Рязанский станкостроительный колледж РГРТУ

МДК.01.01 Разработка программных модулей

Тема 1. Этапы разработки программного обеспечения

Рязань 2020

Оглавление

Этапы разработки программного обеспечения.

Основные понятия

Очень часто у начинающих программистов возникает вопрос, зачем нужно соблюдать какие-то там этапы, ведь разработка программы — это просто сел и написал код. Однако это не так, с таким подходом создать нормальное приложение не получится.

В зависимости от размера программных проектов этапы разработки могут отличаться, в некоторых случаях это будут очень детализированные и бюрократичные этапы, а в некоторых — просто сформулированные в любом удобном для разработчиков виде.

Так, например, при строительстве сарая у себя на даче Вы не будете что-то там детально планировать, исследовать, инспектировать, но в случае, скажем, со строительством электростанции все будет очень детально спланировано, спроектировано, режим работы рабочих будет расписан поминутно, так как цена ошибки на любом этапе будет значительно выше, чем в случае со строительством простого сарая.

Точно так же происходит и при разработке ПО, если проект крупный и очень важный, который возможно будет влиять на жизни людей или связан с огромными финансовыми рисками, все этапы разработки ПО будут соблюдаться, т.е. детально проработаны и даже будут добавляться новые этапы, микроэтапы и так далее.

Все это делается для того, чтобы не допустить появления ошибок и реализовать тот продукт, который действительно нужен.

Чем раньше будут обнаружены ошибки или выявлен неправильных подход в реализации того или иного действия, тем цена этих ошибок будет меньше. Иными словами, в зависимости от этапа обнаружения ошибки ее цена может меняться от 10 до 100 раз. Например, если на самом начальном этапе цена исправления ошибки будет равняться 100 рублей, то на этапе тестирования она может вылиться в 10000. Поэтому этапы разработки ПО очень важны, и разработчик должен их соблюдать и попытаться донести это видение до менеджеров, которым всегда нужен только результат. Так как они или отводят на это слишком мало времени или и вовсе не считают это необходимым, например, зачем при программировании вырабатывать какие-то требования или что-то там проектировать.

Основные этапы разработки ПО

Вот этапы, которые в большинстве случаев должны соблюдаться при разработке программного обеспечения:

Этап 1 – Постановка задачи, анализ требований к проекту
Этап 2 – Выработка требований
Этап 3 – Создание плана разработки
Этап 4 – Разработка архитектуры системы или высокоуровневое проектирование
Этап 5 – Детальное проектирование
Этап 6 – Кодирование и отладка
Этап 7 – Тестирование компонентов
Этап 8 – Интеграция компонентов
Этап 9 – Тестирование всей системы
Этап 10 – Сопровождение, внесение изменений, оптимизация

Некоторым может показаться, что это слишком сложный план, но если Вы будете работать над крупным проектом, то столкнётесь со всем этим, и даже более детализированным планом.

Сейчас давайте рассмотрим каждый этап, т.е. узнаем, какие действия необходимо выполнять на каждом этапе.

Этап 1 – Постановка задачи, анализ требований к проекту

Перед тем как приступать к кодированию, необходимо четко сформулировать задачу, которую Ваша будущая программа должна решать. Так как, не имея хорошего определения задачи, Вы можете потратить много усилий и времени на решение не той задачи, которую требуется решить.

На данном этапе проводится простая формулировка сути задачи без каких-либо намеков на ее возможные решения, при этом формулировать ее следует на языке, понятном пользователю, т.е. она должна быть описана с пользовательской точки зрения.

Выделяются базовые сущности и взаимосвязи между ними. Выявляются аналоги, обеспечивающих достижение подобных целей, их достоинства и недостатки. Определяются структуры входных и выходных данных. Предварительно выбираются методы решения задач.

Этап 2 – Выработка требований

Что такое требования и зачем их нужно выработать?

Требования к программе – это подробное описание всех возможностей программы и действий, которые должна выполнять программа. Такие требования иногда также называют ***«Функциональной спецификацией»*** или просто ***«Спецификацией»***.

Требования вырабатывают для того, чтобы свести к минимуму изменения системы после начала непосредственной разработки. Такие требования должны быть обязательно официальными, т.е. документально оформлены. Так как это гарантирует то, что функциональность системы определяется заказчиком, а не программистом. Даже в случае с внутрикорпоративными разработками такие требования должны быть зафиксированы, например, в виде ***технического задания***, подписанного всеми задействованными лицами, тем самым Вы избежите лишних разговоров и споров, например, о том, что реализованный функционал делает не все или не так.

Выработка требований очень важна, так как она позволяет определить функциональность программы до начала программирования.

Этап 3 – Создание плана разработки

На данном этапе Вы уже должны в формальном виде составить план разработки программного обеспечения с учётом существующей проблемы и выработанных требований. Иными словами, Вы должны составить план того, как Вы будете действовать дальше.

Этап 4 – Разработка архитектуры системы или высокоуровневое проектирование

Архитектура системы – это каркас программы, это высокоуровневое проектирование программы.

Данный этап также очень важный, так как, не имея хорошей архитектуры, Вы можете решать правильную проблему, но прийти к неправильному решению. Хорошая архитектура программы упрощает программирование, а плохая архитектура усложняет его.

Архитектура системы обычно включает:

- Общее описание системы;
- Основные компоненты;
- Формат и способ хранения данных;
- Специфические бизнес-правила;
- Способ организации пользовательского интерфейса;
- Подход к безопасности системы;
- Оценки производительности;
- Возможности масштабирования;

- Моменты, связанные с интернациональностью, т.е. будет ли система интернациональной.

Кроме того, в архитектуру необходимо включить подтверждение того, что при разработке этой архитектуры рассматривались альтернативные варианты в каждом из вышеперечисленных направлений, с обоснованием окончательного выбора и подхода.

Этап 5 – Детальное проектирование

На этом этапе проводится проектирование программы на низком уровне, иными словами, здесь проектируются классы и методы, рассматриваются, оцениваются и сравниваются различные варианты и причины выбора окончательных подходов и способов реализации.

При разработке небольших программ программисты обычно сами проектируют программу на таком уровне, это выглядит как написание псевдокода или рисование схем, поэтому часто этот этап рассматривается как часть непосредственного кодирования и в таких случаях итоговый документ (*если того требует формальность*) состоит преимущественно из различных набросков и заметок программистов.

Но при реализации крупных проектов данному процессу отводится отдельный этап и проектирование в этом случае проводится с очень высокой степенью детальности.

Этап 6 – Кодирование и отладка

Это как раз тот этап, который все знают и, наверное, думают, что это единственный этап в процессе разработке программного обеспечения – это непосредственное написание кода и его отладка. Но, как видите, это далеко не первый и не единственный этап разработки ПО.

