Вместе с гибкостью и возможностью быстро реагировать на изменения, итеративные модели привносят дополнительные сложности в управление проектом и отслеживание его хода. При использовании итеративного подхода значительно сложнее становится адекватно оценить текущее состояние проекта и спланировать долгосрочное развитие событий, а также предсказать сроки и ресурсы, необходимые для обеспечения определенного качества результата.

**Спиральная модель жизненного цикла программного обеспечения**

Данная модель прекрасно сочетает в себе прототипирование и проектирование по стадиям. И из восходящей и нисходящей концепций в эту модель было взято все лучшее.

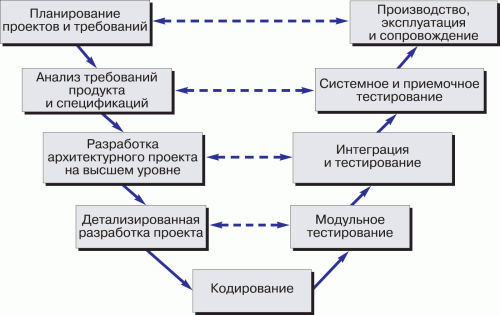


**Преимущества модели:**

1. Результат достигается в кратчайшие сроки.
2. Конкурентоспособность достаточно высокая.
3. При изменении требований не придется начинать все с «нуля».  
   Но у этой модели есть один существенный **недостаток: невозможность регламентирования стадий выполнения.**

**V модель — разработка через тестирование**

Данная модель имеет более приближенный к современным методам алгоритм, однако все еще имеет ряд недостатков. Является одной из основных практик экстремального программирования и предполагает регулярное тестирование продукта во время разработки.



V-модель обеспечивает поддержку в планировании и реализации проекта. В ходе проекта ставятся следующие задачи:  
• **Минимизация рисков:**V-образная модель делает проект более прозрачным и повышает качество контроля проекта путём стандартизации промежуточных целей и описания соответствующих им результатов и ответственных лиц. Это позволяет выявлять отклонения и риски в проекте на ранних стадиях и улучшает качество управления проектов, уменьшая риски.  
• **Повышение и гарантии качества:** V-Model —стандартизованная модель разработки, что позволяет добиться от проекта результатов желаемого качества. Промежуточные результаты могут быть проверены на ранних стадиях. Универсальное документирование облегчает читаемость, понятность и проверяемость.  
• **Уменьшение общей стоимости проекта:** ресурсы на разработку, производство, управление и поддержку могут быть заранее просчитаны и проконтролированы. Получаемые результаты также универсальны и легко прогнозируются. Это уменьшает затраты на последующие стадии и проекты.  
• **Повышение качества коммуникации между участниками проекта:** универсальное описание всех элементов и условий облегчает взаимопонимание всех участников проекта. Таким образом, уменьшаются неточности в понимании между пользователем, покупателем, поставщиком и разработчиком.

**Модель на основе разработки прототипа**

Данная модель основывается на разработке прототипов и прототипирования продукта и относится ко второй группе.

Прототипирование используется на ранних стадиях жизненного цикла программного обеспечения:

* Прояснить неясные требования (прототип UI).
* Выбрать одно из ряда концептуальных решений (реализация сценариев).
* Проанализировать осуществимость проекта.

**Классификация прототипов:**

* **Горизонтальные прототипы** — моделирует исключительно UI, не затрагивая логику обработки и базу данных.
* **Вертикальные прототипы** — проверка архитектурных решений.
* **Одноразовые прототипы**— для быстрой разработки.
* **Эволюционные прототипы** — первое приближение эволюционной системы.

Вкратце можно выразить суть моделей разработки ПО таблицей 1.3.

