Тестирование. Фундаментальная теория. Часть 2 — Методологии разработки ПО

Gennadii Mishchevskii, SDET Lead, QA Manager в Corva.AI

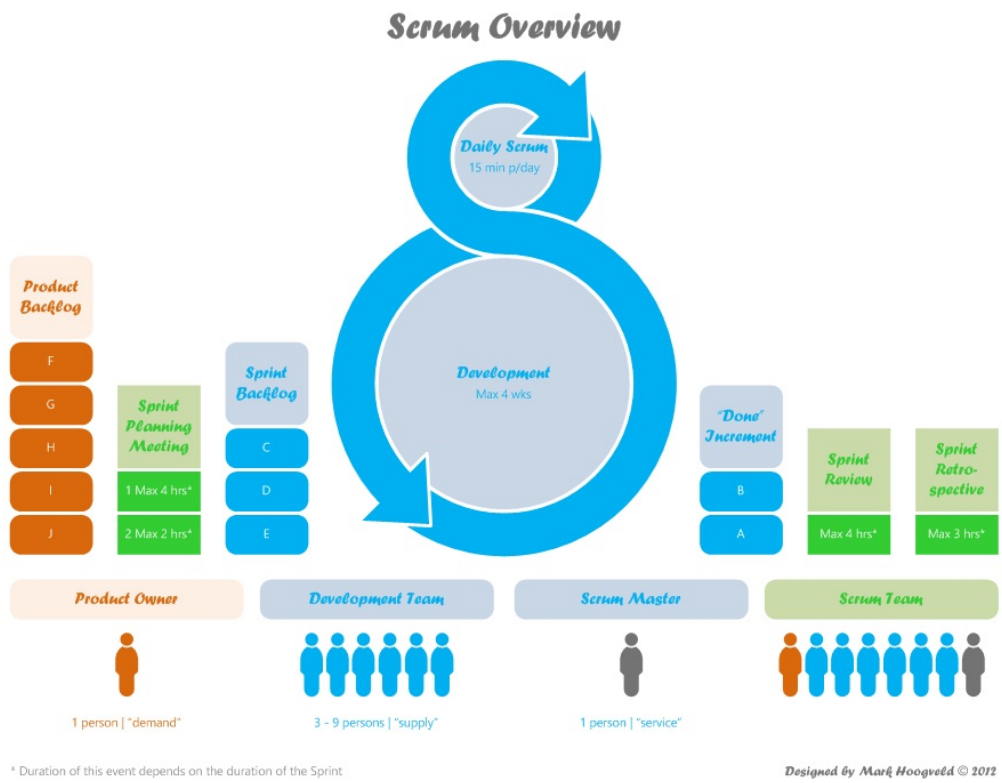
Усі статті, обговорення, новини про тестування — в одному місці. Підписуйтеся на QA DOU!

Наконец-то дошли руки до второй шпоры. Первая лежит здесь. Эта будет посвящена методологиям разработки и так же будет полезна новичкам в тестировании, ибо это так же часто спрашивают на собеседованиях.
И, как всегда, критика, уточнения, дополнения are extremely welcome.

Виды методологий разработки ПО:

• **Каскадная или поэтапная разработка** (в некоторых источниках её называют «водопадной моделью») — процесс создания программного обеспечения представляет собой поток, последовательно проходящий фазы анализа, проектирования, реализации, тестирования, интеграции и поддержки. Так обычно строится работа над крупными проектами с длительным сроком внедрения.
• **Итеративная или инкрементная (эволюционная) модель** приращения продукта позволяет параллельно выполнять ряд задач с непрерывным анализом результатов и корректировкой предыдущих этапов работы. Это более «скоростная» разработка для большого штата квалифицированных программистов.
• **Спиральная методика** характеризуется прохождением проектом повторяющегося цикла в каждой фазе развития: планирование — реализация — проверка — оценка (англ. plan-do-check-act cycle). Так обычно создаются проекты, с окончательно не сформированным видением результата, либо требующие ультрасрочного внедрения по этапам.
• **Гибкая методология Agile-разработки** — т.н. быстрая разработка без ущерба качеству, когда во главу угла ставится работающий продукт, а не его документация. Наиболее современный неформализованный подход к созданию ПО, в процессе которого реагирование на изменения ценятся выше строгого следования плану. Для молодых стремительно развивающихся проектов, которые с каждой итерацией программного обеспечения по сути готовы к его релизу.