Если все вышеперечисленные этапы выполнены, то данный этап подразумевает чисто механическую работу, т.е. кодинг. Программисту в этом случае не нужно что-то выдумывать и самостоятельно разрабатывать, ему нужно просто написать код, который реализует заданный, очень детально описанный в проекте, алгоритм.

После того как код написан, программисту необходимо отладить этот код, чтобы в нем не было никаких ошибок.

Этап 7 – Тестирование компонентов

После того, как код написан, и проведена отладка, необходимо провести тестирование реализованного функционала. Если программа состоит из нескольких компонентов, сначала тестируют каждый компонент в отдельности, так как очень крупные программы включают огромный функционал, который часто разделяют на отдельные компоненты, разработка которых осуществляется по отдельности. В менее крупных проектах этот этап может включать просто тестирование отдельных классов.

Этап 8 – Интеграция компонентов

Когда тестирование всех компонентов закончено, можно переходить к интеграции всех компонентов в единый программный комплекс, этот этап как раз и подразумевает процесс интеграции, т.е. слияния всех компонентов в единую систему.

В небольших проектах этот этап может заключаться в объединении нескольких классов, на что будет затрачено не больше одного дня, но в крупных проектах этот этап может длиться не один месяц.

Этап 9 – Тестирование всей системы

На данном этапе проводится тестирование всей системы, уже с учётом интеграции всех компонентов. На этом этапе можно выявить проблемы взаимодействия компонентов и устранить их. Также на этом этапе основным предметом тестирования является

безопасность, производительность, утечка ресурсов и другие моменты, которые невозможно протестировать на более низких уровнях тестирования.

Этап 10 – Сопровождение, внесение изменений, оптимизация

После запуска программы в промышленную эксплуатацию осуществляется сопровождение этой программы, т.е. внесение изменений на основе выявленных недочетов в процессе эксплуатации системы, также проводится оптимизация функционала или добавление нового, обучение пользователей и т.д.

Жизненный цикл ПО

Понятие жизненного цикла ПО

Соответственно разработкой программного обеспечения тесно связано понятие жизненного цикла программного обеспечения, который определяется как период времени, который начинается с момента принятия решения о необходимости создания ПО и закачивается в момент ее полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим состав процессов ЖЦ ПО, является международный стандарт ISO/IEC 12207: 1995 "Information Technology – Software Life Cycle Process" (ISO – International Organization for Standardization – Международная организация по стандартизации, IEC – International Electrotechnical Commission – Международная комиссия по электротехнике). Этот стандарт определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

В данном стандарте ПО (или программный продукт) определяется как набор компьютерных программ, процедур и, возможно, связанной с ними документацией и данных.

В соответствии со стандартом ISO/IEC 12207 все процессы ЖЦ ПО разделены на три группы (рис.1).

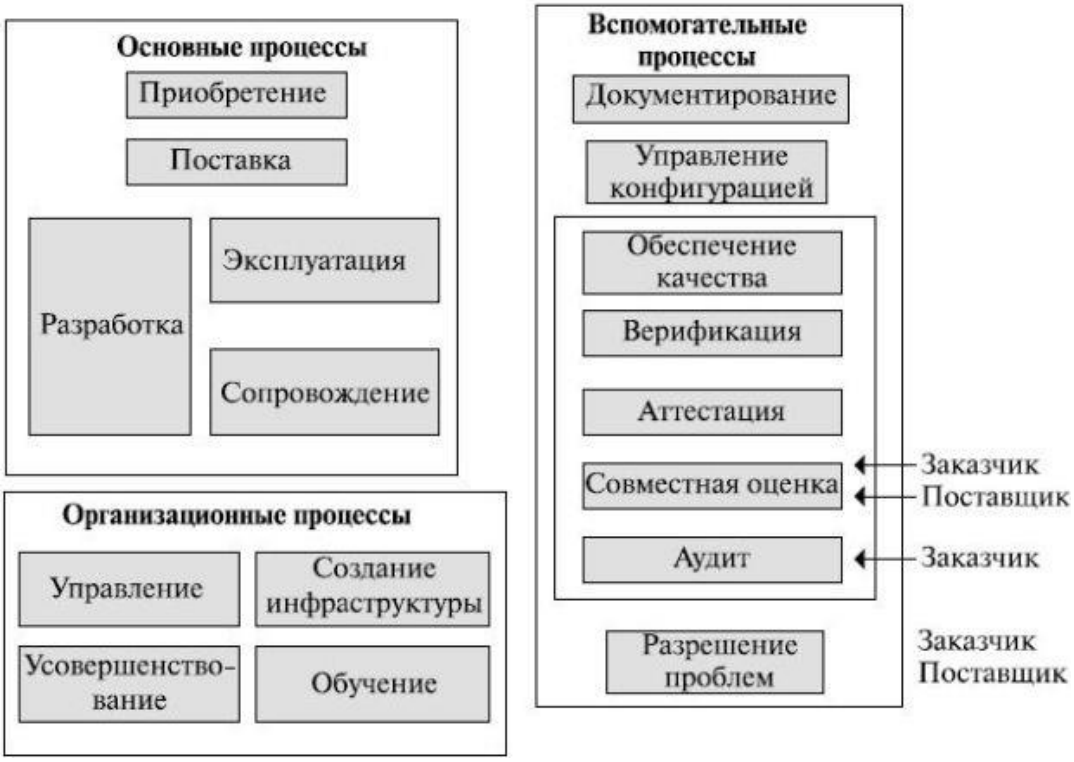


Рисунок 1 - Процессы жизненного цикла программного обеспечения.

В группах определено пять основных процессов: приобретение, поставка, разработка, эксплуатация и сопровождение. Восемь вспомогательных процессов обеспечивают выполнение основных процессов, а именно документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем. Четыре организационных процесса обеспечивают управление, создание инфраструктуры, усовершенствование и обучение.

Основные процессы ЖЦ ПС

Приобретение

Процесс приобретения состоит из действий и задач заказчика, приобретающего ПС. Данный процесс охватывает следующие действия:

1. инициирование приобретения;
2. подготовку заявочных предложений;
3. подготовку и корректировку договора;
4. надзор за деятельностью поставщика;
5. приемку и завершение работ.

Поставка

Процесс поставки охватывает действия и задачи, выполняемые поставщиком, который снабжает заказчика программным продуктом или услугой. Данный процесс включает следующие действия:

1. инициирование поставки;
2. подготовку ответа на заявочные предложения;
3. подготовку договора;
4. планирование работ по договору;
5. выполнение и контроль договорных работ и их оценку;
6. поставку и завершение работ.

Разработка

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком, и охватывает работы по созданию ПО и его компонентов в соответствии с заданными требованиями. Сюда включается оформление проектной и эксплуатационной документации, подготовка материалов, необходимых для проверки работоспособности, и качества программных продуктов, материалов, необходимых для организации обучения персонала и др.