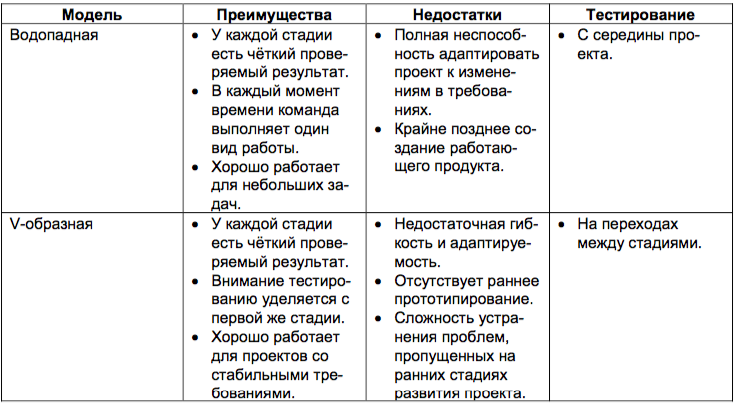




Таблица 1.3.— Сравнение моделей разработки ПО

**Agile (идеология) -манифест разработки программного обеспечения**

Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, занимаясь разработкой непосредственно и помогая в этом другим. Благодаря проделанной работе мы смогли осознать, что:

* Люди и взаимодействие важнее процессов и инструментов.
* Работающий продукт важнее исчерпывающей документации.
* Сотрудничество с заказчиком важнее согласования условий контракта.
* Готовность к изменениям важнее следования первоначальному плану.

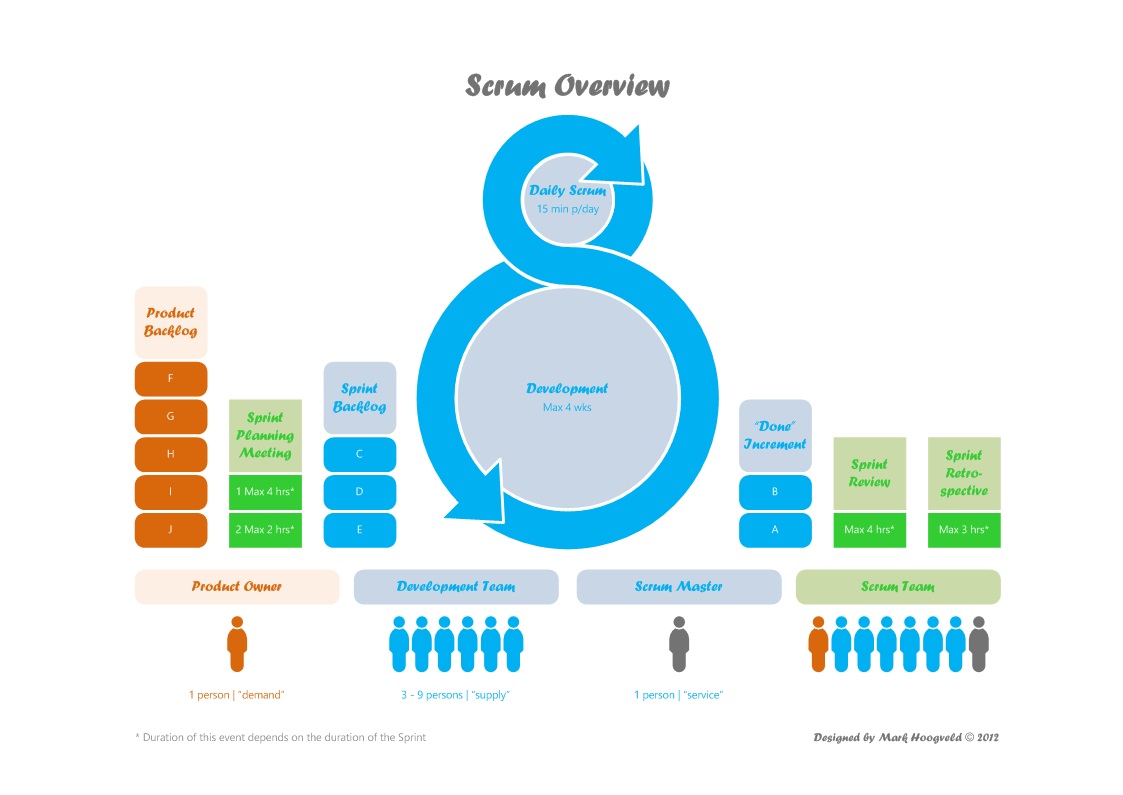
То есть, не отрицая важности того, что справа, мы всё-таки больше ценим то, что слева.

