**РАЗДЕЛ 9**

**ОРГАНИЗАЦИЯ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**1.Методологии разработки программного обеспечения – каскадная, спиральная, инкрементальная. Их сущность и области применения.**

**Каскадная (водопад)**

Данная модель также носит название «водопад». Классическими представителями реализации данной методологии являются стандарты ISO и CMM.

Данная модель предполагает строго последовательное (во времени) и однократное выполнение всех фаз проекта с жестким (детальным) предварительным планированием в контексте предопределенных или однажды и целиком определенных требований к программной системе.

Модель предполагает следующие свойства взаимодействия этапов:

• модель состоит из последовательно расположенных этапов;

• каждый этап полностью заканчивается до того, как начнется следующий;

• этапы не перекрываются во времени: следующий этап не начинается до тех пор, пока не завершится предыдущий;

• возврат к предыдущим этапам не предусмотрен либо всячески ограничен;

• исправление ошибок происходит лишь на стадии тестирования;

• результат появляется только в конце разработки.

Критерием появления результата является отсутствие ошибок и точное соответствие продукта первоначальной спецификации.

**Недостатки**

a) требования к объектам определены недостаточно четко;

b) система обычно слишком велика, чтобы все работы по ее созданию выполнять однократно;

c) предполагаемые скорые изменения в технологиях работ;

Преимущества

a) однократное представление всех возможностей (характеристик) системы;

b) необходимость только единственной фазы перехода от старой системы к новой.

**Инкрементальная (итеративная)**

Итеративная модель предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает “мини-проект”, включая все фазы жизненного цикла в применении к созданию меньших фрагментов функциональности, по сравнению с проектом, в целом. Цель каждой итерации – получение работающей версии программной системы, включающей функциональность, определенную интегрированным содержанием всех предыдущих и текущей итерации. Результата финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации,продукт развивается инкрементально.

С точки зрения структуры жизненного цикла такую модель называют итеративной (iterative). С точки зрения развития продукта – инкрементальной (incremental). Опыт индустрии показывает, что невозможно рассматривать каждый из этих взглядов изолировано. Чаще всего такую смешанную эволюционную модель называют просто итеративной (говоря о процессе) и/или инкрементальной (говоря о наращивании функциональности продукта).

Недостатки

а) требования к объектам определены недостаточно четко;

b) предусмотрены сразу все возможности системы;

c) предполагаемые скорые изменения в технологиях работ;

d) возможные текущие изменения требований к системе;

е) привлечение ресурсов (средств или персонала) на длительный период ограничено.

Преимущества

a) необходимость изначального использования характеристик системы;

b) пригодность для использования промежуточного продукта;

c) естественное разделение системы на наращиваемые компонент (инкременты);

d) возможности наращивания привлекаемого персонала и средств.

**Спиральная модель**

Спиральная модель была впервые сформулирована Барри Боэмом. Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Боэм формулирует “top-10” наиболее распространенных (по приоритетам) рисков:

Дефицит специалистов.

Нереалистичные сроки и бюджет.

Реализация несоответствующей функциональности.

Разработка неправильного пользовательского интерфейса.

“Золотая сервировка”, перфекционизм, ненужная оптимизация и оттачивание деталей.

Непрекращающийся поток изменений.

Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.

Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.

Недостаточная производительность получаемой системы.

“Разрыв” в квалификации специалистов разных областей знаний.

Большая часть этих рисков связана с организационными и процессными аспектами взаимодействия специалистов в проектной команде.

Коммерческими представителями данной методологии являются RUP (Rational Unified Process), MSF (Microsoft Consulting Services). Результат появляется фактически на каждом витке спирали. Этот результат, который является промежуточным, анализируется, а затем выявленные недостатки продукта становятся поводом для инициирования следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в итоге выбирается обоснованный вариант, который доводится до реализации. Спираль завершается тогда, когда клиент и разработчик приходят к согласию относительно результата.

Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Каждый виток разбит на 4 сектора:

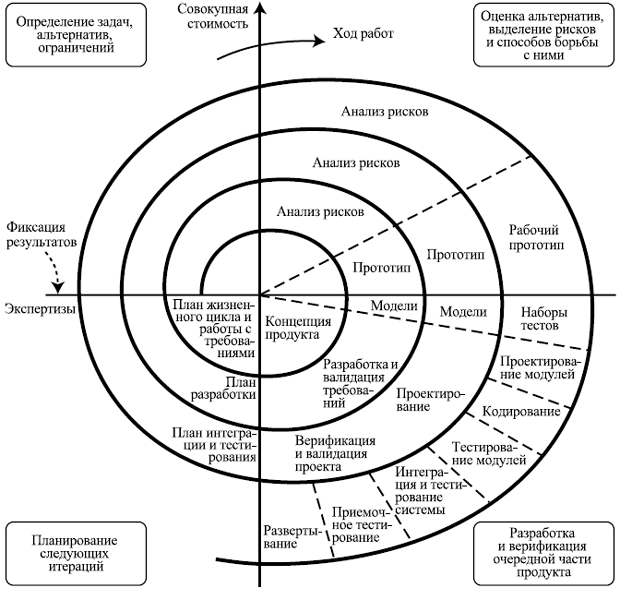
оценка и разрешение рисков,

определение целей,

разработка и тестирование,

планирование.

На каждом витке спирали могут применяться разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт. Модель сочетает в себе возможности модели прототипирования и водопадной модели. Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная задача — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований. Основная проблема спирального цикла — определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков. Одним из возможных подходов к разработке программного обеспечения в рамках спиральной модели жизненного цикла является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development). Под этим термином обычно понимается процесс разработки программного обеспечения, содержащий 3 элемента:

небольшую команду программистов (от 2 до 10 человек);

короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);

повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Спиральная модель ориентирована на большие, дорогостоящие и сложные проекты. В условиях, когда бизнес цели таких проектов могут измениться, но требуется разработка стабильной архитектуры, удовлетворяющей высоким требованиям по нагрузке и устойчивости, имеет смысл применение Spiral Architecture Driven Development. Данная методология, включающая в себя лучшие идеи спиральной модели и некоторых других, позволяет существенно снизить архитектурные риски, что является немаловажным фактором успеха при разработке крупных систем.

**2.Методологии Rational Unified Process: сущность и области применения.**

Ведущей методологией, в которой инструментально поддерживаются все этапы жизненного цикла разработки ПО, является методология Rational Unified Process (RUP). Она опирается на проверенные практикой методы анализа, проектирования и разработки ПО, методы управления проектами. RUP обеспечивает прозрачность и управляемость процесса и позволяет создавать ПО в соответствии с требованиями заказчика на момент сдачи ПО, а также в соответствии с возможностями инструментальных средств поддержки разработки.

Уровень формализации процесса разработки ПО в рамках этой методологии, позволяет определить и понять, кто должен участвовать в разработке, каковы его функции в этом процессе и на каких этапах.