Процесс разработки включает следующие действия:

1. подготовительную работу;
2. анализ требований, предъявляемых к системе;
3. проектирование архитектуры системы;
4. анализ требований, предъявляемых к программному обеспечению;
5. проектирование архитектуры программного обеспечения;
6. детальное проектирование программного обеспечения;
7. кодирование и тестирование программного обеспечения;
8. интеграцию программного обеспечения;
9. квалификационное тестирование программного обеспечения;
10. интеграцию системы;
11. квалификационное тестирование системы;
12. установку программного обеспечения;
13. приемку программного обеспечения.

Эксплуатация

Процесс эксплуатации охватывает действия и задачи организации оператора, эксплуатирующего систему. Процесс эксплуатации включает следующие действия.

1. Подготовительная работа, которая включает проведение оператором следующих задач:
   - планирование действий и работ, выполняемых в процессе эксплуатации, и установка эксплуатационных стандартов;

- определение процедур локализации и разрешения проблем, возникающих в процессе эксплуатации.

2. Эксплуатационное тестирование, осуществляемое для каждой очередной редакции программного продукта, после чего эта редакция передается в эксплуатацию.
3. Собственно эксплуатация системы, которая выполняется в предназначенной для этого среде в соответствии с пользовательской документацией.
4. Поддержка пользователей – оказание помощи и консультаций при обнаружении ошибок в процессе эксплуатации ПО.

Сопровождение

Процесс сопровождения представляет собой действия и задачи, которые выполняются сопровождающей организацией, при изменениях (модификациях) программного продукта и соответствующей документации, вызванных возникшими проблемами или потребностями в модернизации или адаптации ПО.

Изменения, вносимые в соответствующее ПО, не должны нарушать его целостность. Процесс сопровождения включает его перенос в другую среду (миграцию) и заканчивается снятием ПО с эксплуатации.

Процесс сопровождения охватывает следующие действия:

1. подготовительную работу (планирование действий и работ, определение процедур локализации и разрешения проблем, возникающих в процессе сопровождения);
2. анализ проблем и запросов на модификацию ПО (анализ сообщений о возникшей проблеме или запроса на модификацию, оценка масштаба, стоимости модификации, получаемого эффекта, оценка целесообразности модификации);
3. модификацию ПО (внесение изменений в компоненты программного продукта и документацию в соответствии с правилами процесса разработки);
4. проверку и приемку (в части целостности модифицируемой системы);
5. перенос ПО в другую среду (конвертирование программ и данных, параллельная эксплуатация ПО в старой и новой среде в течение некоторого периода времени);
6. снятие ПО с эксплуатации по решению заказчика при участии эксплуатирующей организации, службы сопровождения и пользователей. При этом программные продукты и документации подлежат архивированию в соответствии с договором.

Вспомогательные процессы ЖЦ ПО

Документирование

Процесс документирования. Предусматривает формализованное описание информации, созданной в течение ЖЦ ПО. Данный процесс состоит из набора действий, с помощью которых планируют, проектируют, разрабатывают, выпускают, редактируют, распространяют и сопровождают документы, необходимые для всех заинтересованных лиц, таких как руководство, технические специалисты и пользователи системы.

Процесс документирования включает следующие действия:

1. подготовительную работу;
2. проектирование и разработку;
3. выпуск документации;
4. сопровождение.

Управление конфигурацией

Процесс управления конфигурацией включает административные и технические процедуры на всем протяжении ЖЦ ПО для определения состояния компонентов ПО, описания и подготовки отчетов о состоянии компонентов ПО и запросов на модификацию, обеспечения полноты, совместимости и корректности компонентов ПО, управления хранением и поставкой ПО.

Согласно стандарту IEEE-90 под конфигурацией ПО понимается совокупность его функциональных и физических характеристик, установленных в технической документации и реализованных в ПО. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации по управлению конфигурацией ПО отражены в стандарте ISO/IEC 15288 "Information Technology. Software Life Cycle Process. Configuration Management for Software".

Процесс управления конфигурацией включает следующие действия:

1. подготовительную работу, заключающуюся в планировании управления конфигурацией;
2. идентификацию конфигурации, устанавливающую правила, с помощью которых однозначно идентифицируются компоненты ПО и их версии. При этом каждому компоненту однозначно соответствует комплект документации;
3. контроль конфигурации – действие, предназначенное для систематической оценки предлагаемых модификаций ПО и координированной их реализации с учетом эффективности каждой модификации и затрат на ее выполнение;
4. учет состояния конфигурации, представляющий собой регистрацию состояния компонентов ПО. Обеспечивает подготовку отчетов о реализованных и отвергнутых модификациях версий компонентов ПО. Совокупность отчетов дает однозначное отражение текущего состояния системы и ее компонентов, а также обеспечивает ведение истории модификаций;
5. оценку конфигурации, заключающуюся в определении функциональной полноты компонентов ПО, а также соответствия их физического состояния текущему техническому описанию;
6. управление выпуском и поставку, охватывающие изготовление эталонных копий программ и документации, их хранение и поставку пользователям в соответствии с порядком, принятом в организации.

Обеспечение качества

Процесс обеспечения качества должен обеспечивать гарантии того, что ПО и процессы его ЖЦ соответствуют заданным требованиям и утвержденным планам. Под качеством ПО понимается совокупность свойств, которая характеризует способность ПО удовлетворять заданным требованиям. Для получения достоверных оценок о создаваемом ПО процесс обеспечения его качества должен происходить независимо от субъектов, непосредственно связанных с разработкой программного продукта. При этом могут использоваться результаты других вспомогательных процессов, таких как верификация, аттестация, совместная оценка, аудит и разрешение проблем.

Процесс обеспечения качества включает следующие действия:

1. подготовительную работу (координацию с другими вспомогательными процессами и планирование самого процесса обеспечения качества ПО с учетом используемых стандартов, методов, процедур и средств);
2. обеспечение качества продукта, подразумевающего гарантированное полное соответствие ПО и его документации требованиям заказчика, предусмотренным в договоре;

3. обеспечение качества процесса, предполагающее гарантированное соответствие процессов ЖЦ ПО, методов разработки, среды разработки и квалификации персонала условиям договора, установленным стандартам и процедурам;
4. обеспечение прочих показателей качества ПО, осуществляемое в соответствии с условиями договора и стандартом качества ISO 9001.

Верификация

Процесс верификации состоит в определении того факта, что ПО, являющееся результатом некоторой деятельности, полностью удовлетворяет требованиям или условиям, обусловленным предшествующими действиями. Для повышения эффективности всего процесса ЖЦ ПО верификация должна как можно раньше интегрироваться с использующими ее процессами (т.е. с поставкой, разработкой, эксплуатацией). Процесс верификации может включать анализ, оценку и тестирование.