SCRUM



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

Scrum — это методология управления проектами, которая построена на принципах тайм-менеджмета. 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 ее 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» на столько, 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, что это ежедневник общего пользования
Различают 2 вида бэклогов: Product-бэклог и спринт-бэклог.
Product-бэклог — это полный список всех работ, при реализации которых мы получим конечный продукт.
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

Планирование спринта

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

Планирование спринта состоит из двух последовательных митингов.

Планирование спринта, митинг первый

Участники: команда, Product Owner, Scrum Master, пользователи, менеджемент

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

Артефакт: Sprint Backlog

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.

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.

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.

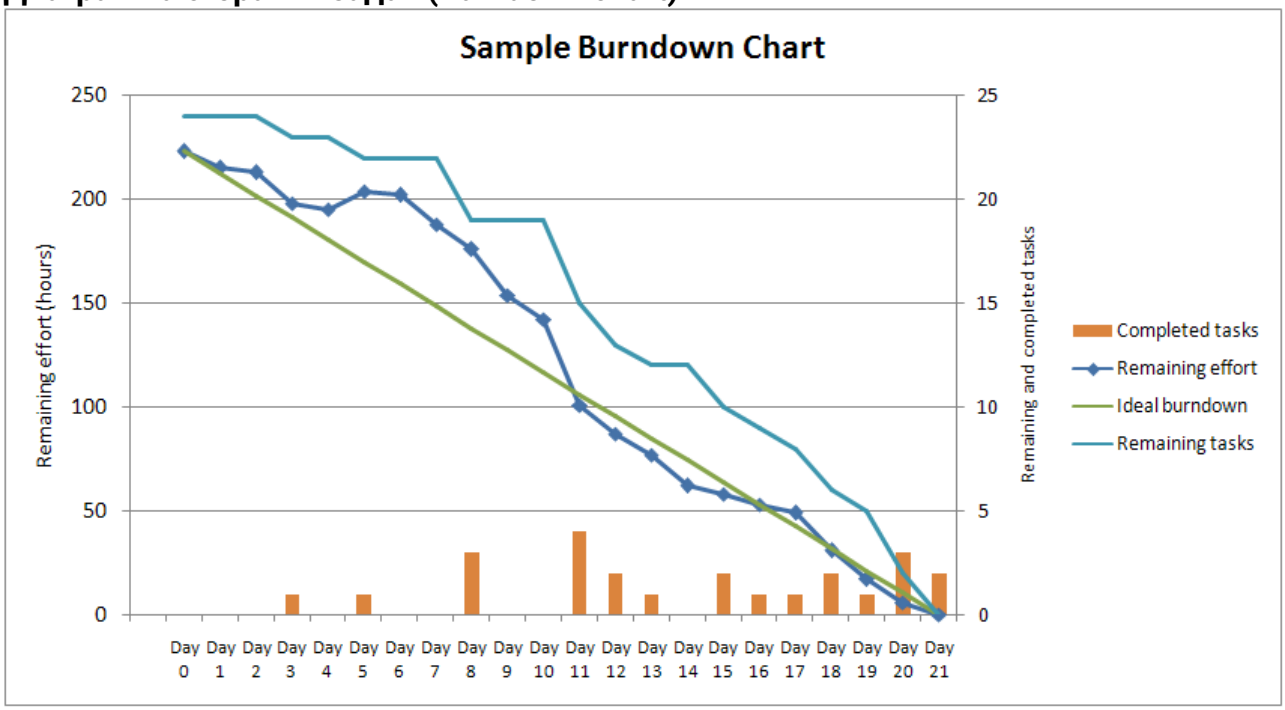
Скрам митинг проводит Скрам Мастер. Он по кругу задает вопросы каждому члену команды

• 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, показывающая 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 Ретроспективы пересмотреть качество существующих процессов, взаимоотношения людей и применяемые инструменты. Команда определяет, что прошло хорошо, а что 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» значит 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 Тойоты Короллы. У вас около 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 — система сама легко подстраивается под изменения.