По концепции RUP любой проект состоит из мини проектов (выпусков) для появления на свет каждого из которых (как и самого проекта) необходимо выполнить ряд параллельных потоков работ в совокупности называемых жизненным циклом:

1) Определение требований;

2) Анализ требований и проектирование;

3) Выполнение (реализация);

4) Тестирование;

5) Внедрение (развертывание);

В зависимости от распределения интенсивности работ на разных этапах или их наборе процесс разработки в целом также условно делят на фазы:

1) Начало

2) Уточнение

3) Конструирование

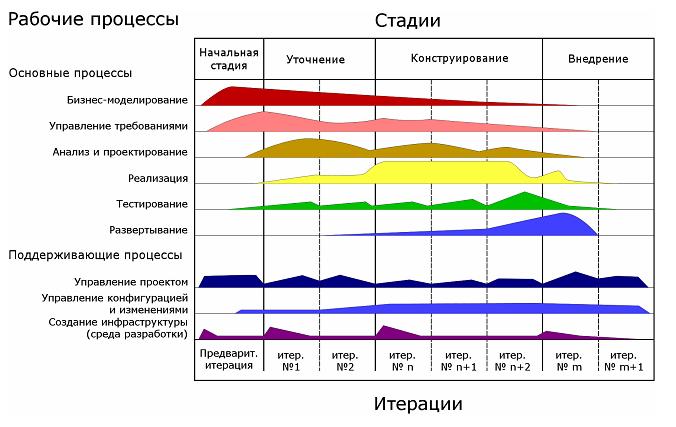
4) Выпуск

По мере хода потоков работ пик в распределении трудозатрат на каждый из них наступает поочередно. Сначала основные затраты идут на сбор и определение требований к разрабатываемой системе. Когда основная доля требований определена, интенсивность работ плавно переходит на анализ собранных требований и проектирование системы на основании этих требований и анализа и т.д. Таким образом каждый этап цикла выполняясь параллельно с остальными, опирается на результат предыдущих этапов и циклов.

Такой итеративный подход позволяет улучшать понимание проблемы через последовательные усовершенствования системы при учете того, что основной каркас (архитектура) системы разрабатывается (проектируется) в самом начале и в дальнейшем происходит лишь наполнение (уточнение).

Разбиение процесса разработки программ на итерации позволяет выполнять в первую очередь те части проекта, реализация которых подвержена наибольшему риску, и как следствие - снизить или устранить этот риск или в случае невозможности преодолеть, отказаться от разработки на ранних стадиях, когда потрачено не так много сил и средств. Кроме того такой подход позволяет контролировать и управлять ходом работы, поскольку при этом видно что сделано, что делается и что осталось сделать и, при необходимости, перерасперделять силы на более важные части.

Итак, RUP - это контролируемый и управляемый процесс, с четкой регламентацией ролей разработчиков, позволяющий разрабатывать ПО максимально точно в соответствии с меняющимися в ходе разработки требованиями и в достаточно предсказуемые сроки.

 Структуру жизненного цикла проекта, выполняемого по технологии RUP удобно рассматривать на координатной плоскости. При этом по горизонтальной оси отложено время, а по вертикальной — основные деятельности, которые обычно выполняются в ходе любого проекта, претендующего на статус успешного.

**Рисунок: Распределение усилий при выполнении проекта:**

В ходе жизненного цикла проекта распределение усилий проектной команды между дисциплинами постоянно меняется. Например, как правило, в начале проекта большая часть усилий затрачивается на анализ и дизайн, а ближе к завершению – на реализацию и тестирование системы. Однако в общем случае задачи из всех девяти дисциплин выполняются параллельно.

Для полноценного внедрения RUP организация должна затратить значительные средства на обучение сотрудников. При этом попытка обойтись своими силами скорее всего будет обречена на неудачу – необходимо искать специалиста по процессам (process engineer) с соответствующим опытом или привлекать консультантов.

**3.Гибкие методологии разработки программного обеспечения. Их сущность, примеры и области применения.**

**Гибкие методологии разработки (англ. *Agile software development*)**

Большинство гибких методологий нацелены на минимизацию рисков, путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели. Каждая итерация сама по себе выглядит как программный проект в миниатюре, и включает все задачи, необходимые для выдачи мини-прироста по функциональности: планирование, анализ требований, проектирование, кодирование, тестирование и документирование. Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации, команда выполняет переоценку приоритетов разработки

Agile-методы делают упор на непосредственное общение лицом к лицу. Отдавая предпочтение непосредственному общению, agile-методы уменьшают объем письменной документации, по сравнению с другими методами. Это привело к критике этих методов, как недисциплинированных.

Многие руководители проектов, работающие в традиционных методологиях вроде «водопада», критикуют agile-методы. Один из повторяющихся пунктов критики: при agile-подходе часто пренебрегают созданием «дорожной карты» развития продукта, равно как и управлением требованиями, в процессе которого и формируется такая «карта». Гибкий подход к управлению требованиями не подразумевает далеко идущих планов (по сути управление требованиями просто не существует в данной методологии), а подразумевает возможность заказчика вдруг и неожиданно, в конце каждой итерации, выставлять новые требования, часто противоречащие архитектуре уже созданного и поставляемого продукта. Такое иногда приводит к катастрофическим «авралам» с массовым рефакторингом и переделками практически на каждой очередной итерации. Кроме того, считается, что работа в agile мотивирует разработчиков решать все поступившие задачи простейшим и быстрейшим возможным способом, при этом зачастую не обращая внимания на правильность кода с точки зрения требований нижележащей платформы (подход — «работает, и ладно», при этом не учитывается, что может перестать работать при малейшем изменении или же дать тяжёлые к воспроизводству дефекты после реального развёртывания у клиента). Это приводит к снижению качества продукта и накоплению дефектов.

Наиболее известные методики:

**+ Экстрема́льное программи́рование (билет 4)**

**+ Методология SCRUM (билет 5)**

**+ Microsoft Solutions Framework (MSF)** методология разработки программного обеспечения, предложенная корпорацией Microsoft. MSF опирается на практический опыт Microsoft и описывает управление людьми и рабочими процессами в процессе разработки решения.

Модель проектной группы MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта.

В соответствии с моделью MSF проектные группы строятся как небольшие многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга. Это дает возможность четко сфокусировать внимание на нуждах проекта. Проектную группу объединяет единое видение проекта, стремление к воплощению его в жизнь, высокие требования к качеству работы и желание самосовершенствоваться.

MSF включает в себя ряд основных принципов. Вот те из них, которые имеют отношение к успешной работе команды:

1. Распределение ответственности при фиксации отчетности

2. Наделяйте членов команды полномочиями

3. Концентрируйтесь на бизнес-приоритетах

4. Единое видение проекта

5. Проявляйте гибкость — будьте готовы к переменам

6. Поощряйте свободное общение

**+Канбан** - система организации производства и снабжения, позволяющая реализовать принцип «точно в срок».

Бережливое производство.

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

Например, общеизвестная проблема SCRUM - это большие издержки от обсуждений, встреч и большие потери времени на стыках спринтов (когда как минимум день уходит на закрытие одного спринта, а потом день на открытие нового. И если спринт - 2 недели, то 2 дня из 2 недель - это 20%, чертовски много). В итоге чуть ли не 30-40% времени при применении SCRUM тратится на поддержание самого процесса - на ежедневные митинги, на 5% workshop, на спринт ретроспектив и т.п. 30%!

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.

Что нового и полезного дает такая доска с лимитами?

Во-первых, уменьшение числа параллельно выполняемых задач сильно уменьшает время выполнения каждой отдельной задачи. Нет нужды переключать контекст между задачами, отслеживать разные сущности, планировать их и т.д. - делается только то, что нужно. Нет нужды устраивать спринт планнинги и 5% воркшопы, т.к. планирование уже сделано в столбце “очередь задач”, а детальная проработка задачи начинается ТОЛЬКО тогда, когда задача начинает выполняться.

