Федеральное агентство по образованию
Нижегородский государственный университет им. Н.И. Лобачевского

Национальный проект «Образование»
Инновационная образовательная программа ННГУ. Образовательно-научный центр «Информационно-телекоммуникационные системы: физические основы и математическое обеспечение»

С.Н. Карпенко

Введение в программную инженерию

Учебно-методические материалы по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике»

Нижний Новгород
2007

Учебно-методические материалы подготовлены в рамках
инновационной образовательной программы ННГУ: Образовательно-научный центр «Информационно-телекоммуникационные системы: физические основы и математическое обеспечение»

Карпенко С.Н. Введение в программную инженерию. Учебно-методические материалы по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике». Нижний Новгород, 2007, 103 с.

Предметом программной инженерии является круг вопросов и проблем, возникающих при промышленной разработке программных продуктов. Особенность такой разработки связана с коммерческим характером разрабатываемых программ, их конструктивной сложностью, коллективным характером работы и рядом других специфических характеристик.
В материалах дается понятие программной инженерии, ее методические основы и принципы; описывается структура и организация жизненного цикла программного продукта и характеристики основных моделей жизненного цикла; рассматриваются общие вопросы управления программным проектом, принципы формирования и управления командой разработчиков, основы планирования проекта; приводятся основы методологии управления качеством ИТ процесса.

ОГЛАВЛЕНИЕ

ПОНЯТИЕ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Программная инженерия (промышленное программирование) обычно ассоциируется с разработкой больших и сложных программ коллективами разработчиков. Становление и развитие этой области деятельности было вызвано рядом проблем, связанных с высокой стоимостью программного обеспечения, сложностью его создания, необходимостью управления и прогнозирования процессов разработки.

В конце 60-х – начале 70-х годов прошлого века произошло событие, которое вошло в историю как первый кризис программирования. Событие состояло в том, что стоимость программного обеспечения стала приближаться к стоимости аппаратуры («железа»), а динамика роста этих стоимостей позволяла прогнозировать, что к середине 90-годов все человечество будет заниматься разработкой программ для компьютеров. Тогда и заговорили о программной инженерии (или технологии промышленного программирования, как это называлось в России) как о некоторой дисциплине, целью которой является сокращение стоимости программ.

С тех пор программная инженерия прошла достаточно бурное развитие. Этапы развития программной инженерии можно выделять по-разному. Каждый этап связан с появлением (или осознанием) очередной проблемы и нахождением путей и способов решения этой проблемы

Сам термин – software engineering (программная инженерия) - впервые был озвучен в октябре 1968 года на конференции подкомитета НАТО по науке и технике (г. Гармиш, Германия). Присутствовало 50 профессиональных разработчиков ПО из 11 стран. Рассматривались проблемы проектирования, разработки, распространения и поддержки программ. Там впервые и прозвучал термин «программная инженерия» как некоторая дисциплина, которую надо создавать и которой надо руководствоваться в решении перечисленных проблем.

Вскоре после этого в Лондоне состоялась встреча 22-х руководителей проектов по разработке ПО. На встрече анализировались проблемы и перспективы развития ПО. Отмечалась возрастающее воздействие ПО на жизнь людей. Впервые серьезно заговорили о надвигающемся кризисе ПО. Применяющиеся принципы и методы разработки ПО требовали постоянного усовершенствования. Именно на этой встрече была предложена концепция жизненного цикла ПО (SLC – Software Lifetime Cycle) как последовательности шагов-стадий, которые необходимо выполнить в процессе создания и эксплуатации ПО. Вокруг этой концепции было много споров. В 1970 г. У.У. Ройс (W.W. Royce) произвел

идентификацию нескольких стадий в типичном цикле и было высказано предположение, что контроль выполнения стадий приведет к повышению качества ПО и сокращению стоимости разработки.