Верификация может проводиться с различными степенями независимости (от самого исполнителя до специалистов другой организации, не зависящей от поставщика, разработчика и т.д.).

В процессе верификации проверяются следующие условия:

1. непротиворечивость требований, предъявляемых к системе, и степень учета потребностей пользователей;
2. возможность поставщика выполнить заданные требования;
3. соответствие выбранных процессов ЖЦ ПО условиям договора;
4. адекватность стандартов, процедур и среды разработки процессам ЖЦ ПО;
5. соответствие проектных спецификаций ПО заданным требованиям;
6. корректность описания в проектных спецификациях входных и выходных данных, последовательности событий, интерфейсов, логики и т.д.;
7. соответствие кода проектным спецификациям и требованиям;
8. тестируемость и корректность кода, его соответствие принятым стандартам кодирования;
9. корректность интеграции компонентов ПО в систему;
10. адекватность, полнота и непротиворечивость документации.

Аттестация

Процесс аттестации предназначен для определения полноты соответствия заданных требований и созданного ПО их конкретному функциональному назначению (тому, что требуется потребителю). Под аттестацией обычно понимается подтверждение и оценка достоверности проведенного тестирования программного продукта. Аттестация должна гарантировать полное соответствие ПО спецификациям, требованиям и документации, а также возможность безопасного и надежного применения ПО пользователем.

Аттестация, как и верификация, может осуществляться с различными степенями независимости (вплоть до организации, не зависящей от поставщика, разработчика, оператора или службы сопровождения).

Процесс совместной оценки предназначен для оценки состояния работ по проекту и программному продукту, создаваемому при выполнении этих работ. Он сосредоточен в основном на контроле планирования и управления ресурсами, персоналом, аппаратурой и инструментальными средствами проекта.

Оценка применяется как на уровне управления проектом, так и на уровне технической реализации проекта и проводится в течение всего срока действия договора. Данный процесс может выполняться двумя сторонами, участвующими в договоре, при этом одна сторона проверяет другую.

Аудит

Процесс аудита представляет собой определение соответствия проекта и продукта требованиям, планам и условиям договора. Аудит может выполняться двумя любыми сторонами, участвующими в договоре, когда одна сторона проверяет другую.

Аудит – это ревизия (проверка), проводимая компетентным органом (лицом) в целях обеспечения независимой оценки степени соответствия ПО или процессов установленным требованиям.

Аудит служит для установления соответствия реальных работ и отчетов требованиям, планам и контракту. Аудиторы не должны иметь прямой зависимости от разработчиков ПО. Они определяют состояние работ, использование ресурсов, соответствие документации спецификациям и стандартам, корректность тестирования и др.

Процесс разрешения проблем предусматривает анализ и разрешение проблем (включая обнаруженные несоответствия), которые обнаружены в ходе разработки, эксплуатации или других процессов независимо от их происхождения или источника.

Организационные процессы ЖЦ ПО

Управление

Процесс управления состоит из действий и задач, которые могут выполняться любой стороной, управляющей своими процессами. Данная сторона (менеджер) отвечает за управление выпуском продукта, управление проектом и управление задачами соответствующих процессов, таких как приобретение, поставка, разработка, эксплуатация, сопровождение и др.

Процесс управления включает следующие действия:

1. инициирование и определение области управления – менеджер должен убедиться, что необходимые для управления ресурсы (персонал, оборудование и технология) имеются в его распоряжении в достаточном количестве;
2. планирование, как действие, подразумевает выполнение следующих задач:
   - составление графиков выполнения работ;
   - оценку затрат;
   - выделение требуемых ресурсов;
   - распределение ответственности;
   - оценку рисков, связанных с конкретными задачами;
   - создание инфраструктуры управления.

Создание инфраструктуры

Процесс создания инфраструктуры охватывает выбор и поддержку технологий, стандартов и инструментальных средств, используемых для разработки, эксплуатации или сопровождения ПО. Инфраструктура должна модифицироваться и сопровождаться в соответствии с изменениями требований к соответствующим процессам. Инфраструктура, в свою очередь, является одним из объектов управления конфигурацией.

Процесс создания инфраструктуры включает следующие действия:

- подготовительную работу;
- создание инфраструктуры;
- сопровождение инфраструктуры.

Усовершенствование

Процесс усовершенствования предусматривает оценку, измерение, контроль и собственно усовершенствование процессов ЖЦ ПО. Этот процесс включает три основных действия:

- создание процесса;

- оценку процесса;
- усовершенствование процесса.

Усовершенствование процессов ЖЦ ПО направлено на повышение производительности труда всех участвующих в них специалистов за счет совершенствования используемой технологии, методов управления, выбора инструментальных средств и обучения персонала. Усовершенствование основано на анализе достоинств и недостатков каждого процесса. Такому анализу способствует накопление в организации исторической, технической, экономической и иной информации по реализованным проектам.

Обучение

Процесс обучения включает первоначальное обучение и последующее постоянное повышение квалификации персонала и состоит из трех действий:

- подготовительной работы;
- работки учебных материалов;
- реализации планов обучения.

Модели жизненного цикла ПО

Code-and-Fix

Исторически, первой моделью разработки программного обеспечения была Code-and-Fix. Используя Code-and-Fix, мы только пишем код.

Описать правила этого метода просто:

- получаем начальное понимание потребностей заказчика;
- начинаем программировать;
- когда что-то будет готово, показываем «это» заказчику;
- получив отзывы, исправляем наш код;
- повторяем цикл до полного удовлетворения заказчика (или пока у него не кончатся деньги, терпение…)