Во-вторых, сразу видны затыки. Например, если тестеры не справляются с тестированием, то они очень скоро заполнят весь свой столбец и программисты, закончившие новую задачу, уже не смогут переместить ее в столбец тестирования, т.к. он заполнен. Что делать? Тут время вспомнить, что “мы - команда” и решить эту проблему. Например, программисты могут помочь тестерам завершить одну из задач тестирования и только тогда передвинуть новую задачу на освободившееся место. Это позволит выполнить обе задачи быстрее.

В-третьих, можно вычислить время на выполнение усредненной задачи. Мы можем помечать на карточке дату, когда она попала в очередь задач, потом дату, когда ее взяли в работу и дату, когда ее завершили. По этим трем точкам для хотя бы 10 задач можно уже посчитать среднее время ожидания в очередь задач и среднее время выполнения задачи. А из этих цифр менеджер или product owner может уже рассчитывать всё, что ему угодно.

Весь Канбан можно описать всего тремя основными правилами:

1. Визуализируйте производство

- Разделите работу на задачи, каждую задачу напишите на карточке и поместите на стену или доску.

- Используйте названные столбцы, чтобы показать положение задачи в производстве.

2. Ограничивайте WIP (work in progress или работу, выполняемую одновременно) на каждом этапе производства.

3. Измеряйте время цикла (среднее время на выполнение одной задачи) и оптимизируйте постоянно процесс, чтобы уменьшить это время.

Всего 3 правила!

Например, в SCRUM - 9 базовых правил. В XP - 13,а в классическом RUP - аж более 120. Почувствуйте разницуJ

**Приблизительная сравнительная таблица**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | scrum | xp | msf | kanban |
| деление на команды | 1-4 команды или больше. По 7+(-)2 человека | Одна команда на проект.Коллективное владение кодом (парное программирование с перемешиванием). 3-16 чел | многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетенций друг друга (6 кластеров - основных ролей), минимум может быть - 3 человека |  |
| выделение сроков | да, для каждого спринта (2-4 недели) |  |  | нет, только среднее время на реализацию всей задачи |
| наличие правил | 9 | 12 |  | 3 |
| проверки | частые совещания | каждодневные тесты |  | доска, проверка после выполнения задачи |
| задачи | частые задачи | чем чаще выходит "мини-продукт", тем лучше |  | задач меньше, но они больше |
| участники команды | Scrum master, Experienced Engineer, Junior Engineer, [QA Tester], [Writer] | Customer, Programmer, Tester, Tracker, Coach | program manager, developer, QAE, release manager, product manager |  |

**4.Экстремальное программирование: сущность и области применения**

Экстрема́льное программи́рование ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Extreme Programming, XP) — одна из гибких методологий разработки программного обеспечения.

XP — является ответом сообщества программистов на наступление формальных подходов к созданию программных продуктов и призвано вернуть в среду разработчиков дух творчества. Неоправданное, гипертрофированное внимание к Процессу в ущерб другим факторам разработки породило массу формальных процедур — но качество полученных продуктов и количество успешных проектов оказалось разочаровывающим.

В основе экстремального программирования лежат не конкретные методики, как принято считать, а лишь четыре базовых принципа: общение, простота, обратная связь и смелость. Именно с них необходимо начинать.

В командах, работающих по методу XP, всегда приветствуется *общение* - самое быстрое средство обмена информацией и опытом. Это очень важно, когда требуется максимальная скорость разработки. Общение и необходимость объяснения своих действий другим членам команды вынуждает делать все максимально *просто*. Если не получается с первого раза, над упрощением работают еще и еще, пока не будет достигнута главная цель - максимальная понятность кода другим разработчикам.

Что бы мы ни делали, мы всегда стремимся достичь какой-то цели. Если мы замечаем, что отклоняемся от нее, то корректируем свои действия соответствующим образом. Поэтому в экстремальном программировании принято за правило видеть результат своих действий настолько быстро, насколько это возможно. Или, говоря техническим языком, обеспечить максимально быструю *обратную связь.*

Экстремальное программирование спрашивает нас: почему бы не воспитать в себе *смелость*? Ведь она очень важна в работе. Разве можно без смелости принять на себя ответственность за выполнение какой-то задачи, да еще в конкретные сроки? Разве можно без смелости осознать, что ты уперся в тупик, сделать шаг назад и поискать обходной путь? И, наконец, что позволит разработчику признать свою ошибку в оценке задачи и вовремя предупредить об этом остальных вместо того, чтобы поставить их перед фактом уже тогда, когда все сроки истекут? Польза храбрости налицо, и каждый успех, даже в самой маленькой задачке, способен эту храбрость развить.

В XP тестирование является той основой, на которой строится разработка. При этом каждый программист пишет тесты одновременно с кодом разрабатываемой системы. Эти тесты используются при постоянной интеграции и в процессе сборки системы, что дает стабильный фундамент для дальнейшей работы.

На этом фундаменте ХР строит эволюционный процесс проектирования, основанный на реорганизации кода системы в течение каждой последующей итерации. При этом проектируется только та функциональность, которая относится к текущей итерации, а любые будущие потребности не учитываются. Получившийся в результате процесс требует от разработчиков дисциплины, и в то же время сочетает ее с высокой адаптивностью. Такое удивительное сочетание позволяет предположить, что ХР является наиболее развитой адаптивной методологией.

**Основные приёмы XP**

Тестирование

В XP особое внимание уделяется двум разновидностям тестирования:

§ тестирование модулей (unit testing);

§ приемочное тестирование (acceptance testing).

Тесты модулей позволяют разработчикам убедиться в том, что их код работает корректно. Они также помогают другим разработчикам понять, зачем нужен тот или иной фрагмент кода и как он функционирует.

Приемочные тесты позволяют убедиться в том, что система действительно обладает заявленными возможностями. Кроме того, приемочные тесты позволяют проверить корректность функционирования разрабатываемого продукта.

Игра в планирование

Основная цель игры в планирование — быстро сформировать приблизительный план работы и постоянно обновлять его по мере того, как условия задачи становятся все более четкими. Артефактами игры в планирование является набор бумажных карточек, на которых записаны пожелания заказчика (customer stories), и приблизительный план работы по выпуску следующих одной или нескольких небольших версий продукта.

Заказчик всегда рядом

«Заказчик» в XP — это не тот, кто оплачивает счета, а конечный пользователь программного продукта. XP утверждает, что заказчик должен быть всё время на связи и доступен для вопросов.

Парное программирование

Парное программирование предполагает, что весь код создается парами программистов, работающих за одним компьютером. Один из них работает непосредственно с текстом программы, другой просматривает его работу и следит за общей картиной происходящего. При необходимости клавиатура свободно передается от одного к другому. В течение работы над проектом пары не фиксированы: рекомендуется их перемешивать, чтобы каждый программист в команде имел хорошее представление о всей системе. Таким образом, парное программирование усиливает взаимодействие внутри команды.

Непрерывная интеграция

Если вы будете выполнять интеграцию разрабатываемой системы достаточно часто, вы сможете избежать большей части связанных с этим проблем. В XP интеграция кода всей системы выполняется несколько раз в день, после того, как разработчики убедились в том, что все тесты модулей корректно срабатывают.

Рефакторинг

XP подразумевает, что однажды написанный код в процессе работы над проектом почти наверняка будет неоднократно переделан. Разработчики XP безжалостно переделывают написанный ранее код для того, чтобы улучшить его. Этот процесс называется рефакторингом.