Предпосылки и история

Повторное использование кода (модульное программирование)

Проблема. На первых этапах становления программной инженерии (даже когда она так еще не называлась) было отмечено, что высокая стоимость программ связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах. Вызвано это было тем, что в различных программах как части этих программ решались одинаковые (или похожие) задачи: решение нелинейных уравнений, расчет заработной платы, … Использование при создании новых программ ранее написанных фрагментов сулило существенное снижение сроков и стоимости разработки.

Модульное программирование. Главный принцип модульного программирования состоял в выделении таких фрагментов и оформлении их в виде модулей. Каждый модуль снабжался описанием, в котором устанавливались правила его использования – интерфейс модуля. Интерфейс задавал связи модуля с основной программой – связи по данным и связи по управлению. При этом возможность повторного использования модулей определялась количеством и сложностью этих связей, или насколько эти связи удалось согласовывать с организацией данных и управления основной программы. Наиболее простыми в этом отношении оказались модули решения математических задач: решения уравнений, систем уравнений, задач оптимизации. К настоящему времени накоплены и успешно используются большие библиотеки таких модулей.

Для многих других типов модулей возможность их повторного использования оказалась проблематичной в виду сложности их связей с основной программой. Например, модуль расчета зарплаты, написанный для одной фирмы, может не подойти для другой, т.к. зарплата в этих фирмах рассчитывается не во всем одинаково. Повторное использование модулей со сложными интерфейсами является достаточно актуальной и по сей день. Для ее решения разрабатываются специальные формы (структуры) представления модулей и организации их интерфейсов.

Рост сложности программ (структурное программирование)

Проблема. Следующий этап возрастания стоимости ПО был связан с переходом от разработки относительно простых программ к разработке сложных программных

комплексов. Следует отметить, что этот переход был вызван появлением вычислительной техники третьего поколения (интегральные схемы). С переходом на использование интегральных схем производительность компьютеров возросла на порядки, что и создало предпосылки для решения сложных задач. К числу таких сложных задач относятся: системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д. Сложность таких комплексов оценивалась следующими показателями:

- Большой объем кода (миллионы строк)
- Большое количество связей между элементами кода
- Большое количество разработчиков (сотни человек)
- Большое количество пользователей (сотни и тысячи)
- Длительное время использования

Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию. По аналогии с промышленной технологией стали говорить о жизненном цикле программного продукта, как о последовательности определенных этапов: этапа проектирования, разработки, тестирования, внедрения и сопровождения.

Структурное программирование. Этап сопровождения программного комплекса включал действия по исправлению ошибок в работе программы и внесению изменений в соответствии с изменившимися требованиями пользователей. Основная причина высокой стоимости (а порой и невозможности выполнения) этапа сопровождения состояла в том, что программы были плохо спроектированы – документация была не понятна и не соответствовала программному коду, а сам программный код был очень сложен и запутан. Нужна была технология, которая обеспечит «правильное» проектирование и кодирование. Основные принципы технологии структурного проектирования и кодирования:

- Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т.д. (принцип «разделяй и властвую»)
- Применение специальных языков проектирования и средств автоматизации использования этих языков
- Дисциплина проектирования и разработки: планирование и документирование проекта, поддержка соответствие кода проектной документации
- Структурное кодирование без goto

Модификация программ (ООП)

Проблема. Следующая проблема роста стоимости программ была вызвана тем, что изменение требований к программе стали возникать не только на стадии сопровождения, но и на стадии проектирования – проблема заказчика, который не знает, что он хочет. Создание программного продукта превратилось в его перманентное перепроектирование. Возник вопрос, как проектировать и писать программы, чтобы обеспечить возможность внесений изменений в программу, не меняя ранее написанного кода.

Объектно-ориентированное программирование. Решением этой проблемы стало использование подхода или метода, который стали называть объектно-ориентированным проектированием и программированием. Суть подхода состоит в том, что вводится понятие класса как развитие понятия модуля с определенными свойствами и поведением, характеризующими обязанностями класса. Каждый класс может порождать объекты – экземпляры данного класса. При этом работают основные принципы (парадигмы) ООП:

- Инкапсуляция – объединение в классе данных (свойств) и методов (процедур обработки).
- Наследование – возможность вывода нового класса из старого с частичным изменением свойств и методов
- Полиморфизм – определение свойств и методов объекта по контексту

Некоторые итоги

Программная инженерия (или технология промышленного программирования) как некоторое направление возникло и формировалось под давлением роста стоимости создаваемого программного обеспечения. Главная цель этой области знаний - сокращение стоимости и сроков разработки программ.