Основная задача карт Канбан в этой системе — это уменьшать количество «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 Тойоте 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, буть 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:

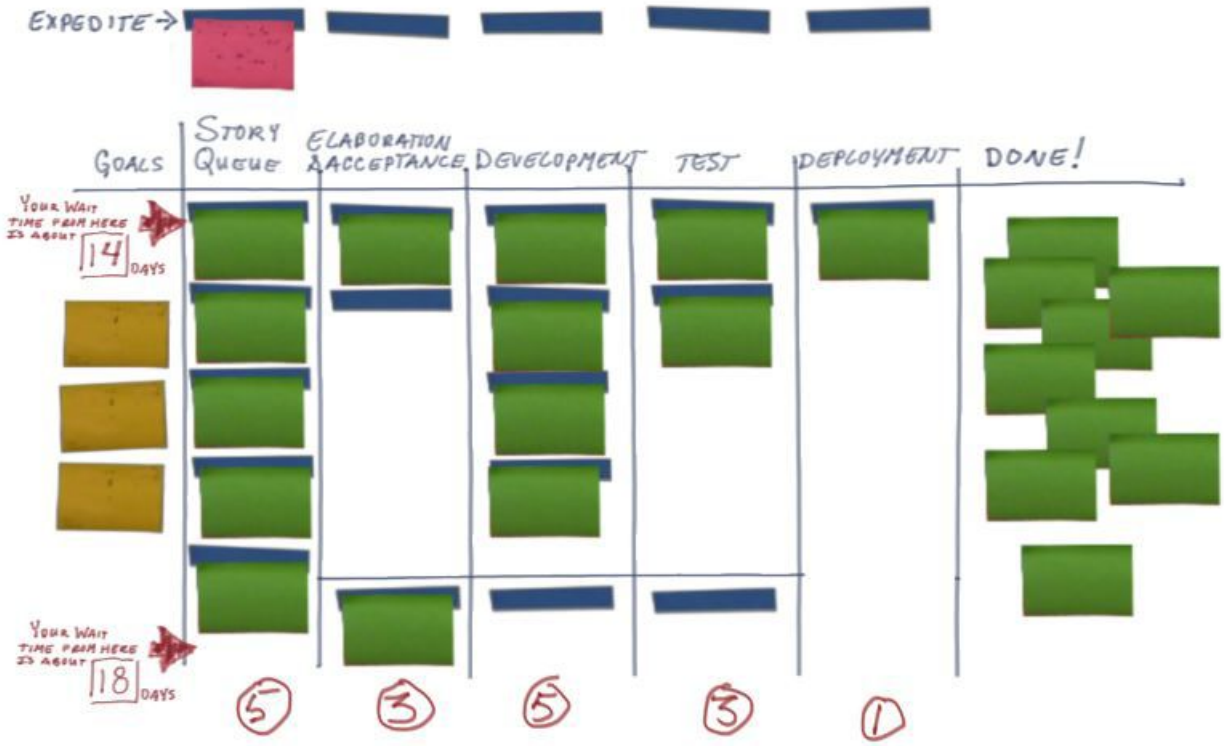
— В Канбан нет таймбоксов ни на что (ни на задачи, ни на спринты)
— В Канбан задачи больше и их меньше
— В Канбан оценки сроков на задачу опциональные или вообще их нет
— В Канбан «скорость работы команды» отсутствует и считается только среднее время на полную реализацию задачи

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 ошибочно вначале.

Если менеджер верит команде, то зачем иметь оценку времени? Задача менеджера — это создать приоритизированный 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 не точна — 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, что 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, а 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, какие 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» я понимаю не только программистов, но и других специалистов. Например, для столбца «Тестирование» разработчики — это тестеры, т.к. тестирование — это их обязаность.