Частые небольшие релизы

Версии (releases) продукта должны поступать в эксплуатацию как можно чаще. Работа над каждой версией должна занимать как можно меньше времени. При этом каждая версия должна быть достаточно осмысленной с точки зрения полезности для бизнеса.

Чем раньше мы выпустим первую рабочую версию продукта, тем раньше заказчик начнет получать за счет нее дополнительную прибыль. Следует помнить, что деньги, заработанные сегодня, стоят дороже, чем деньги, заработанные завтра. Чем раньше заказчик приступит к эксплуатации продукта, тем раньше разработчики получат от него информацию о том, что соответствует требованиям заказчика. Эта информация может оказаться чрезвычайно полезной при планировании следующего выпуска.

Простота дизайна

XP исходит из того, что в процессе работы условия задачи могут неоднократно измениться, а значит, разрабатываемый продукт не следует проектировать заблаговременно целиком и полностью. Если в самом начале работы вы пытаетесь от начала и до конца детально спроектировать систему, вы напрасно тратите время.

Стандарты кодирования

Все члены команды в ходе работы должны соблюдать требования общих стандартов кодирования. Благодаря этому:

§ члены команды не тратят время на глупые споры о вещах, которые фактически никак не влияют на скорость работы над проектом;

§ обеспечивается эффективное выполнение остальных практик.

Команда должна сформировать набор правил, а затем каждый член команды должен следовать этим правилам в процессе кодирования. Перечень правил не должен быть исчерпывающим или слишком объемным. Задача состоит в том, чтобы сформулировать общие указания, благодаря которым код станет понятным для каждого из членов команды. Стандарт кодирования поначалу должен быть простым, затем он может постепенно усложняться по мере наработки опыта группой разработчиков. Не нужно тратить слишком много времени на предварительную разработку стандарта.

Коллективное владение

Коллективное владение означает, что каждый член команды несёт ответственность за весь [исходный код](http://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Таким образом, каждый вправе вносить изменения в любой участок программы. Парное программирование поддерживает эту практику: работая в разных парах, все программисты знакомятся со всеми частями кода системы. Важное преимущество коллективного владения кодом — в том, что оно ускоряет процесс разработки, поскольку при появлении ошибки её может устранить любой программист.

**Применение**: Во многом XP была создана как попытка описать процессы и практики, которые часто как бы сами собой появляются в эффективных, сплоченных командах разработчиков. Может показаться, что процесс в таких командах отсутствует, и, скорее всего, то же самое будут утверждать сами разработчики. Однако это не так, поскольку работа профессиональной команды всегда четко (хотя и неформально) организована и не имеет ничего общего с беспорядком и хаосом.

Это во многом определяет условия, необходимые для эффективного использования XP. Прежде всего, XP имеет шансы работать только в команде опытных, профессиональных разработчиков. Поскольку большую роль в XP играет прямое общение, команда не должна быть разбита на несколько частей – внедрение XP в распределенной географически команде будет крайне рискованным мероприятием. По той же причине возможный размер команды ограничен сверху – по всей видимости, числом в 10-15 человек.

Другие практики XP приносят свои ограничения. Далеко не всегда можно обеспечить постоянное присутствие представителя заказчика в проектной команде (например, если потенциальные пользователи системы делятся на несколько классов с частично конкурирующими требованиями).

Поскольку XP практически не делает попыток предотвратить размывание границ проекта (scope creep), будет не очень хорошей идеей использовать XP в проекте с фиксированной ценой. Фактически проекты XP обладают жестким графиком, но переменными границами, поэтому предпочтительным типом контракта будет повременная оплата (time&materials).

Практика поддержания максимально простой архитектуры может завести в тупик в конце проекта, когда окажется, что для реализации завершающих историй требуется полное перепроектирование системы (оказывается, нам нужно было предусмотреть возможность интеграции с системой SAP R/3, а также перевода на японский язык всего пользовательского интерфейса!). Даже если подобная ситуация не возникнет, нет никакой гарантии, что создаваемая ad hoc архитектура не будет намного более запутанной и сложной, чем было бы продуманное заранее решение.

Подобным образом может неконтролируемо откладываться разрешение некоторых нетехнологических рисков, наподобие неопределенных границ проекта.

Несмотря на все перечисленные ограничения, XP может замечательно работать в подходящих для него условиях. Благодаря крайне низким накладным расходам, в таких ситуациях этот процесс может показать исключительную эффективность.

XP является достаточно гибкой методологией. Не обязательно внедрять XP во всей компании, вполне разумно ограничиться теми командами и проектами, которые могут получить от этого реальный выигрыш. Например, для разработки ядра продукта можно использовать XP, а проекты по внедрению основывать на процессе RUP. Также не обязательно использовать все классические практики XP (на самом деле, мало кто это делает) – как правило, разумно ограничиться теми из них, которые сочетаются с корпоративной культурой и особенностями проектов.

Scrum — это платформа для процессов гибкой разработки. Она не включает в себя конкретные инженерные приемы. Напротив, XP фокусируется на инженерных приемах, но не включает в себя всеохватывающую платформу процессов разработки. Это не значит, что Scrum не рекомендует определенные инженерные приемы, или что в XP нет процесса. Например, первый Scrum реализовал все инженерные приемы, которые теперь известны как XP. Однако платформа Scrum для разработки программного обеспечения была призвана начать работу команды в течение двух или трех дней, в то время как для реализации инженерных приемов часто требуются многие месяцы. Поэтому вопрос, когда (и стоит ли) внедрять те или иные приемы, остается на рассмотрение каждой команды. Соавторы Scrum Джеф Сазерленд и Кен Швабер рекомендуют командам Scrum немедленно начинать работу и создать список препятствий и план улучшения процесса. Так как инженерные приемы идентифицируются как препятствия, командам следует обращаться к приемам XP как к способу улучшения. Лучшие команды реализуют Scrum, дополненный приемами XP. Scrum помогает XP масштабироваться, а XP способствует надлежащей работе Scrum.

**5.Методология SCRUM: сущность и области применения.**

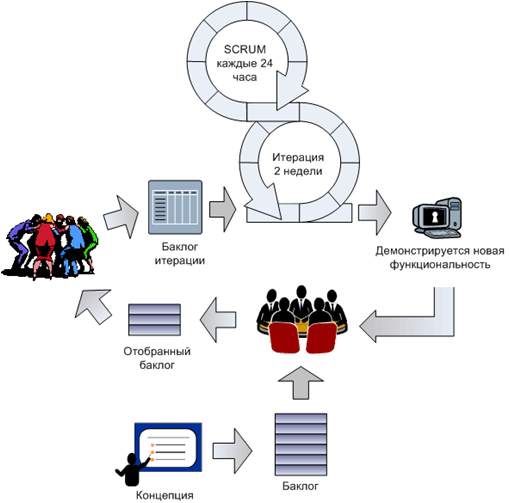
Scrum — методология управления разработкой информационных систем для [гибкой разработки программного обеспечения](http://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B1%D0%BA%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F). Scrum чётко делает акцент на качественном контроле процесса разработки.

Scrum — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные небольшие промежутки времени (спринты от 2 до 4 недель) предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Главные действующие роли в Scrum:

ScrumMaster — тот, кто ведёт Scrum митинги и следит, чтобы при этом соблюдались все принципы Scrum (роль не предполагает ничего кроме корректного ведения самогоScrum-а, руководитель проекта скорее относится к Product Owner и не должен являться ScrumMaster);

Владелец Продукта (Product Owner) — человек, который представляет интересы конечных пользователей и других заинтересованных в продукте сторон;

Команда (Scrum Team), состоящая как из разработчиков, так и из тестировщиков, архитекторов, аналитиков и т. д. (при этом размер команды в идеале составляет 7±2 человека). Команда является единственным полностью вовлечённым участником разработки, и отвечает за результат как единое целое. Никто кроме команды не может вмешиваться в процесс разработки на протяжении спринта.