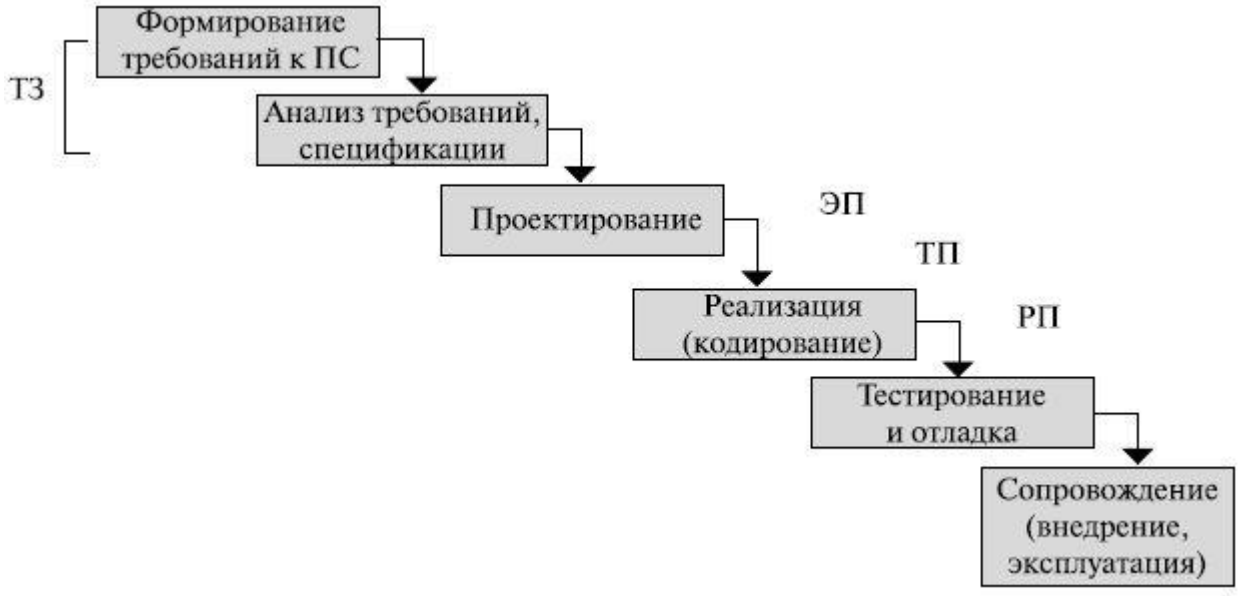
Используя Code-and-Fix, мы только пишем код. Здесь все очень просто: нет необходимости что-либо планировать, нет необходимости что-либо документировать. Поэтому Code-and-Fix требует минимальной квалификации разработчиков, соответственно, им можно платить меньше денег.

Но, у всего есть своя цена. Как показал опыт, такой подход очень скоро приводит к коду, который невозможно поддерживать: исправление одной ошибки приводит к появлению нескольких новых; внесение минимальных изменений в одну из частей программы, приводит к разрушению функций, реализуемых другими частями.

Все последующее развитие методик разработки выросло из стремления решить эти проблемы.

Каскадная модель (классический жизненный цикл)

Эта модель обязана своим появлением У. Ройсу ([1], 1970 г.). Модель имеет и другое название – водопад (waterfall). Особенность модели – переход на следующую ступень осуществляется только после того, как будет полностью завершена работа на предыдущей стадии; возвратов на пройденные стадии не предусматривается.



Требования к разрабатываемой ПО, определенные на стадиях формирования и анализа, строго документируются в виде ТЗ и фиксируются на все время разработки

проекта. Каждая стадия завершается выпуском полного комплекта документации (ТЗ, ЭП, ТП, РП), достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков. Критерием качества разработки при таком подходе является точность выполнения спецификаций ТЗ. Основное внимание разработчиков сосредоточивается на достижении оптимальных значений технических характеристик разрабатываемой ПС – производительности, объема занимаемой памяти и др.

Преимущества каскадной модели:

- на каждой стадии формируется законченный набор проектной документации, отвечающей критериям полноты и согласованности;
- выполняемые в логической последовательности стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

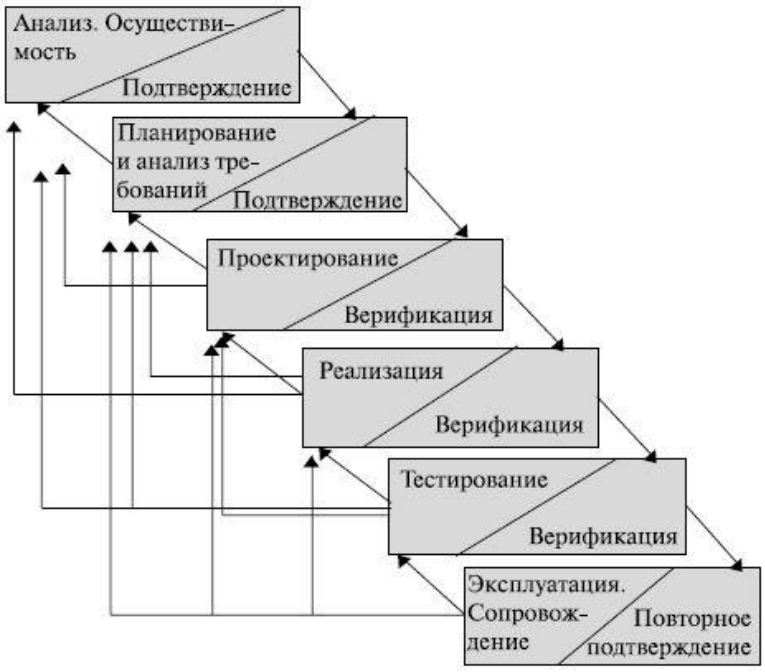
Каскадный подход хорошо зарекомендовал себя при построении ПО, для которых в самом начале проекта можно полно и четко сформулировать все требования. Пока все это контролируется стандартами и различными комиссиями госприемки, схема работает хорошо.

Недостатки каскадной модели:

- выявление и устранение ошибок производится только на стадии тестирования, которое может существенно растянуться;
- реальные проекты часто требуют отклонения от стандартной последовательности шагов;
- цикл основан на точной формулировке исходных требований к ПС, реально в начале проекта требования заказчика определены лишь частично;
- результаты работ доступны заказчику только по завершении проекта.

Итерационная модель ЖЦ ПО

С ростом коммерческих проектов выяснилось, что не всегда удается детально проработать проект будущей системы, поскольку многие аспекты ее функционирования в динамических сферах деятельности (бизнес) меняются, пока система создается. Потребовалось изменить процесс разработки так, чтобы гарантировать внесение необходимых исправлений после завершения какого-либо этапа разработки. Так появилась итерационная модель ЖЦ ПО, называемая моделью с промежуточным контролем или моделью с циклическим повторением фаз.



В итерационной модели недостатки проектирования и программирования могут быть устранены позже путем частичного возврата на предыдущую стадию. Чем ниже уровень обнаружения ошибки, тем дороже ее исправление. Если стоимость усилий, необходимых для обнаружения и устранения ошибок на стадии написания кода, принять за единицу, то стоимость выявления и устранения ошибки на стадии выработки требований будет в 5-10 раз меньше, а стоимость выявления и устранения ошибки на стадии сопровождения – в 20 раз больше.

В такой ситуации огромное значение приобретает этап формулирования требований, составление спецификаций и создание плана системы. Программные архитекторы несут личную ответственность за все последующие изменения проектных решений. Объем документации исчисляется тысячами страниц, число утверждающих заседаний огромно. Многие проекты так никогда и не покидают этап планирования, впав в "паралич анализа". Одним из возможных путей исключения подобных ситуаций является макетирование (прототипирование).