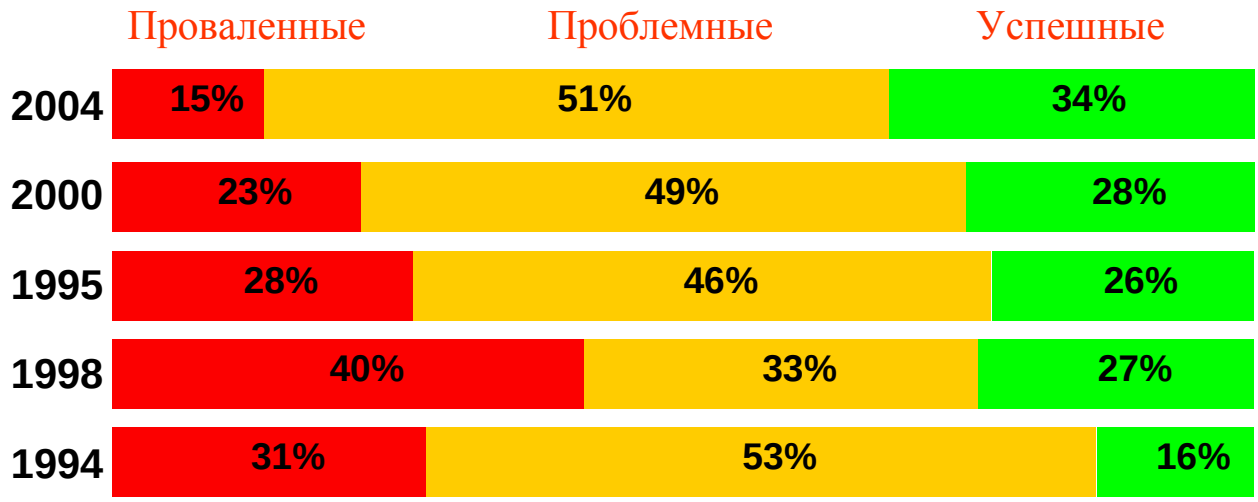
Программная инженерия прошла несколько этапов развития, в процессе которых были сформулированы фундаментальные принципы и методы разработки программных продуктов. Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта. Фундаментальными методами проектирования и разработки являются модульное, структурное и объектно-ориентированное проектирование и программирование.

Продолжение кризиса программирования

Несмотря на то, что программная инженерия достигла определенных успехов, перманентный кризис программирования продолжается. Связано это с тем, рубеж 80–90-х годов отмечается как начало информационно-технологической революции, вызванной взрывным ростом использования информационных средств: персональный компьютер, локальные и глобальные вычислительные сети, мобильная связь, электронная почту, Internet и т.д.

Цена успеха – кризис программирования принимает хронические формы:

- США тратит ежегодно более $200 млрд. на более чем 170 тыс. проектов разработки ПО в сфере IT;
- 31,1% из них закрываются, так и не завершившись; 52,7% проектов завершаются с превышением первоначальных оценок бюджета/сроков и ограниченной функциональностью;
- потери от недополученного эффекта внедрения ПО измеряются триллионами.



Представленная на схеме статистика по 30,000 проектам по разработке ПО в американских компаниях [1] показывает следующее распределение между:

- *У*спешными проектами – вовремя и в рамках бюджета был выполнен весь намеченный фронт работ
- Проблемными проектами – нарушение сроков, перерасход бюджета и/или сделали не все, что требовалось
- Проваленными проектами – не были доведены до конца из-за перерасхода средств, бюджета, качества.

Программная инженерия – что это такое?