Ежедневно проводятся мини-митинги, в конце каждого месяца – большое собрание.

На протяжении каждого спринта создаётся функциональный рост программного обеспечения. Набор возможностей, которые реализуются в каждом спринте, происходят из этапа, называемого product backlog (список бизнес-требований и технических требований), обладающего наивысшим приоритетом по уровню требований к работе, который должен быть выполнен. Запросы на выполнение работ (backlog items), определенных на протяжении совета по планированию спринта (sprint planning meeting), перемещаются в этап спринта. На протяжении этого собрания Владелец Продукта информирует о заданиях, которые должны быть выполнены. Тогда Команда определяет, сколько из желаемого они могут выполнить, чтобы завершить необходимые части на протяжении следующего спринта. Во время спринта команда выполняет определенный фиксированный список заданий (т. н. sprint backlog). На протяжении этого периода никто не имеет права менять список требований к работе, что следует понимать как заморозку требований (requirements) во время спринта.

**6.Стандарты в описании процессов жизненного цикла разработки программного обеспечения. Стандарты ИСО в области системной и программной инженерии. Корпоративные стандарты.**

**7.Управление конфигурацией: содержание процесса управления конфигурацией, организация информационной поддержки процесса управления конфигурацией**

Управление конфигурацией – один из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО ИС.

Конфигурация ПО - совокупность функциональных и физических характеристик, установленных в технической документации и реализованных в ПО.

Процесс управления конфигурацией включает:

* идентификацию конфигурации;
* контроль конфигурации;
* учёт состояния конфигурации;
* оценку конфигурации;
* управление выпуском и поставку.

При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учёта их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учёта, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO/IEC 12207.

К управлению конфигурацией следует отнести функции анализа производительности и оптимизации системы.

Большинство систем имеют оптимальные настройки по умолчанию и не требуют особого вмешательства. Однако производители сетевых операционных систем включают в них наборы эмпирических правил, помогающих администратору вносить изменения в настройки с минимальным риском ухудшить другие показатели или сделать систему неработоспособной. Администратору следует их изучить и знать перечень параметров, которые необходимо контролировать. Многих проблем можно избежать еще на стадии планирования сети.

Сюда же можно отнести задачу, связанную с учётом системных ресурсов. Учёт ресурсов позволяет заметить тенденции к появлению узких мест до того, как появятся проблемы с производительностью и провести соответствующую модернизацию. Кроме того, система учёта необходима при платном использовании ресурсов, например, контроль использования дискового пространства, печати, учёт трафика.

ИС на каждом этапе ее построения обладает определенной базовой для данного этапа конфигурацией. Соответственно и процессы проекта должны быть выстроены таким образом, чтобы в результате их осуществления была достигнута именно эта конфигурация.

Управление конфигурацией ИС и процессами проекта позволяет координировать управление перечисленными видами деятельности, исходя из тех изменений, которые постоянно возникают в ходе реализации проекта.

В плане управления конфигурацией в компании следует:

установить и поддерживать базовые конфигурации;

иметь опись (карту) ИС, актуализируемую с учетом жизненного цикла, в которую входят аппаратура, программное обеспечение и документация;

установить и обеспечить практическое применение настроек для конфигурирования средств безопасности в продуктах, входящих в ИС.

Цели управления конфигурацией:

\* контроль вносимых изменений;

\* улучшение качества продукта или услуги;

\* повышение степени удовлетворенности пользователей и/или заказчиков;

\* организация взаимодействия различных рабочих групп. Действия:

\* создание или обновление рабочего пространства по заданному профилю;

\* внесение изменений в файлы проекта;

\* интеграция изменений с изменениями, внесенными другими участниками;

\* фиксирование базовой линии текущих версий файлов проекта;

\* регистрация запросов;

\* назначение исполнителей и сроков;

\* контроль исполнения (периодический контроль).

Правильная реализация дисциплины управления конфигурацией при разработке и сопровождении ПО позволяет значительно сократить финансовые потери. При принятии решения о внедрении процесса УК в организации необходимо учитывать как прямые, так и кос венные преимущества и затраты.

К прямым преимуществам можно отнести повышение производительности труда, которое обычно поддается подсчету. К косвенным преимуществам относится увеличение доли рынка за счет более быстрого вывода на рынок новых продуктов, что довольно сложно поддается подсчету, но может принести большую выгоду.

Консультанты компании «Ай-Теко» обладают достаточной квалификацией и практическим опытом проведения проектов по внедрению и созданию системы автоматизации процесса управления конфигурациями. Ниже предлагается примерный вариант построения процесса:

* осуществить планирование и проектирование процесса управления конфигурациями в реальных условиях функционирования организации;
* выбрать оптимальные для данного случая технологии, которые будут использоваться при построении и функционировании процесса;
* осуществить проектирование и наполнение Базы Данных Учётных Элементов;
* осуществить дополнительную настройку программных средств в соответствии с пожеланиями заказчика;
* обучить персонал работе в соответствии с разработанными процедурами при помощи программных средств;
* осуществить пилотный ввод процесса в эксплуатацию;
* разработать и осуществить документирование процедур совершенствования процесса, учёта известных ошибок и вносимых изменений;
* осуществить доработку процесса в соответствии с накопленным в ходе эксплуатации опытом;
* окончательно передать процесс в эксплуатацию заказчику.

**8.Аудит проекта разработки ПО. Место аудита в общей структуре управления проектом. Роль стандартов в процессе аудита**

У компаний, внедряющих на своих предприятиях автоматизированную систему, нередко возникают вопросы: правильно ли реализуется этот проект, будет ли получен ожидаемый эффект и не окажутся ли деньги, потраченные на автоматизацию, выброшенными на ветер. Кроме того, при внедрении системы возможны разногласия между руководителями предприятия в вопросах, адекватности полученных от автоматизации результатов. Получить ответы на подобные вопросы, выявить существующие недостатки ведения проекта и разработать меры по их устранению, можно с помощью услуги аудита проекта внедрения информационной системы (ИС).

Специалисты компаний рекомендуют работы по аудиту проекта внедрения ИС в следующих типовых случаях:

* если **длительность** проекта внедрения информационной системы составляет **свыше полугода**;
* при значительном **отклонении фактических показателей сроков или бюджета** проекта, от предусмотренных в плане проекта;
* при существенном **изменении целей проекта или его рамок и границ** от первоначального замысла.
* **Проведение аудита**

Аудит - это последний шаг в процессе реализации информационной безопасности. После определения состояния информационной безопасности внутри организации, создания соответствующих политик и процедур, приведения в действие технических средств контроля и обучения персонала проведение аудита позволит удостовериться, что все средства контроля сконфигурированы правильно.

Обсуждая место аудита в процессе безопасности, мы в действительности говорим о трех разных функциях:

* аудит соблюдения политики;
* периодическая оценка существующих проектов и оценка новых проектов;
* проверка возможности нарушения защиты.

Каждая из этих функций занимает свое место в процессе обеспечения безопасности.