Каскадная модель (waterfall)

— высокий уровень формализации процессов;

— большое количество документации;

— жесткая последовательность этапов жизненного цикла без возможности возврата на предыдущий этап.

Минусы

• Watrefall проект должен постоянно иметь актуальную документацию. Обязательная актуализация проектной документации. Избыточная документация

• Очень не гибкая методологии

• 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 выйдет 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 контроля качества

«Водоворот» или каскадная модель с промежуточным контролем

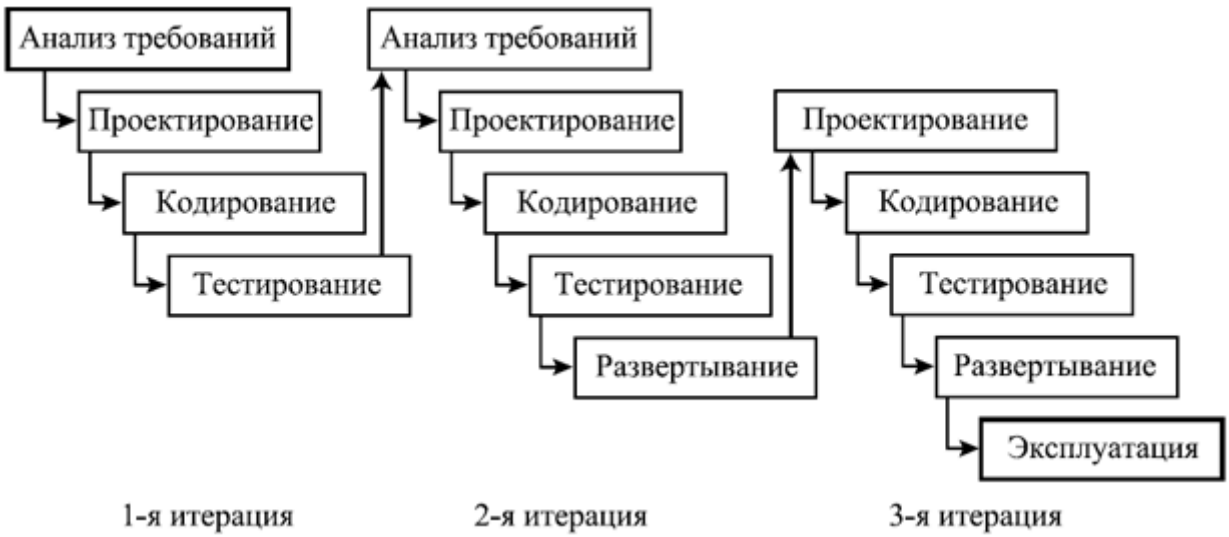
В этой модели предусмотрен промежуточный контроль за счет обратных связей. Но это достоинство порождает и недостатки. Затраты на реализацию проекта при таком подходе возрастают практически в 10 раз. Эта модель, как вы уже поняли,

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, изменения политик 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.