Инкрементная модель

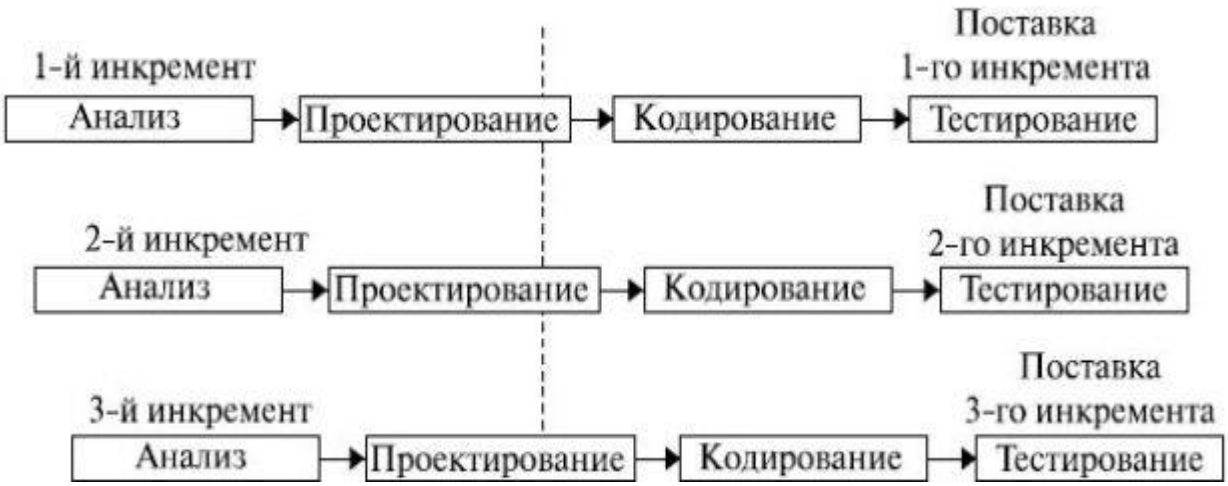
Инкрементная модель является классическим примером инкрементной стратегии конструирования. Она объединяет элементы последовательной водопадной модели с итерационной философией макетирования (предложена Б. Боэмом как усовершенствование каскадной модели). Каждая линейная последовательность здесь вырабатывает поставляемый инкремент ПО.

Например, ПО для обработки слов в 1-м инкременте (версии) реализует функции базовой обработки файлов, функции редактирования и документирования; во 2-м инкременте – более сложные возможности редактирования и документирования; в 3-м инкременте – проверку орфографии и грамматики; в 4-м инкременте – возможности компоновки страницы.

Первый инкремент приводит к получению базового продукта, реализующего базовые требования (правда, многие вспомогательные требования остаются нереализованными). План следующего инкремента предусматривает модификацию базового продукта, обеспечивающую дополнительные характеристики и функциональность.

По своей природе инкрементный процесс итеративен, но, в отличие от макетирования, инкрементная модель обеспечивает на каждом инкременте работающий продукт.

Одной из современных реализаций инкрементного подхода является экстремальное программирование (ориентировано на очень малые приращения функциональности).



Спиральная модель ЖЦ ПО

Спиральная модель – классический пример применения эволюционной стратегии конструирования. Модель (автор Б. Боэм, 1988) базируется на лучших свойствах классического жизненного цикла и макетирования, к которым добавляется новый элемент – анализ риска, отсутствующий в этих парадигмах. Модель определяет четыре действия, представляемые четырьмя квадрантами спирали.

Спиральная модель жизненного цикла программного обеспечения

1. Планирование – определение целей, вариантов и ограничений.
2. Анализ риска – анализ вариантов и распознавание/выбор риска.
3. Конструирование – разработка продукта следующего уровня.
4. Оценивание – оценка заказчиком текущих результатов конструирования.

Интегрирующий аспект спиральной модели очевиден при учете радиального измерения спирали. С каждой итерацией по спирали строятся все более полные версии ПС. В первом витке спирали определяются начальные цели, варианты и ограничения, распознается и анализируется риск. Если анализ риска показывает неопределенность

требований, на помощь разработчику и заказчику приходит макетирование, используемое в квадранте конструирования.

Для дальнейшего определения проблемных и уточненных требований может быть использовано моделирование. Заказчик оценивает инженерную (конструкторскую) работу и вносит предложения по модификации (квадрант оценки заказчиком). Следующая фаза планирования и анализа риска базируется на предложениях заказчика. В каждом цикле по спирали результаты анализа риска формируются в виде "продолжать, не продолжать". Если риск слишком велик, проект может быть остановлен.

В большинстве случаев движение по спирали продолжается, с каждым шагом продвигая разработчиков к более общей модели системы. В каждом цикле по спирали требуется конструирование (нижний правый квадрант), которое может быть реализовано классическим жизненным циклом или макетированием. Заметим, что количество действий по разработке (происходящих в правом нижнем квадранте) возрастает по мере продвижения от центра спирали.

Эти действия пронумерованы и имеют следующее содержание:

1. начальный сбор требований и планирование проекта;
2. та же работа, но на основе рекомендаций заказчика;
3. анализ риска на основе начальных требований;
4. анализ риска на основе реакции заказчика;
5. переход к комплексной системе;
6. начальный макет системы;
7. следующий уровень макета;
8. сконструированная система;
9. – оценивание заказчиком.

Модель спирального процесса разработки является наиболее распространенной в настоящее время. Самыми известными ее вариантами являются RUP (Rational Unified Process) от фирмы Rational и MSF (Microsoft Solution Framework). В качестве языка моделирования используется язык UML (Unified Modeling Language). Создание системы предполагается проводить итерационно, двигаясь по спирали и, проходя через одни и те же стадии, на каждом витке уточнять характеристики будущего продукта. Казалось бы, теперь все хорошо: и планируется только то, что можно предвидеть, разрабатывается то, что запланировано, и пользователи начинают знакомиться с продуктом заранее, имея возможность внести необходимые коррективы.

Однако для этого нужны очень большие средства. Действительно, если раньше можно было создавать и распускать группы специалистов по мере необходимости, то теперь все они должны постоянно участвовать в проекте: архитекторы, программисты, тестировщики, инструкторы и т. д. Более того, усилия различных групп должны быть синхронизированы, чтобы своевременно отражать проектные решения и вносить необходимые изменения.

Методики разработки программного обеспечения

MSF (Microsoft Solution Framework)

В 1994 году компания Microsoft выпустила пакет руководств MSF (Microsoft Solutions Framework) по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий. Целью было достижение максимальной отдачи от IT-проектов компании.

Текущая версия – MSF 4.0 была представлена в 2005 году. В этой версии произошло разделение методологии на два направления: MSF for Agile Software Development и MSF for CMMI Process Improvement.