**Основополагающие принципы Agile-манифеста**

Мы следуем таким принципам:

* Наивысшим приоритетом для нас является удовлетворение потребностей заказчика благодаря регулярной и ранней поставке ценного программного обеспечения.
* Изменение требований приветствуется даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
* Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
* На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
* Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
* Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
* Работающий продукт — основной показатель прогресса.
* Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.
* Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
* Простота — искусство минимизации лишней работы — крайне необходима.
* Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
* Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

**SCRUM**



**Scrum (Скрам)**— это не аббревиатура, этот термин взят из регби, который обозначает схватку вокруг мяча.

Сам термин Scrum можно определить так — это методология управления проектами, которая построена на принципах тайм-менеджмента. Основной ее особенностью является вовлеченность в процесс всех участников, причем у каждого участника есть своя определенная роль. Суть в том, что не только команда работает над решением задачи, но все те, кому интересно решение задачи. Не просто поставили задачу и расслабились, а постоянно «работают» с командой и эта работа не означает только постоянный контроль.

Основные термины, которые используются в методологии:

* **Владелец продукта (Product owner)**— человек, который имеет непосредственный интерес в качественном конечном продукте, он понимает, как это продукт должен выглядеть/работать. Этот человек не работает в команде, он работает на стороне заказчика/клиента (это может быть как другая компания, так и другой отдел), но этот человек работает с командой. И это тот человек, который расставляет приоритеты для задач.
* **Scrum-мастер**— это человек, которого можно назвать руководителем проекта, хотя это не совсем так. Главное, что это человек «зараженный Scrum-бациллой» настолько, что несет ее как своей команде, так и заказчику и, соответственно, следит за тем, чтобы все принципы Scrum соблюдались.
* **Scrum-команда**— это команда, которая принимает все принципы Scrum и готова с ними работать.
* **Спринт**— отрезок времени, который берется для выполнения определенного (ограниченного) списка задач. Рекомендуется брать 2-4 недели (длительность определяется командой один раз).
* **Бэклог (backlog)**— это список всех работ. Можно сказать, это ежедневник общего пользования. Различают 2 вида бэклогов: Product-бэклог и спринт-бэклог.

**1.Product-бэклог**— это полный список всех работ, при реализации которых мы получим конечный продукт.

**2. Спринт-бэклог**— это список работ, который определила команда и согласовала с Владельцем продукта на ближайший отчетный период (спринт). Задания в спринт-бэклог берутся из product-бэклога.

* **Планирование спринта**— это совещание, на котором присутствуют все (команда, Scrum-мастер, Владелец продукта). В течение этого совещания Владелец продукта определяет приоритеты заданий, которые он хотел бы увидеть выполненными по истечении спринта. Команда оценивает по времени, сколько из желаемого они могут выполнить. В итоге получается список заданий, который не может меняться в течение спринта и к концу спринта должен быть полностью выполнен.

**Пример работы PR-агентства.** Как бы это могло выглядеть, если бы они работали по Scrum:

Компания-клиент «Икс» хочет провести через 2 месяца масштабное мероприятие для своих партнеров и журналистов. Услуги по организации такого мероприятия компания «Икс» заказала у агентства «Зет». Компанию «Икс» представляет PR-менеджер, который отвечает за организацию мероприятия со стороны клиента. В терминологии Scrum этот человек называется "Владелец продукта". Со стороны агентства за организацию мероприятия отвечает account-менеджер (Scrum-мастер), в подчинении которого находится команда (Scrum-команда). На совместном совещании (планировании спринта) компания и агентство решают, что они будут отчитываться/планировать каждые 2 недели (длина спринта). На первые 2 недели они запланировали список задач (спринт-бэклог), однако команда оценила, что не все из этого списка они успеют выполнить. Тогда PR-менеджер (он же Владелец продукта) говорит, какие из этого списка задач более приоритетные на ближайшие 2 недели, после чего команда берется за выполнение заданий. Единственное, что здесь должно быть учтено, что, на момент планирования первого спринта, должен быть спланирован весь список заданий на 2 месяца (product-бэклог), чтобы не получилось так, что к моменту проведения мероприятия что-то не выполнено.