**Аудит соблюдения политики**

Аудит соблюдения политики - это традиционная функция аудита. Организация имеет политику, определяющую настройки и конфигурацию систем безопасности. Аудит определяет реальное состояние дел. Любые отклонения отмечаются как нарушения. Подобные проверки могут выполняться внутренним персоналом или внешними консультантами. И в том и в другом случае этот процесс требует участия системных администраторов.

Аудит соблюдения политики не должен ограничиваться только проверкой конфигурации систем. Он должен проявлять интерес к тому, как выполняются другие формы управления информацией. Соблюдается ли информационная политика? Как хранятся и передаются секретные документы?

Проверки должны проводиться раз в год. Они могут выполняться персоналом отдела безопасности, но, возможно, выполнение аудита больше подходит для отдела аудита организации или для сторонней фирмы. Причина в том, что в данном случае могут быть затронуты интересы самого отдела безопасности, что приведет к возникновению конфликта интересов.

**Периодическая оценка проектов и оценка новых проектов**

Компьютерная и сетевая среда внутри организации находятся в состоянии постоянного изменения. Эти изменения приводят к быстрому старению результатов оценки за счет сокращения некоторых рисков и введения новых. По этой причине оценка должна выполняться периодически. Полная оценка организации должна выполняться раз в два года. Как и в случае с крупными проверками, серьезные оценки выполняются персоналом отдела безопасности, если он обладает необходимыми навыками. Возможно, для этих целей больше подходит сторонняя организация.

Небольшие оценки должны выполняться в случае разработки новых проектов или изменений в организационной среде. Для каждого нового проекта отдел безопасности привлекается к работе на стадии проектирования, чтобы определить, имеет ли проект какие-либо риски, и происходит ли в результате разработки проекта появление или сокращение рисков внутри организации. Этот тип оценки должен изучать новый проект в контексте его использования по отношению к другим структурным элементам организации. Если риски определены на ранней стадии проекта, проектирование может быть скорректировано или введены другие механизмы для управления риском.

**Проверка возможности нарушения защиты**

Проверка возможности нарушения защиты - это спорная тема. Часто такие проверки выполняются вместо оценки. На самом деле, они имеют ограниченную ценность в программе безопасности. Причина этого проста: при проверках предпринимаются попытки воспользоваться установленной уязвимостью, чтобы получить доступ к системам и информации внутри организации. Если такая проверка имеет успех, то единственный вывод из всего этого - обнаружена, по крайней мере, одна уязвимость. Если проверка нарушения защиты терпит неудачу, то вывод такой - проверяющий не смог обнаружить и использовать уязвимость. Это вовсе не значит, что уязвимости не существует.

Почему же тогда необходимо выполнять проверку возможности нарушения защиты? Если организация провела оценку и применила подходящие средства управления риском, она может выборочно проверить некоторых из них. Проверка защиты подходит для следующих случаев.

Способность системы обнаружения вторжений выявить попытку нарушения защиты.

Уместность процедуры реагирования на инцидент, связанный с безопасностью.

Информация о сети, которую можно узнать через средства управления сетевым доступом.

Уместность физической безопасности помещения.

Адекватность информации, предоставляемой сотрудникам программой повышения осведомленности в плане безопасности.

**Основные этапы проведения аудита**

В обобщенном виде ИТ-аудит проводится в два этапа:

* этап "Планирование ИТ-аудита".
* этап "Проведение ИТ-аудита".

На этапе "Планирования ИТ-аудита":

* Анализируются:
  + структура бизнес-процессов;
  + платформы и структура информационных систем, поддерживающих бизнес-процессы;
  + структура ролей и распределения ответственности, включая аутсорсинг;
  + бизнес-риски и бизнес-стратегия.
* Определяются информационные критерии, наиболее значимые для существующих бизнес-процессов.
* Идентифицируются ИТ-риски.
* Оценивается общий уровень контроля рассматриваемых бизнес-процессов.
* На основе полученной информации осуществляется выбор границ и объектов исследования: ИТ-процессов и связанных с ними ИТ-ресурсов.

На этапе "Проведения ИТ-аудита" выполняются следующие виды работ:

* Идентификация существующих механизмов управления и документирование процедур (сбор и первичный анализ информации);
* Оценка эффективности существующих механизмов управления при выполнении задач управления, их целесообразность и пригодность;
* Тест соответствия (получение гарантий пригодности существующих механизмов управления для решения задач управления);
* Детальное тестирование с целью выполнения корректирующих действий для улучшения состояния системы управления ИТ.

**Результаты проведения ИТ-аудита**

Результаты ИТ-аудита компании классифицируются на три группы:

* Организационные – планирование, управление, документооборот функционирования ИС.
* Технические – сбои, неисправности, оптимизация работы элементов ИС, непрерывное обслуживание, создание инфраструктуры и т.д
* Методологические – подходы к решению проблемных ситуаций, управлению и контролю, общая упорядоченность и структуризация.

Проведенный ИТ-аудит позволит обоснованно создать следующие документы:

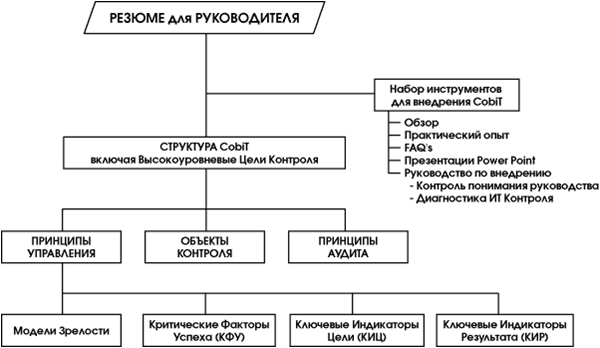
* **Основные документы:**
* Отчет о результатах ИТ-аудита компании.
* Отчет о результатах аудита информационной безопасности компании.
* **Дополнительные документы** (документы, которые могут быть разработаны по согласованию сторон в дополнение к основным):
* Долгосрочный план развития ИТ/ИС;
* Краткосрочный план развития ИТ/ИС;
* Отчет о текущем состоянии ИТ/ИС.
* Техническое задание на изменение ИТ/ИС
* Методология работы и настройки (доводки) ИТ/ИС компании.
* Концепция построения политики безопасности ИТ/ИС компании.
* Политика безопасности ИТ/ИС компании.
* План восстановления ИТ/ИС в чрезвычайной ситуации.
* Порядок действий в случае нарушения защиты информации.
* План-график проведения последующих ИТ-аудитов.

Результатные документы могут иметь сокращенную и полную редакции. Документ в сокращенной редакции – это документ, содержащий основные итоги и рекомедации по тематике ИТ-аудита и предназначенный для топ-менеджеров компании. Документ в полной редакции – это документ с подробным изложением материала, предназначенный для менеджеров среднего звена и ИТ-спецалистов.

**Стандарты ИТ-аудита**

Вопросу аудита и внутреннего контроля за информационными системами посвящены несколько зарубежных стандартов аудита и специальный российский стандарт «Аудит в условиях компьютерной обработки данных (КОД)». Из зарубежных источников можно отметить международный стандарт аудита CobiT, ISA 401, положения по международной практике аудита 1002, 1003,1004, 1008, 1009. В них отражены вопросы практики аудита в среде компьютерных информационных систем, оценки рисков и надежности системы внутреннего контроля в условиях КОД, техника проведения аудита с учетом использования современных информационных технологий.