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности, т. е. с каждой новой версией эволюционирует функциональность решения. Для малых проектов может быть достаточным выпуск одной версии. Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. Документация должна изменяться по мере реализации проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов для каждой стадии разработки продукта, которые могут быть использованы для планирования и контроля процесса разработки. Решение не представляет ценности, пока оно не внедрено. Именно по этой причине модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение, вплоть до момента, когда решение начинает давать отдачу.

RUP (Rational Unified Process)

Один из самых известных процессов, использующих итеративную модель разработки – RUP. Он был создан во второй половине 1990-х годов в компании Rational Software. Термином RUP обозначает как методологию, так и продукт компании IBM (ранее Rational) для управления процессом разработки. Методология RUP описывает абстрактный общий процесс, на основе которого организация или проектная команда должна создать специализированный процесс, ориентированный на ее потребности.

Основные характеристики:

- разработка требований, для описания требований в RUP используются прецеденты использования (use cases). Полный набор прецедентов использования системы вместе с логическими отношениями между ними называется моделью прецедентов использования. Каждый прецедент использования – это описание сценариев взаимодействия пользователя с системой, полностью выполняющего конкретную пользовательскую задачу. Согласно RUP все функциональные требования должны быть представлены в виде прецедентов использования.
- итеративная разработка, проект RUP состоит из последовательности итераций с рекомендованной продолжительностью от 2 до 6 недель. Перед началом очередной итерации определяется набор прецедентов использования, которые будут реализованы к её завершению.

AGILE-МЕТОДИКИ

Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto. Agile не включает практик, а определяет ценности и принципы, которыми руководствуются команды.

Agile Manifesto разработан и принят 11—13 февраля 2001 года на лыжном курорте The Lodge at Snowbird в горах Юты. Agile Manifesto содержит 4 основные идеи и 12 принципов. Примечательно, что Agile Manifesto не содержит практических советов.

Основные идеи:

- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;

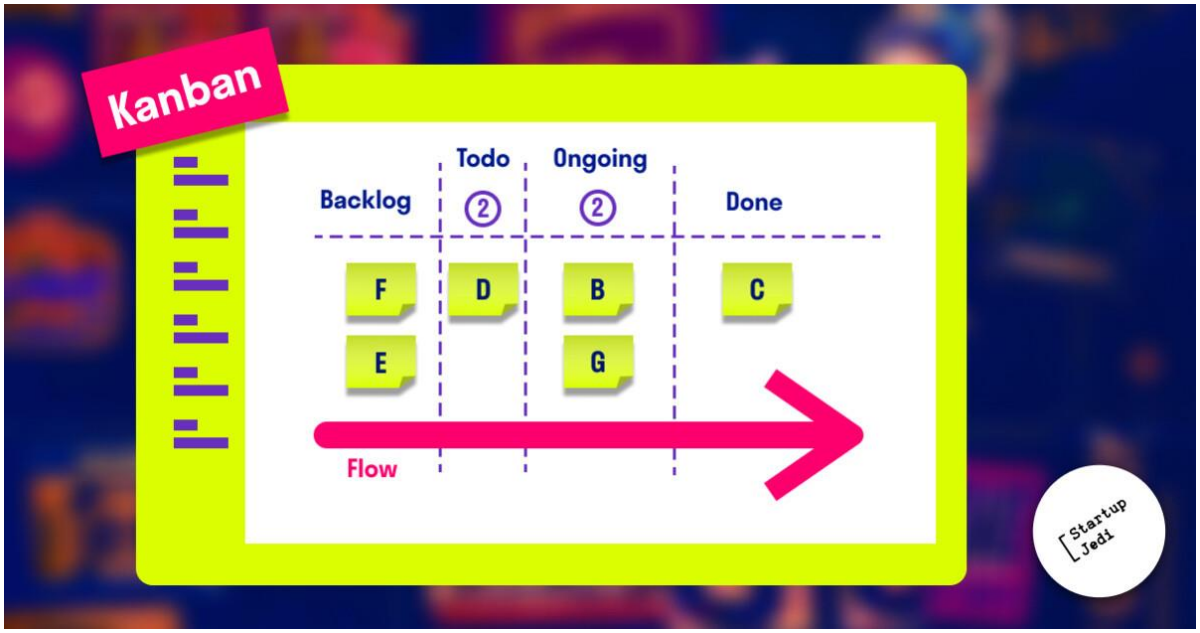
- сотрудничество с заказчиком важнее согласования условий контракта;
- готовность к изменениям важнее следования первоначальному плану.

Принципы, которые разъясняет Agile Manifesto:

- удовлетворение клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения;
- приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);
- частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще);
- тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;
- проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
- рекомендуемый метод передачи информации — личный разговор (лицом к лицу);
- работающее программное обеспечение — лучший измеритель прогресса;
- спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;
- постоянное внимание улучшению технического мастерства и удобному дизайну;
- простота — искусство не делать лишней работы;
- лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;
- постоянная адаптация к изменяющимся обстоятельствам. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Agile имеет множество вариаций и фреймворков. Среди самых известных: Scrum, Kanban, экстремальное программирование (XP), Lean, Crystal Clear.

Kanban



Является одной из методологий Agile. Метод базируется на концепции бережливого производства, основанной на стремлении к устранению всех потерь — временных, производственных, логистических, качественных.

Особенность Kanban — задачи должны выполняться точно в срок, нагрузка между командой распределяется равномерно.

На практике это выглядит следующим образом. Каждая задача по проекту описывается в отдельной карточке и добавляется на доску — виртуальную или настоящую. Карточка и доска — неотъемлемые элементы Kanban. Все задачи, которые необходимо сделать, собраны в специальной колонке, условно, она может называться “сделать”/ “to do”. Исполнитель выбирает задачу и перемещает в колонку “в процессе” / “in progress”. Когда задача сделана, она попадает в соответствующую колонку “готово” / “done”. На практике колонок может быть гораздо больше, чем три. К примеру, колонки на доске могут выглядеть так: “обсуждается” (backlog), “согласовано” (ready), “кодируется” (coding), “тестируется” (testing), “подтверждается” (approval) и “сделано” (done).

Scrum

Одна их самых популярных методологий Agile. В отличие от канбан, у скрама гораздо больше элементов — различные митинги (от ежедневных пятиминутных, до планирований спринтов, демо), четкое разделение по ролям. Кроме того, разработка подразделяется на спринты — которые длятся от недели до четырех недель и заканчиваются выпуском части продукта.

В переводе с английского scrum — это драка либо схватка вокруг мяча. Термин пришел из регби и означает специфическую игровую ситуацию, в которой участники команд смыкаются в три линии с каждой стороны, когда в игру вводится мяч после нарушения правил; задача игроков — выиграть за счет совместных усилий команды.

Почти так же применяется методология Scrum при разработке ПО: команда кросс-функциональных специалистов совместно реализует проект. Важно, чтобы это были профессионалы с высокой мотивацией, инициативные и готовые вносить изменения как в ход реализации проекта, так и в продукт.