На сегодняшний день нет единого определения понятия «программная инженерия». Ниже приведено несколько таких определений, данных крупными специалистами в этой области, или зафиксированные в документах ведущих организаций [2]:

- установление и использование обоснованных инженерных принципов (методов) для экономного получения ПО, которое надежно и работает на реальных машинах. [Bauer 1972].
- та форма инженерии, которая применяет принципы информатики (computer science) и математики для рентабельного решения проблем ПО. [CMU/SEI-90-TR-003]
- применение систематического, дисциплинированного, измеряемого подхода к разработке, использованию и сопровождению ПО [IEEE 1990].
- дисциплина, целью которой является создание качественного ПО, которое завершается вовремя, не превышает выделенных бюджетных средств и удовлетворяет выдвигаемым требованиям [Schach, 99]

Для того, чтобы получить представление о том, что такое программная инженерия, следуя [3] попробуем разобраться в следующих вопросах:

- Что такое программное обеспечение (software)?
- Что такое программная инженерия?
- В чем отличие программной инженерии от информатики (computer science)?
- В чем отличие программной инженерии от других инженерий?
- Что такое методы программной инженерии?
- Что такое CASE (Computer-Aided Software Engineering)?
- Какими свойствами обладает хорошая программа?

Software - программное обеспечение или программный продукт?

В [4] программное обеспечение определяется как набор компьютерных программ, процедур и связанной с ними документации и данных. Взгляд на ПО как только на программу, сидящую в компьютере слишком узок. Дело в том, что продается (поставляется) не только программа, но еще и документация, в которой можно прочитать как установить программу и как ей пользоваться и данные для установки программы в различных условиях (конфигурационные файлы). Поэтому ПО иногда называют программным продуктом. Т.е. программный продукт (программное обеспечение) – это не

только программы, а также вся связанная с ними документация и конфигурационные данные, необходимые для корректной работы программы. А специалисты по программному обеспечению разрабатывают программные продукты, т.е. такое ПО, которое может быть продано потребителю.

В зависимости от того, для кого разрабатываются программные продукты (конкретного заказчика или рынка, программные продукты бывают двух типов:

- коробочные продукты (generic products – общие продукты или shrink-wrapped software – упакованное ПО)
- заказные продукты (bespoke – сделанный на заказ или customized products – настроенный продукт). Важная разница между ними заключается в том, кто ставит задачу (определяет, или специфицирует требования). В первом случае это делают сами разработчики на основе анализа рынка (маркетинга) – и при этом рискуют сами. Во втором – заказчик и при этом рискует, что разработчик не сможет реально выполнить все требования в срок и при выделенном бюджете.

Что такое программная инженерия?

Программная инженерия — это инженерная дисциплина, которая связана со всеми аспектами производства ПО от начальных стадий создания спецификации до поддержки системы после сдачи в эксплуатацию. В этом определении есть две ключевые фразы:

- Инженерная дисциплина
- Все аспекты производства ПО

Инженерная дисциплина. Инженеры – это те специалисты, которые выполняют практическую работу и добиваются практических результатов. Ученый может сказать: проблема неразрешима в рамках существующих теорий и это будет научный результат, достойный опубликования и защиты диссертации.

Для решения задачи инженеры применяют теории, методы и средства, пригодные для решения данной задачи, но они применяют их выборочно и всегда пытаются найти решения, даже в тех случаях, когда теорий или методов, соответствующих данной задаче, еще не существует. В этом случае инженер ищет метод или средство для решения задачи, применяет его и несет ответственность за результат – ведь метод или средство еще не проверены. Набор таких инженерных методов или способов, теоретически возможно не обоснованных, но получивших неоднократное подтверждение на практике, играет большую практическую роль. В программной инженерии они получили название лучших практик (best practices).

Инженеры работают в условиях ограниченных ресурсов: временных, финансовых и организационных (оборудование, техника, люди). Иными словами, продукт должен быть создан в установленные сроки, в рамках выделенных средств, оборудования и людей. Хотя это в первую очередь относится к созданию заказных продуктов (оговаривается в условиях контракта), но при создании коробочных продуктов эти ограничения имеют не меньшее значение, т.к. здесь они диктуются условиями рыночной конкуренции.