**9.Стандарт CobiT и его роль в управлении и аудите ИТ.**

Аббревиатура CobiT расшифровывается как Контрольные ОБъекты для Информационных и смежных Технологий. За этой аббревиатурой скрывается набор документов, в которых изложены принципы управления и аудита информационных технологий. CobiT позиционируется как открытый стандарт "де-факто", в настоящее время переживающий свое третье издание.

В состав стандарта входят шесть книг, ориентированных на разные аудитории:

Резюме для руководителя. Описание стандарта CobiT, ориентированное на топ-менеджеров организации для принятия ими решения о применимости стандарта в конкретной организации.

Описание структуры. Книга содержит развернутое описание структуры стандарта, высокоуровневых целей контроля и пояснения к ним, необходимые для эффективной навигации и результативной работы со стандартом.

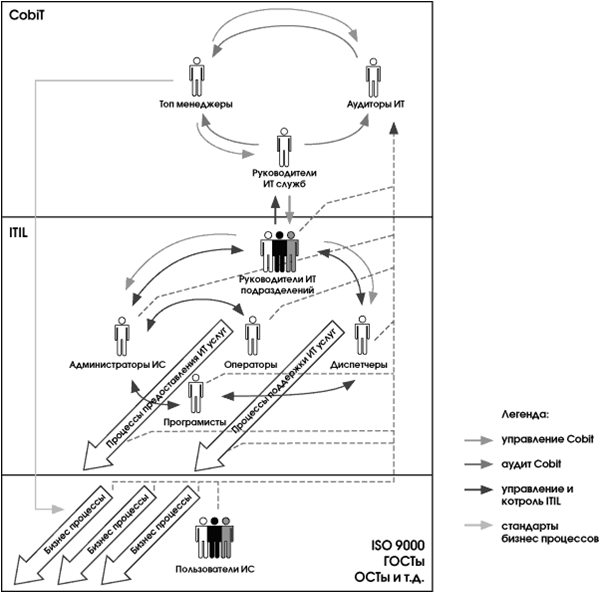
Объекты контроля. В книгу включены детальные описания объектов контроля, содержащие расшифровку каждого из объектов.

Принципы управления. Книга отвечает на вопросы как управлять ИТ, как правильно поставить достижимую цель, как ее достичь и как проконтролировать полноту ее достижения. Предназначена для руководителей ИТ-служб.

Принципы аудита. Правила проведения ИТ-аудита. Описание того, у кого можно получить необходимую информацию, как ее проверить, какие вопросы задавать? Книга предназначена для внутренних и внешних аудиторов ИТ, а также консультантов в сфере ИТ.

Набор инструментов внедрения стандарта — практические советы по ежедневному использованию стандарта в управлении и аудите ИТ. Книга предназначена для внутренних и внешних аудиторов ИТ, консультантов в сфере ИТ.

**Рисунок 1. Состав книг CobiT**

Модель процессов, выстраиваемая на базе CobiT, предпочтительней других подходов, в основе которых не лежат бизнес-процессы организации (методики и стандарты аудита производителей программно-аппаратных средств), по нескольким причинам:

По определению: процесс — это действие, направленное на достижение результата, при оптимальном использовании ресурсов, и которое может корректироваться при его выполнении. При выполнении процесса все задействованные ресурсы структурируются и выстраиваются таким образом, чтобы максимально эффективно выполнять этот процесс.

Во-вторых, процессы в подавляющем большинстве организаций, а особенно их цели не так часто изменяются, по сравнению с организационными объектами (организационно-штатная структура: сотрудники, отделы, департаменты и т.д.).

В-третьих, развертывание информационной системы или внедрение информационных технологий не может быть ограничено спецификой одного отдела или департамента, а затрагивает руководителей, пользователей из других подразделений и ИТ-специалистов. Таким образом, прикладные системы (прикладное программное обеспечение то, что видит пользователь) — это неотъемлемая часть структуры CobiT и могут быть стандартно оценены, как и прочие объекты контроля CobiT, в рамках единой структуры и с применением единых метрик.

CobiT — это сохранение единого подхода к сбору, анализу информации, подготовке выводов и заключений на всех этапах управления, контроля и аудита ИТ, возможность сравнения существующих ИТ-процессов с "лучшими" практиками, в том числе отраслевыми.

Как следует из рисунка, CobiT предоставляет топ-менеджерам возможность донести цели и задачи бизнеса до руководителей ИТ-служб, преобразовав стратегические и тактические планы организации в четкие и понятные планы развития ИТ. Руководители ИТ-служб, в свою очередь, управляют руководителями подразделений на основе полученных указаний в соответствии с CobiT.

Методология ITIL применяется для оптимизации процесса обслуживания информационных систем с точки зрения управления. Если процессы предоставления и поддержки ИТ услуг (ITIL) в организации не внедрены, то Cobit предоставляет механизмы управления и на этом уровне. При этом CobiT можно применить в части управления эксплуатацией информационной системой, но только в качестве инструмента общего управления и контроля. Здесь необходимо учитывать, что CobiT не предоставляет инструментов для управления или аудита аппаратно-программного обеспечения конкретных фирм производителей. Так, например, аудит программного обеспечения Microsoft должен выполняться в соответствии с методиками, разработанными Microsoft, и на соответствие требованиям Microsoft, но результаты подобной проверки могут и должны быть использованы в процессах CobiT. Так очень схематично можно определить место и роль CobiT в части управления ИТ.

Итак, ИТ-аудиторы собирают, анализируют информацию и предоставляют отчеты и заключения руководителям организации в соответствии с руководством аудитора CobiT (объекты контроля, "размещенные" в соответствии с рекомендациями CobiT). Основываясь на аудиторских оценках и заключениях о степени достижения целей управления, руководители организации обоснованно, с точки зрения соответствия ИТ целям и задачам бизнеса, осуществляют управление процессами организации для реализации ее бизнес-целей.

**10.Сертификация и лицензирование в проекте разработки программного обеспечения. Ответственность за использование нелицензионного программного обеспечения**

**Сертифика́ция услуг (работ)** — это независимое подтверждение соответствия утвержденным требованиям с целью соблюдения «Закона о защите прав потребителей» поставщиком работ и услуг на территории Российской Федерации.

Правила сертификации услуг (работ) регламентируются Постановлением Госстандарта РФ от 05.08.1997 N 17 «О принятии и введении в действие Правил сертификации»[[3]](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F#cite_note-2).

**В качестве требований при сертификации услуг (работ) могут являться:**

* законодательные акты Российской Федерации;
* правила выполнения отдельных видов работ и оказания отдельных видов услуг, утвержденные постановлениями Правительства Российской Федерации;
* государственные стандарты, санитарные правила и нормы, строительные нормы и правила и другие документы, которые в соответствии с законами Российской Федерации устанавливают обязательные требования к работам и услугам.

**Сертификация программного обеспечения** подтверждает его соответствие каким либо требованиям. Для программного обеспечения основными требованиями является

* безопасность работы с документами;
* невозможность случайной утечки обрабатываемой информации;
* отсутствие так называемых “закладок” – скрытых функций программного обеспечения, выполняющих недокументированные функции – например, слежения за пользователем, передачи секретной информации (ключей шифрования и тд).

Практически каждая страна имеет собственные центры сертификации. Сертификация программного обеспечения в России осуществляется Федеральной службой технического и экспортного контроля (бывшая Гостехкомиссия России) и ФСБ.