Рассмотрим пример применения Scrum.

1. На этапе формирования продукта вы с командой решаете, кто из вас будет исполнять роль product owner — человека, который отвечает за связь команды с потребителем и инвесторами (если ваши инвесторы будут интересоваться ходом разработки продукта).

2. Product owner формирует список пожеланий к продукту, собирает первичную информацию от возможных пользователей и затем формирует бэклог продукта — список задач, выполнение которых в конце концов приведет вас к выходу на рынок.

3. Ваша команда определяет размер спринта — периода, в конце которого вы должны сделать какой-то рабочий кусок продукта, и выбирает задачи для первого спринта из бэклога.

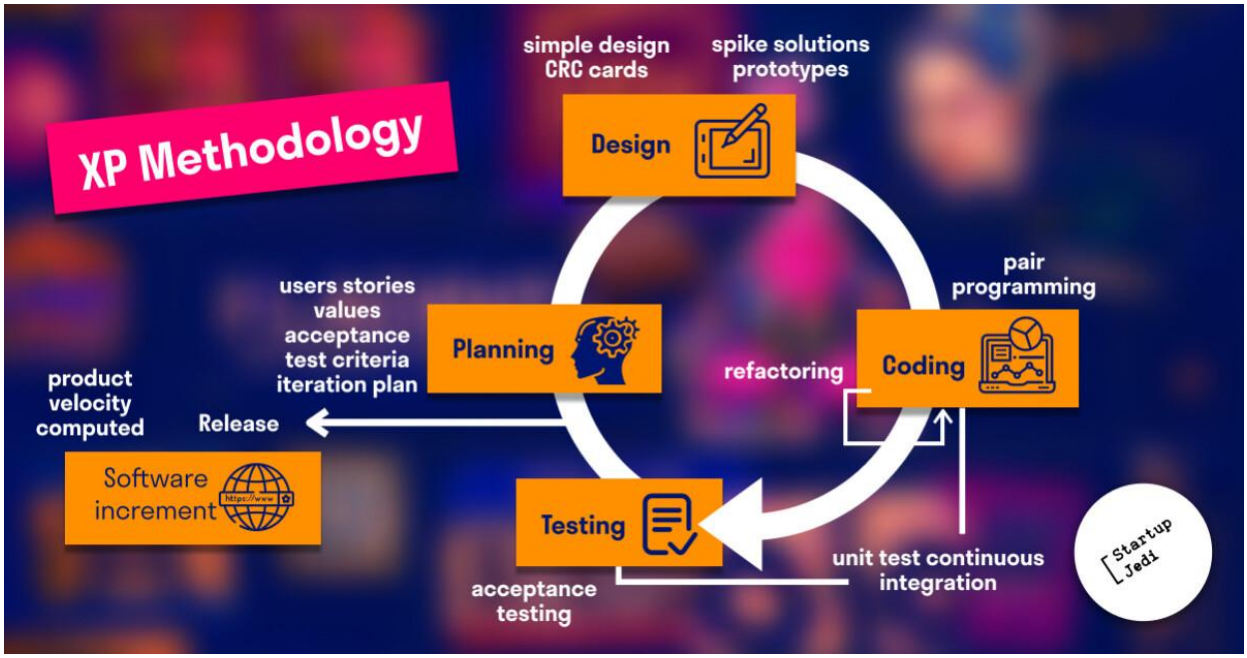
4. В течение первого спринта вы отслеживаете качественные и количественные характеристики своей работы. Неотъемлемая часть скрама — ежедневные короткие (5–10) минут митинги, в течение которых каждый из участников команды рассказывает, что он планирует сделать за день, делится возникающими сложностями или, наоборот, успехами.

5. Ход выполнения задач отслеживается по скрам-доске, на которой все задачи двигаются от условной позиции “сделать” до “выполнено”.

6. По завершении спринта вы демонстрируете выполненную часть работы и собираете обратную связь — от членов команды, клиентов, в т.ч. потенциальных.

7. Цикл спринтов повторяется до того момента, пока продукт не будет полностью завершен.

Экстремальное программирование (XP)



eXtreme Programming, экстремальное программирование, XP — гибкая методология разработки, которая появилась в конце 90-х годов прошлого столетия. Авторы взяли лучшие, на их взгляд, практики гибкой разработки и усилили их до максимума — отсюда и слово “экстремальный” в названии.

В отличие от канбана, скрама, которые можно применять в самых разных стартапах и бизнесах, да и в организации личных дел, XP применяется исключительно в разработке программных продуктов. В рамках экстремального программирования выделяются четыре процесса: кодирование, тестирование, дизайн, слушание. Если описать эту методологию несколькими словами, то ее характеризуют оперативность, высокое качество, командная работа.

Особенностью XP являются некоторые практики, самая известная из которых — парное программирование. Суть его заключается в том, что два разработчика

одновременно работают над кодом для одной функции продукта: сначала один пишет, а второй наблюдает и исправляет ошибки, затем они меняются местами. Таким образом, в процессе создания кода есть два альтернативных решения, на каждом этапе выбирается лучшее. Парное программирование работает по принципу: одна голова — хорошо, а две лучшею

Другая особенность экстремального программирования заключается в том, что сначала готовятся тесты, и только потом — код. При этом тесты пишут сами программисты. Тестирование позволяет исправить большинство ошибок на стадии создания кода.

Третья особенность — коллективное владение кодом: каждый программист в команде имеет доступ к коду продукта и может вносить в него изменения. В том случае, если изменения привели к некорректной работе системы, исправить все должен тот программист, который внес эти изменения.

Экстремальное программирование предполагает также работу в рамках небольших релизов — от одного дня до месяца. При этом чем короче релизы, тем лучше качество продукта.

Наконец, интеграция новых частей в систему происходит так быстро как это возможно. Как только тесты показали, что функция работает корректно, она интегрируется в систему.

Выводы

Как видим, даже если вам кажется, что вы работаете “без заморочек” — без всяких там методологий и прочего — на самом деле даже это прописано в теории. Скорее всего, вы работаете по модели Code-and-Fix и, возможно, уже совсем скоро столкнетесь со всеми ее недостатками. Чтобы их избежать, важно перед началом работы над продуктом проанализировать его и вместе с командой, решить, по каким фазам будет идти разработка. Оптимальный вариант выбирается, исходя их того, сформировали вы уже конкретные требования к своему продукту или планируете “совершенствовать” его во время работы; какой у вас бюджет и насколько кросс-функциональна команда; насколько длителен проект по времени выполнения и как обстоят дела с финансированием.

Нет идеальной модели и нет идеальной методологии, однако в ваших силах подобрать такую, которая позволит вам максимально эффективно выстроить свою работу и в конце концов вывести продукт на рынок.