Все аспекты производства ПО. Программная инженерия занимается не только техническими вопросами производства ПО (специфицирование требований, проектирование, кодирование,…), но и управлением программными проектами, включая вопросы планирования, финансирования, управления коллективом и т.д. Кроме того, задачей программной инженерии является разработка средств, методов и теорий для поддержки процесса производства ПО.

Программные инженеры применяют систематичные и организованные подходы к работе для достижения максимальной эффективности и качества ПО. Их задача состоит в адаптации существующих методов и подходов к решению свой конкретной проблемы.

В чем отличия от информатики?

Информатика (computer science) занимается теорией и методами вычислительных и программных систем, в то время как программная инженерия занимается практическими проблемами создания ПО. Информатика составляет теоретические основы программной инженерии и инженер по программному обеспечению должен знать информатику. Так же, как инженер по электронике должен знать физику. В идеале, программная инженерия должна быть поддержана какими-то теориями информатики, но самом деле это не всегда так. Программные инженеры зачастую используют приемы, которые применимы только в конкретных условиях и не могут быть обобщены на другие случаи, а элегантные теории информатики не всегда могут быть применены к реальным большим системам.

И наконец, информатика – это не единственный теоретический фундамент программной инженерии, т.к. круг проблем, стоящих перед программным инженером значительно шире просто написания программ. Это еще управление финансами, организация работ в коллективе, взаимодействие с заказчиком и т.д. Решение этих проблем требуют фундаментальных знаний, выходящих за рамки информатики.

В чем отличие от других инженерий?

Отличие программной инженерии от других инженерий интересно прежде всего с точки зрения двух вопросов:

- Почему доля провальных проектов в программной инженерии так велика по сравнению с другими инженериями?
- Можно ли в программной инженерии применять опыт других инженерий?

Эти вопросы является фундаментальными для программной инженерии. По этому поводу высказывается много мнений (и часто противоположных). Остановимся на некоторых более или менее очевидных отличиях программной инженерии от других инженерий.

Прежде всего, отметим, что жизненный цикл продукта любой инженерии в упрощенном виде включает фазы: проектирование, создание образца, испытание, производство, эксплуатация.

Компьютерная программа – это (в отличие от объектов других инженерий) не материальный объект (просьба не путать с носителем программы – устройством памяти любого типа). Отсюда следуют следующие отличия. Фаза производства состоит в копировании образца на другие носители. Стоимость фазы исчезающее мала. Если кодирование считать элементом проектирования (что очень близко к истине), то отсутствует также и фаза создания образца (строится компилятором и линковщиком)

Отсюда следуют следующие выводы:

- Стоимость программы – это стоимость только ее проектирования
- Стоимость проектирования коробочных продуктов «размазывается» по копиям
- Стоимость заказных продуктов (массово не копируемых) остается высокой

Второе существенное отличие состоит в том, что программа – искусственный объект. Т.е. для программы нет объективных законов, которым бы подчинялось ее поведение. Например, у инженера – строителя есть объективные законы строительной механики: равновесия моментов и сил, устойчивости механических систем и т.д. Инженер – строитель может проверить свои архитектурные решения на соответствие этим законам и тем самым обеспечить удачу проекта. Эти законы объективны, они будут действовать всегда. У программного инженера на первый взгляд также есть типовые, проверенные временем архитектурные решения (например, клиент-серверная архитектура). Но эти решения определяются уровнем развития вычислительной техники (и адекватным им уровнем требований). С появлением техники с принципиально новыми возможностями программному инженеру придется искать новые решения.

Прямым следствием отсутствия возможности «теоретического» контроля проекта является то, что тестирование продукта – это единственный способ убедиться в его

качестве. Именно поэтому стоимость тестирования составляет существенную стоимость ПО. Кстати, строительный инженер, как правило, лишен возможности такого «тестирования» своего продукта перед сдачей его в эксплуатацию.