**Жизненный цикл спринта**

\*\*1.Планирование спринта.\*\*

В начале каждого спринта проводится планирование этого самого спринта. В планировании спринта участвуют заказчики, пользователи, менеджмент, Product Owner, Скрам Мастер и команда.

Планирование спринта состоит из двух последовательных митингов.

1)Планирование спринта, митинг первый.

**Участники:**команда, Product Owner, Scrum Master, пользователи, менеджмент

**Цель:** Определить цель спринта (Sprint Goal) и Sprint Backlog -функциональность, которая будет разработана в течение следующего спринта для достижения цели спринта.

**Артефакт**: Sprint Backlog

2) Планирование спринта, митинг второй.

**Участники:**Скрам Мастер, команда

**Цель:** определить, как именно будет разрабатываться определенная функциональность для того, чтобы достичь цели спринта. Для каждого элемента Sprint Backlog определяется список задач и оценивается их продолжительность.

**Артефакт:** в Sprint Backlog появляются задачи

Если в ходе спринта выясняется, что команда не может успеть сделать запланированное на спринт, то Скрам Мастер, Product Owner и команда встречаются и выясняют, как можно сократить scope работ и при этом достичь цели спринта.

\*\*2.Остановка спринта \(Sprint Abnormal Termination\).\*\*

Остановка спринта производится в исключительных ситуациях. Спринт может быть остановлен до того, как закончатся отведенные 30 дней. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить Product Owner, если необходимость в достижении цели спринта исчезла.

После остановки спринта проводится митинг с командой, где обсуждаются причины остановки спринта. После этого начинается новый спринт: производится его планирование и стартуются работы.

\*\*3.Daily Scrum Meeting.\*\*

Этот митинг проходит каждое утро в начале дня. Он предназначен для того, чтобы все члены команды знали, кто и чем занимается в проекте. Длительность этого митинга строго ограничена и не должна превышать 15 минут. Цель митинга — поделиться информацией. Он не предназначен для решения проблем в проекте. Все требующие специального обсуждения вопросы должны быть вынесены за пределы митинга.

Скрам митинг проводит Скрам Мастер. Он по кругу задает вопросы каждому члену команды:

• Что сделано вчера?

• Что будет сделано сегодня?

• С какими проблемами столкнулся?

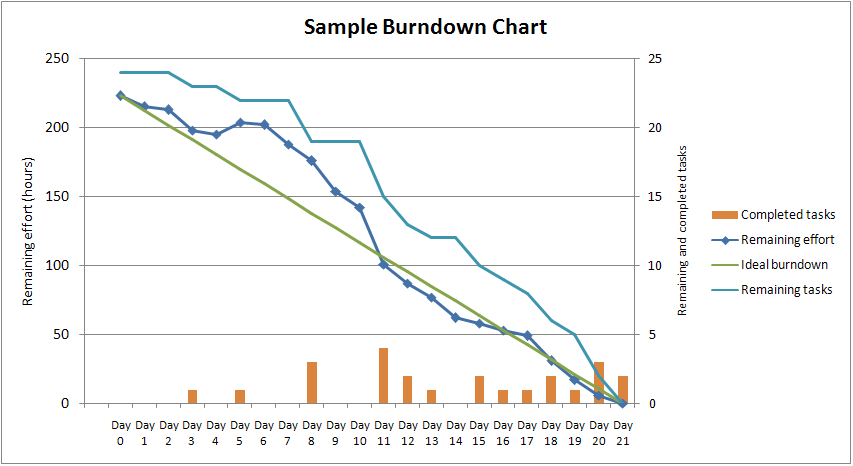
Скрам Мастер собирает все открытые для обсуждения вопросы в виде Action Items, например, в формате что/кто/когда, например:

• Обсудить проблему с отрисовкой контрола.