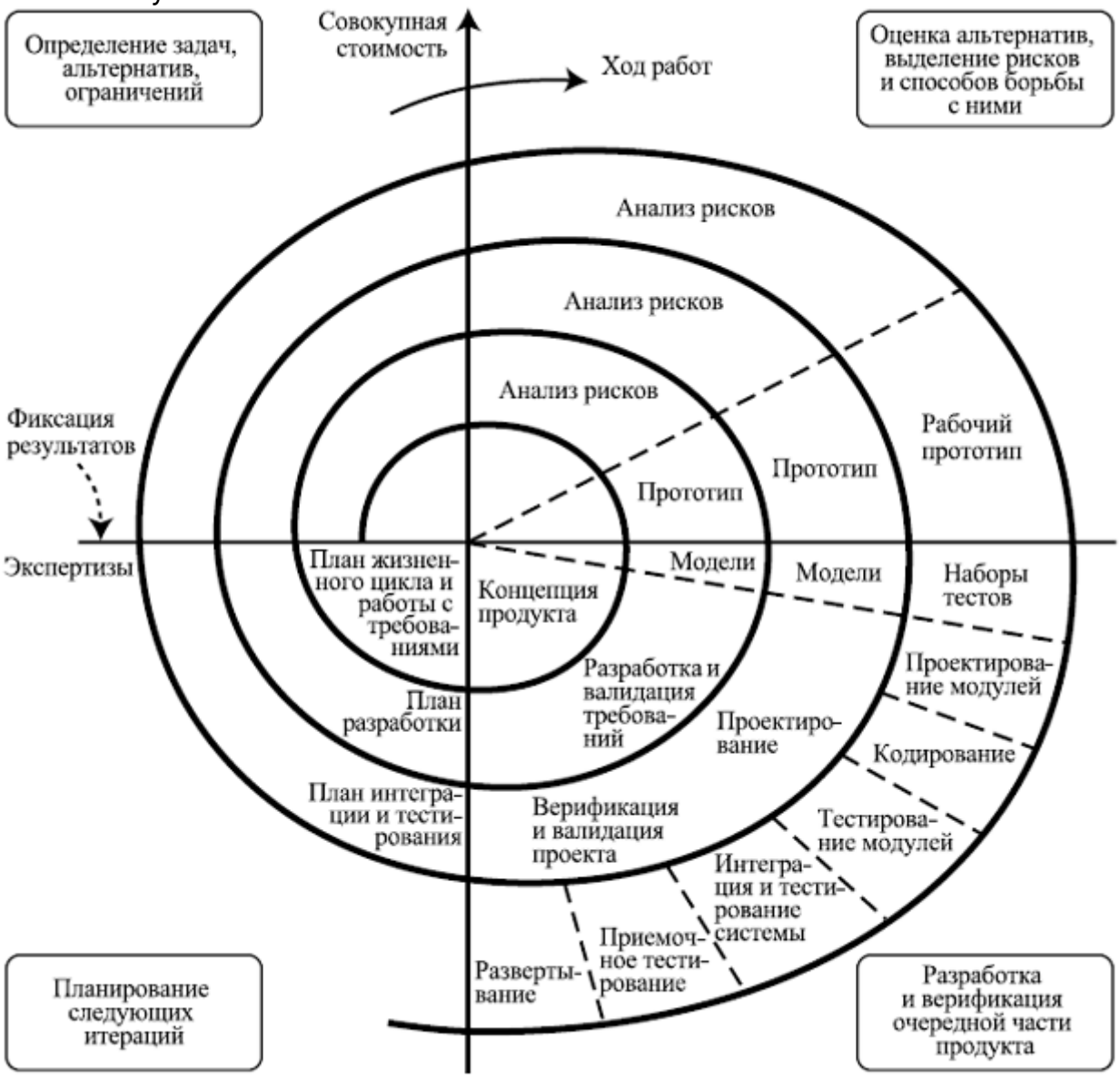
Каскадная модель с возможностью возвращения на предшествующий шаг при необходимости пересмотреть его результаты, становится итеративной. Итеративный процесс предполагает, что разные виды деятельности не привязаны намертво к определенным этапам разработки, а выполняются по мере

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 постадийное прототипирование и проектирование. 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 в проекте и риски 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:
o Прояснить не ясные требования (прототип UI)
o Выбрать одно из ряда концептуальных решений (реализация сцинариев)
o Проанализировать осуществимость проекта
Классификация протопипов:
o Горизонтальные прототипы — моделирует исключительно UI не затрагивая логику обработки и базу данных.
o Вертикальные прототипы — проверка архитектурных решений.
o Одноразовые прототипы — для быстрой разработки.
o Эволюционные прототипы — первое приближение эволюционной системы.