Ну и наконец, программная инженерия – молодая дисциплина, опыт которой насчитывает всего несколько десятков лет. По сравнению с опытом строительной инженерии (тысячелетия) это конечно очень мало. Программную инженерию иногда сравнивают с ранней строительной, когда законы строительной механики еще не были известны и строительные инженеры действовали методом проб и ошибок, накапливая бесценный опыт. Несмотря на молодой возраст, программная инженерия также накопила определенный опыт, который позволяет (при разумном его применении) делать удачные проекты. Этот опыт выражен в основных принципах программной инженерии, которые мы с вами сейчас рассмотрим.

Подробнее о проблемах проектирования ПО можно посмотреть в неоднозначной статье Кони Бюрера «От ремесла к науке: поиск основных принципов разработки ПО» http://interface.ru/fset.asp?Url=/rational/science.htm

Из чего складывается стоимость ПО?

Структура стоимости ПО существенно зависит от типа ПО, применяемых методов его разработки и … метода оценки. Так, многие авторы отмечают высокую долю стоимости этапа сопровождения. Для некоторых типов ПО она может составлять 60 и более процентов от общей стоимости. Между тем, этап сопровождения включает выполнение двух видов работ: исправление ошибок в программе (несоответствий первоначальным требованиям) и внесение изменений в программу (добавление новых требований). При другом подходе к оценке можно считать, что этап сопровождения на стоит оценивать отдельно, т.к. исправление ошибок можно отнести к продолжению тестирования, а внесение изменений – к новому проекту.

Типовое распределение стоимости между основными этапами (без сопровождения) выглядит следующим образом:

- 15% - спецификация – формулировка требований и условий разработки
- 25% - проектирование – разработка и верификация проекта
- 20% - разработка – кодирование и тестирование компонент
- 40% - интеграция и тестирование – объединение и сборочное тестирование продукта

Отклонения от этой схемы в зависимости от типа ПО выглядят следующим образом:

Для коробочного ПО характерна более высокая доля тестирования за счет сокращения прежде всего доли спецификации (до 5%)

Распределение стоимости заказного ПО зависит от его сложности. При сложном ПО также возрастает доля интеграции и тестирования, но за счет сокращения доли проектирования и разработки Доля спецификаций может возрастать. Сокращение доли проектирования и разработки достигается за счет применения опробованных проектных решений и повторного использования готовых компонент.

Применение опробованных решений и готовых компонент при создании коробочных продуктов позволяет повысить качество и сократить сроки разработки.

Методы программной инженерии?

Метод программной инженерии — это структурный подход к созданию ПО, который способствует производству высококачественного продукта эффективным в экономическом аспекте способом. В этом определении есть две основные составляющие: (а) создание высококачественного продукта и (б) экономически эффективным способом. Иными словами, метод – это то, что обеспечивает решение основной задачи программной инженерии: создание качественного продукта при заданных ресурсах времени, бюджета, оборудования, людей.

Начиная с 70-х годов создано достаточно много методов разработки ПО. Наиболее известны:

- Метод структурного анализа и проектирования Том ДеМарко (1978),
- Метод сущность-связь проектирования информационных систем Чен (1976)
- Метод объектно-ориентированного анализа Буч (1994), Рамбо (1991).

Метод программной индустрии основан на идее создания моделей ПО с поэтапным преобразованием этих моделей в программу – окончательную модель решаемой задачи. Так, на этапе спецификаций создается модель – описание требований, которая далее преобразуется в модель проекта ПО, проект – в программный код. При этом важно, чтобы модели метода представлялись графически с помощью некоторого языка представления моделей.