• Петя и Вася.

• Сразу после скрама.

**Диаграмма сгорания задач (Burndown chart)**



Диаграмма, которая показывает количество сделанной и оставшейся работы. Обновляется ежедневно с тем, чтобы в простой форме показать подвижки в работе над спринтом. График должен быть общедоступен.  
Существуют разные виды диаграммы:  
• **Диаграмма сгорания работ для спринта** — показывает, сколько уже задач сделано и сколько ещё остаётся сделать в текущем спринте.  
• **Диаграмма сгорания работ для выпуска проекта** показывает, сколько уже задач сделано и сколько ещё остаётся сделать до выпуска продукта (обычно строится на базе нескольких спринтов).

**Ретроспектива**

В конце каждого Спринта Скрам Команда собирается на ретроспективу. **Цель**: пересмотреть качество существующих процессов, взаимоотношения людей и применяемые инструменты. Команда определяет, что прошло хорошо, а что прошло не очень, а также выявляет потенциальные возможности для улучшений. Они создают план улучшений на будущее.

**Канбан**

Термин "канбан" имеет дословный перевод: «Кан» значит видимый, визуальный и «бан» - карточка или доска. На заводах Тойота карточки "канбан" используются повсеместно для того, чтобы не загромождать склады и рабочие места заранее созданными запчастями. Например, представьте, что Вы ставите двери на Тойота Королла. У Вас возле рабочего места находится пачка из 10 дверей. Вы их ставите одну за другой на новые машины и, когда в пачке остается 5 дверей, то Вы знаете, что пора заказать новые двери. Вы берете карточку "канбан", пишете на ней заказ на 10 дверей и относите ее тому, кто делает двери. Вы знаете, что он их сделает как раз к тому моменту, как у Вас закончатся оставшиеся 5 дверей. И именно так и происходит: когда Вы ставите последнюю дверь, прибывает пачка из 10 новых дверей. И так постоянно: Вы заказываете новые двери только тогда, когда они вам нужны.

А теперь представьте, что такая система действует на всём заводе. Нигде нет складов, где запчасти лежат неделями и месяцами. Все работают только по запросу и производят именно столько запчастей, сколько запрошено. Если вдруг заказов стало больше или меньше, то система сама легко подстраивается под изменения.

Основная задача карт "канбан" в этой системе — это уменьшение количества «выполняющейся в данный момент работы» (work in progress).

Например, на всю производственную линию может быть выделено ровно 10 карточек для дверей. Это значит, что в каждый момент времени на линии не будет больше 10 готовых дверей. Когда заказывать новые двери и сколько — это задача для того, кто их устанавливает. Только он знает свои потребности и только он может помещать заказы производителю дверей, но он всегда ограничен числом 10.

Этот метод Бережливого производства (Lean manufacturing) был придуман в Тойота, и сейчас многие производственные компании по всему миру его внедряют или уже внедрили.

Но это всё относится к производству, а не к разработке программного обеспечения.

А что же такое канбан- разработка применительно к ПО и чем она отличается от других гибких методологий, будь то SCRUM или XP?

Во-первых, нужно сразу понять, что канбан — это не конкретный процесс, а система ценностей. Как, впрочем, и SCRUM с XP. Это значит, что никто Вам не скажет, что и как делать по шагам.

Во-вторых, весь канбан можно описать одной простой фразой — «уменьшение выполняющейся в данный момент работы (work in progress)».

В-третьих, канбан — это даже еще более «гибкая» методология, чем SCRUM и XP. Это значит, что она не подойдет всем командам и для всех проектов. И это также значит, что команда должна быть еще более готовой к гибкой работе, чем даже команды, использующие SCRUM и XP.