Модель Хаоса

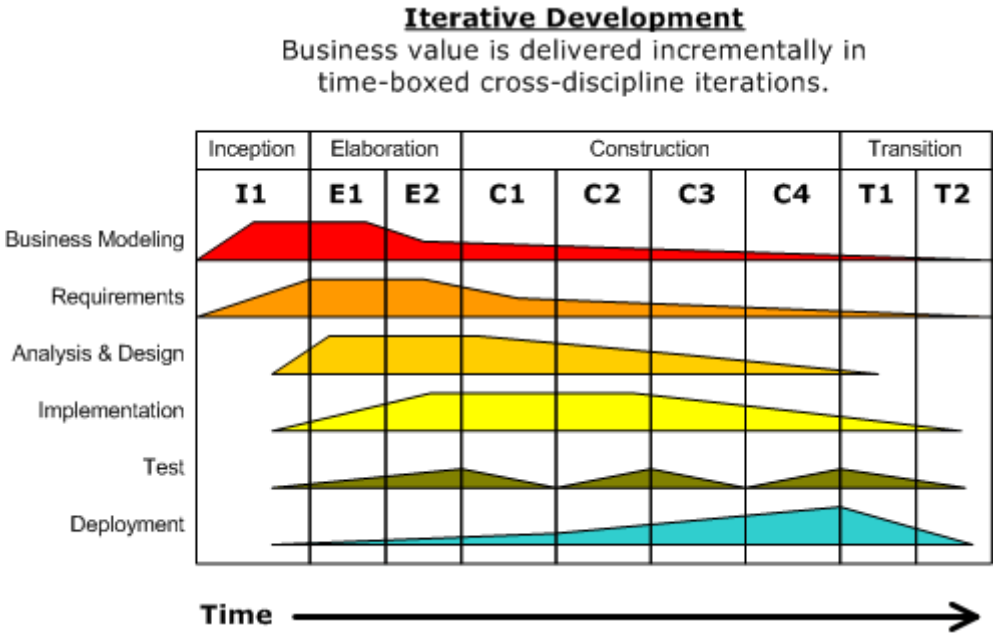
Вкратце Стратегия хаоса — это стратегия разработки программного обеспечения основанная на модели хаоса. Главное правило — это,всегда решать наиболее важную задачу первой.

Экстремальное программирование (XP)

Двенадцать основных приёмов экстремального программирования (по первому изданию книги Extreme programming explained) могут быть объединены в четыре группы:
• 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)
• Непрерывный, а не пакетный процесс
• Непрерывная интеграция (Continuous integration)
• Рефакторинг (Design improvement, Refactoring)
• Частые небольшие релизы (Small releases)
• 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)
• Социальная защищенность программиста (Programmer welfare):
• 40-часовая рабочая неделя (Sustainable pace, Forty-hour week)

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 по окончании каждого этапа и каждой итерации создавать все требуемые документы и достигнуть максимального уровня формализации, 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.

MICROSOFT SOLUTIONS FRAMEWORK

— методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

Базовые концепции и принципы модели процессов MSF:

• единое видение проекта — все заинтересованные лица и просто участники проекта должны чётко представлять конечный результат, всем должна быть понятна цель проекта;
• управление компромиссами — поиск компромиссов между ресурсами проекта, календарным графиком и реализуемыми возможностями;
• гибкость — готовность к изменяющимся проектным условиям;
• концентрация на бизнес-приоритетах — сосредоточенность на той отдаче и выгоде, которую ожидает получить потребитель решения;
• поощрение свободного общения внутри проекта;
• создание базовых версии — фиксация состояния любого проектного артефакта, в том числе программного кода, плана проекта, руководства пользователя, настройки серверов и последующее эффективное управление изменениями, аналитика проекта.

MSF предлагает проверенные методики для планирования, проектирования, разработки и внедрения успешных IT-решений. Благодаря своей гибкости, масштабируемости и отсутствию жестких инструкций MSF способен удовлетворить нужды организации или проектной группы любого размера. Методология MSF состоит из принципов, моделей и дисциплин по управлению персоналом, процессами, технологическими элементами и связанными со всеми этими факторами вопросами, характерными для большинства проектов.

Источники: www.web-pharus.ru, habrahabr.ru, youmanager.pro, fresh2l.com, ru.wikipedia.org, 2programmer.ru/tehnolog1?start=5, tim.com.ua.

Nice to read: www.scrumalliance.org/...ns/Core-Scrum-Russian.pdf