Методы должны включать в себя следующие компоненты:

- Описание моделей системы и нотация, используемая для описания этих моделей (например, объектные модели, конечно-автоматные модели и т.д.)
- Правила и ограничения, которые надо выполнять при разработке моделей (например, каждай объект должен иметь одинаковое имя)

- Рекомендации — эвристики, характеризующие хорошие приемы проектирования в данном методе (скажем, рекомендация о том, что ни у одного объекта не должно быть больше семи подобъектов)
- Руководство по применению метода — описание последовательности работ (действий), которые надо выполнить для построения моделей (все атрибуты должны быть задокументированы до определения операций, связанных с этим объектом)

Нет идеальных методов, все они применимы только для тех или иных случаев. Нет абсолютных методов – применяемые на практике методы могут включать элементы различных подходов. Выбор метода составляет задачу специалиста по программной инженерии.

Что такое CASE?

CASE - Computer Aided System Engineering - различного рода инструментальные программы, используемые для поддержки процесса создания программ

CASE средства могут быть классифицированы по нескольким признакам:

- По уровню применения:
  - Upper CASE -средства анализа требований
  - Middle CASE - средства проектирования
  - Low CASE - ссредства разработки приложений
- Специализированные
  - Средства проектирования баз данных
  - Средства реинжиниринга (восстановления) модели (формирование ERD на основе анализа схем БД или формирования диаграмм на основе анализа программных кодов)
- Вспомогательные
  - Планирования и управления проектом
  - Конфигурационного управления
  - Тестирования
- Интегрированные CASE охватывают все этапы и процессы создания ПО от анализа требований до тестирования и выпуска документации. Интегрированные CASE выступают, как правило, в виде набора согласованных по интерфейсу средств, предназначенных для поддержки отдельных этапов процесса.

В настоящее время существует очень много CASE средств и фирм, специализирующихся на их разработке (Rational). При выборе CASE средств следует руководствоваться основным принципом: сначала метод создания ПО, а потом – CASE средства, применимые для этого метода. Выбор наоборот чреват нехорошими последствиями.

Свойства хорошей программы?

Прежде всего, хорошая программа должна делать то, что ожидает от нее заказчик – т.е. удовлетворять требованиям заказчика. Такие требования называют функциональными. Но кроме функциональных требований, существует еще ряд общих характеристик, которым в той или иной степени должна обладать каждая программа. Эти характеристики принято называть нефункциональными требованиями. К нефункциональным требованиям относят:

- Сопровождаемость (maintainability). Сопровождаемость означает, что программа должна быть написана с расчетом на дальнейшее развитие. Это критическое свойство системы, т.к. изменения ПО неизбежны вследствие изменения бизнеса. Сопровождение программы выполняют, как правило, не те люди, которые ее разрабатывали. Сопровождаемость включает такие элементы как наличие и понятность проектной документации, соответствие проектной документации исходному коду, понятность исходного кода, простота изменений исходного кода, простота добавления новых функций.
- Надежность (dependability). Надежность ПО включает такие элементы как:
    - Отказоустойчивость – возможность восстановления программы и данных в случае сбоев в работе
    - Безопасность – сбои в работе программы не должны приводить к опасным последствиям (авариям)
    - Защищенность от случайных или преднамеренных внешних воздействий (защита от дурака, вирусов, спама).
- Эффективность (efficiency). ПО не должно впустую тратить системные ресурсы, такие как память, процессорное время, каналы связи. Поэтому эффективность ПО оценивается следующими показателями: время выполнения кода, загруженность процессора, объем требуемой памяти, время отклика и т.п.
- Удобство использования (usability). ПО должно быть легким в использовании, причем именно тем типом пользователей, на которых рассчитано приложение.

Это включает в себя интерфейс пользователя и адекватную документацию. Причем, пользовательский интерфейс должен быть не интуитивно, а профессионально понятным пользователю.

Следует отметить, что реализация нефункциональных требований часто требует больших затрат, чем функциональных. Так, сопровождаемость требует значительных усилий по поддержанию соответствия проекта исходному коду и применения специальных методов создания модифицируемых программ. Надежность – дополнительных средств восстановления системы при сбоях. Эффективность – поиска специальных архитектурных решений и оптимизации кода. А удобство – проектирования не «интуитивно» понятного, а профессионально понятного интерфейса пользователя.