**Разница между канбан и SCRUM:**

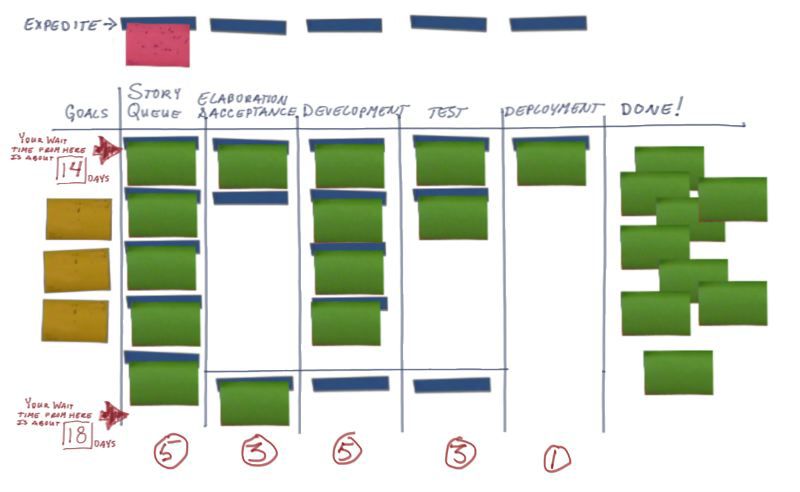
* В канбан нет таймбоксов ни на что (ни на задачи, ни на спринты).
* В канбан задачи больше и их меньше.
* В канбан оценки сроков на задачу опциональные или вообще их нет.
* В канбан «скорость работы команды» отсутствует и считается только среднее время на полную реализацию задачи.

Канбан-разработка отличается от SCRUM, в первую очередь, ориентацией на задачи. Если в SCRUM основная ориентация команды — это успешное выполнение спринтов (надо признать, что это так), то в канбан на первом месте задачи.

Спринтов никаких нет, команда работает над задачей с самого начала и до завершения. Деплоймент задачи делается тогда, когда она готова. Презентация выполненной работы — тоже. Команда не должна оценивать время на выполнение задачи, ибо это имеет мало смысла и почти всегда ошибочна вначале.

Если менеджер верит команде, то зачем иметь оценку времени? Задача менеджера — это создать приоритезированный пул задач, а задача команды — выполнить как можно больше задач из этого пула. Всё. Никакого контроля не нужно. Всё, что нужно от менеджера — это добавлять задачи в этот пул или менять им приоритет. Именно так он управляет проектом.

Команда для работы использует канбан-доску. Например, она может выглядеть так:



Столбцы слева направо:

* **Цели проекта:**необязательный, но полезный столбец. Сюда можно поместить высокоуровневые цели проекта, чтобы команда их видела и все про них знали. Например, «Увеличить скорость работы на 20%» или «Добавить поддержку Windows 7».
* **Очередь задач:**тут хранятся задачи, которые готовы к тому, чтобы начать их выполнять. Всегда для выполнения берется верхняя, самая приоритетная задача и ее карточка перемещается в следующий столбец.
* **Проработка дизайна:**этот и остальные столбцы до «Закончено» могут меняться, т.к. именно команда решает, какие шаги проходит задача до состояния «Закончено». Например, в этом столбце могут находиться задачи, для которых дизайн кода или интерфейса еще не ясен и обсуждается. Когда обсуждения закончены, задача передвигается в следующий столбец.
* **Разработка:**тут задача висит до тех пор, пока разработка фичи не завершена. После завершения она передвигается в следующий столбец. Или ,если архитектура не верна или неточна, задачу можно вернуть в предыдущий столбец.
* **Тестирование:**в этом столбце задача находится, пока она тестируется. Если найдены ошибки — возвращается в Разработку. Если нет — передвигается дальше.
* **Деплоймент:**у всех проектов свой деплоймент. У кого-то это значит выложить новую версию продукта на сервер, а у кого-то — просто закомитить код в репозиторий.
* **Закончено:**сюда стикер попадает только тогда, когда все работы по задаче закончены полностью.