Использование сертифицированных продуктов дает следующие преимущества:

* При штатном использовании сертифицированных продуктов нет необходимости использовать дополнительные средства защиты обрабатываемой информации, что повышает скорость и надежность работы продуктов, снижает стоимость используемых решений и уровень требований к квалификации системного администратора;
* Сертифицированное ПО соответствует требованиям нормативных документов, регламентирующих применение защищенных автоматизированных систем и может быть использовано для их построения.

Сертификация программного обеспечения обеспечивает обновления защищенных продуктов, что дает возможность поддержания высокого уровня защищенности работающих систем.

Процесс сертификации заключается не в сертификации единичного образца или партии программного продукта – реализовано аттестованное производство сертифицированных версий программных продуктов, включая выпуск физических носителей, тестирование их параметров и маркировку каждого носителя штрих-кодом

### Лицензирование программного обеспечения

Программное обеспечение (ПО) – неотъемлемая часть вычислительной техники. Как продукт интеллектуальной собственности, ПО имеет защиту. Распространяется эта защита на законность копирования, изменения программного продукта и оформляется в виде лицензии. Получение лицензии означает согласие со всеми пунктами использования, прописанными в договоре. Любое несанкционированное (без согласования с его автором) использование ПО – безусловно, деяние противоправное.

В российском законодательстве термин «лицензия» трактуется двояко:

* Разрешение государства, выданное уполномоченным органом  на определенный вид деятельности. Список сфер деятельности, подлежащих безусловному лицензированию, устанавливается законодательно.
* Разрешение автора или издателя, обладающего исключительными правами на предмет интеллектуальной собственности (товарный знак, кинокартину, программу для ЭВМ), даваемое другому лицу для использования оговариваемым образом.

Разрешительные документы выдаются госорганами на основании ФЗ «О лицензировании отдельных видов деятельности».

Лицензии-разрешения на использование баз данных, ПО для ЭВМ, как правило, заключаются как двустороннее соглашение. Этот документ называется лицензионный договор – **лицензия на программное обеспечение**.

По договору основной правообладатель передает пользователю право использовать объект договора с оговоренными ограничениями или без оных. Объем полномочий покупателя определяется соглашением сторон. Но правообладатель может передать не больше прав, чем имеет.

Современное законодательство разработало разные виды лицензий.

1. Передаются по соглашению исключительные или неисключительные права. Передача неисключительных прав означает, что правообладатель продолжает использовать интеллектуальную собственность в прежнем объёме. Пользователь получает с правообладателем равные права. Правообладатель может в дальнейшем неоднократно передавать такие же права неограниченному кругу лиц.

Передача исключительных прав на ПО приводит к приобретению пользователем исключительных прав на интеллектуальную собственность. Сам правообладатель не может уже ни самостоятельно осуществлять переданные правомочия, ни передавать их полномочия другим лицам.

2. Другое деление видов лицензий применимо лишь к лицензиям ПО для ЭВМ: программы могут распространяться как OpenWare – на условиях открытого кода либо без оного условия.

При варианте распространения на условиях открытого кода, пользователь получает право использовать ПО, а также исходные коды программ с полномочиями модификации исходных текстов по своему усмотрению, изменяя в любой степени.

В дальнейшем модифицированный программный продукт может быть использован в зависимости от лицензии. Сейчас разработано два рода лицензий, оговаривающих детали условий передачи ПО с открытым кодом. Это GNU GPL и FreeBSD. Коренная разница между ними в преемственности свойств открытого кода: по лицензии GNU GPL, любые программные продукты, разработанные изменением полученного на этих условиях программного кода, далее могут быть распространены лишь в границах лицензии GNU GPL.

Лицензия Free BSD даёт обладателю ПО гораздо больше свободы: модифицированные им программы могут далее распространяться на других, предусмотренных автором изменений условиях, как безвозмездно, так и за плату.

Следует отметить, что есть и другие виды лицензий, содержащих условие передачи исходного кода правообладателем с разной величиной прав пользователя.

3. Часто **лицензирование программного обеспечения** классифицируется по возмездности. Так ПО разделяется на бесплатное, условно-бесплатное и коммерческое. Надо учесть, что бесплатность использования программ не означает полномочий на получение исходных кодов. Зачастую бесплатность означает и отсутствие гарантий должного функционирования ПО.

Условно-бесплатная **лицензия на программное обеспечение** означает временные или функциональные ограничения ПО, снять которые можно, оплатив полную версию.

При платном распространении ПО обычно пользователь получает некоторые гарантии производителя и его обязательства по сопровождению продукта.

4. **Лицензирование программного обеспечения** можно ещё разделить по пользователю и по цели договора. Тут выделяются соглашения EULA – с конечным пользователем, и те лицензии, что предусматривают доработку приобретателем программного обеспечения.

Следует отметить, что стандартной формы лицензионного соглашения EULA пока не выработано. Как правило, мелкие фирмы ориентируются на текст лицензионных соглашений крупных компаний, например, Microsoft.

5. При предоставлении прав на ПО конечному пользователю в ряде случаев говорится о OEM/BOX – «связанных» лицензиях и «независимо предоставляемых».

При OEM-поставках с ЭВМ приобретается пакет предустановленного ПО. Такие лицензии значительно выгоднее финансово, но предполагают меньший размер полномочий пользователя.

Коробочная (BOX) поставка включает ПО на носителе и документацию на него. Покупатель получает несколько больший объем правомочий. Часто BOX-поставка использует «оберточную лицензию»: вскрытая коробка означает признание лицензионного соглашения (нанесённого на обертку).

**Лицензирование программного обеспечения** – залог успешности производителя. Грамотный выбор типа лицензии позволяет разработчику или продавцу защитить интеллектуальную собственность и оптимизировать бизнес.

**Ответственность**

Использование нелицензионного программного обеспечения является нарушением авторских и смежных прав и влечет за собой административную (ст. 7.12. КоАП РФ) и уголовную (ст. 146 УК РФ) ответственность.

Административная ответственность предусматривает наложение штрафа с конфискацией нелицензионных экземпляров программного обеспечения, а также оборудования, используемого для их воспроизведения.

Уголовная ответственность наступает, если стоимость использованного нелицензионного программного обеспечения или прав на него превышает 50000 рублей. В этом случае предусмотрен штраф в размере до двухсот тысяч рублей или в размере заработной платы или иного дохода за период до восемнадцати месяцев, либо обязательные работы на срок от ста восьмидесяти до двухсот сорока часов, либо лишение свободы на срок до двух лет.

Если же будет доказано, что использование нелицензионного программного обеспечения было совершено по предварительному сговору или организованной группой, или в особо крупном размере, то есть стоимость программного обеспечения или прав на него превышает 250000 рублей, или лицом с использованием своего служебного положения, то виновные лица наказываются лишением свободы на срок до шести лет со штрафом в размере до пятисот тысяч рублей или в размере заработной платы или иного дохода за период до трех лет либо без такового.

**Российское законодательство не делает различий в зависимости от организационно-правовой формы юридического лица. То есть, организация (и/или ее руководитель) независимо от того, является ли она коммерческой или некоммерческой, будет нести административную, гражданско-правовую или уголовную ответственность за несоблюдение законов.**

Как свидетельствует судебная практика, по общему правилу ответственность за использование нелицензионного программного обеспечения (ПО) лежит на генеральном директоре (или управляющей организации) общества в силу того, что именно эти органы принимают решения о выделении денежных средств на покупку ПО. Однако возможна ситуация возложения ответственности за использование нелицензионного ПО на системного администратора (вследствие ненадлежащего исполнения возложенных на него трудовых обязанностей, которое повлекло нарушение норм действующего законодательства).