В любой работе случаются срочные задачи. Запланированные или нет, но такие, которые надо сделать прямо сейчас. Для таких можно выделить специальное место (на картинке отмечено, как «Expedite»). В Expedite можно поместить одну срочную задачу и команда должна начать ее выполнять немедленно и завершить как можно быстрее. Но может быть только одна такая задача! Если появляется еще одна — она должна быть добавлена в «Очередь задач».

А теперь самое важное. Видите цифры под каждым столбцом? Это число задач, которые могут быть одновременно в этих столбцах. Цифры подбираются экспериментально, но, считается, они должны зависеть от числа разработчиков в команде. Например, если вы имеете 8 программистов в команде, то в строку «Разработка» вы можете поместить цифру 4. Это значит, что одновременно программисты будут делать не более 4-х задач, значит, у них будет много причин для общения и обмена опытом. Если вы поставите туда цифру 2, то 8 программистов, занимающихся двумя задачами, могут заскучать или терять слишком много времени на обсуждениях. Если поставить 8, то каждый будет заниматься своей задачей и некоторые задачи будут задерживаться на доске надолго, а ведь главная задача канбан — это уменьшение времени прохождения задачи от начала до стадии готовности.

Никто не даст точный ответ, какими должны быть эти лимиты, но попробуйте, для начала, разделить число разработчиков на 2 и посмотреть, как это работает в вашей команде. Потом эти числа можно подогнать под вашу команду. Под «разработчиками» понимаются не только программисты, но и другие специалисты. Например, для столбца «Тестирование» разработчики — это тестеры, т.к. тестирование — это их обязанность.

**Экстремальное программирование (XP)**

Двенадцать основных приёмов экстремального программирования (по первому изданию книги Extreme programming explained) могут быть объединены в четыре группы:

**1.Короткий цикл обратной связи (Fine-scale feedback):**

• Разработка через тестирование (Test-driven development).

• Игра в планирование (Planning game).

• Заказчик всегда рядом (Whole team, Onsite customer).

• Парное программирование (Pair programming).

**2.Непрерывный, а не пакетный процесс:**

• Непрерывная интеграция (Continuous integration).

• Рефакторинг (Design improvement, Refactoring).

• Частые небольшие релизы (Small releases).

**3.Понимание, разделяемое всеми:**

• Простота (Simple design).

• Метафора системы (System metaphor).

• Коллективное владение кодом (Collective code ownership) или выбранными шаблонами проектирования (Collective patterns ownership).

• Стандарт кодирования (Coding standard or Coding conventions).

**4.Социальная защищенность программиста (Programmer welfare):**

• 40-часовая рабочая неделя (Sustainable pace, Forty-hour week).

**RATIONAL UNIFIED PROCESS (RUP)**

**RATIONAL UNIFIED PROCESS (RUP)** — методология разработки программного обеспечения, созданная компанией Rational Software.



**В основе методологии лежат 6 основных принципов:**

* Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.
* Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.
* Ранняя идентификация и непрерывное устранение возможных рисков.
* Концентрация на выполнении требований заказчиков к исполняемой программе.
* Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.
* Постоянное обеспечение качества на всех этапах разработки проекта.

Использование методологии RUP направлено на итеративную модель разработки. Особенность методологии состоит в том, что степень формализации может меняться в зависимости от потребностей проекта. Можно создавать все требуемые документы и достигнуть максимального уровня формализации, по окончании каждого этапа и каждой итерации а можно создавать только необходимые для работы документы, вплоть до полного их отсутствия. За счет такого подхода к формализации процессов методология является достаточно гибкой и широко популярной. Данная методология применима как в небольших и быстрых проектах, где за счет отсутствия формализации требуется сократить время выполнения проекта и расходы, так и в больших и сложных проектах, где требуется высокий уровень формализма, например, с целью дальнейшей сертификации продукта. Это преимущество дает возможность использовать одну и ту же команду разработчиков для реализации различных по объему и требованиям.