Профессиональные и этические требования

Развитие средств вычислительной техники, коммуникаций и программных систем (Internet, телекоммуникации, распределенные системы, IP телефония, компьютерные игры и обучающие программы) оказывает все большее воздействие на общество. Роль специалистов по программному обеспечению при этом все время возрастает. Они работают в определенном правовом и социальном окружении, находятся под действием, международных, национальных и местных законодательств.

Ясно, что программисты, как и специалисты других профессий, должны быть честными и порядочными людьми. Но вместе с тем, программисты не могут руководствоваться только моральными нормами или юридическими ограничениями, т.к. они обычно бывают связаны более тонкими профессиональными обязательствами:

- Конфиденциальность – программные специалисты должны уважать конфиденциальность в отношении своих работодателей или заказчиков независимо от того, подписывалось ли ими соответствующее соглашение.
- Компетентность – программный специалист не должен завышать свой истинный уровень компетентности и не должен сознательно браться за работу, которая этому уровню не соответствует.
- Защита интеллектуальной собственности – специалист должен соблюдать законодательство и принципы защиты интеллектуальной собственности при использовании чужой интеллектуальной собственности. Кроме того, он должен защищать интеллектуальную собственность работодателя и клиента. Внимание:

создаваемая им интеллектуальная собственность является собственностью работодателя или клиента.

- Злоупотребление компьютером – программный специалист не должны злоупотреблять компьютерными ресурсами работодателя или заказчика; под злоупотреблениями мы здесь понимаем широкий спектр — от игр в компьютерные игрушки на рабочем месте до распространения вирусов и т.п.

Кодекс этики IEEE-CS/ACM

В разработке таких этических обязательств ведущую роль играют профессиональные сообщества. Такие общества, как ACM (Association for Computing Machinery - Ассоцтация по вычислительной технике), IEEE (Institute of Electrical and Electronic Engineers – Институт инженеров по электротехнике и электронике) и BCS (British Computer Society – Британское компьютерное общество) совместно разработали и опубликовали IEEE-CS/ACM Software Engineering Code of Ethics and Professional Practices – Кодекс этики и профессиональной практики программной инженерии. Члены этих организация принимают обязательство следовать этому кодексу в момент вступления в организацию.

Кодекс содержит восемь Принципов, связанных с поведением и решениями, принимаемыми профессиональными программистами, включая практиков, преподавателей, менеджеров и руководителей высшего звена. Кодекс распространяется также на студентов и «подмастерьев», изучающих данную профессию. Кодекс имеет краткую и полную версии

Краткая версия кодекса

- суммирует стремления кодекса на высоком уровне абстракции.
- полная версия показывает как эти стремления отражаются на деятельности профессиональных программистов.
- без высших принципов детали кодекса станут казуистическими и нудными;
- без деталей стремления останутся возвышенными, но пустыми;
- вместе же они образуют целостный кодекс.

Программные инженеры должны добиваться, чтобы анализ, спецификация, проектирование, разработка, тестирование и сопровождение программного обеспечения стали полезной и уважаемой профессией. В соответствии с их приверженностью к процветанию, безопасности и благополучию общества, программные инженеры будут руководствоваться следующими Восемью Принципами:

1. ОБЩЕСТВО - программные инженеры будут действовать соответственно общественным интересам.

2. КЛИЕНТ И РАБОТОДАТЕЛЬ - программные инженеры будут действовать в интересах клиентов и работодателя, соответственно общественным интересам.

3. ПРОДУКТ - программные инженеры будут добиваться, чтобы произведенные ими продукты и их модификации соответствовал высочайшим профессиональным стандартам.

4. СУЖДЕНИЕ - программные инженеры будут добиваться честности и независимости в своих профессиональных суждениях.

5. МЕНЕДЖМЕНТ - менеджеры и лидеры программных инженеров будут руководствоваться этическим подходом к руководству разработкой и сопровождением ПО, а также будут продвигать и развивать этот подход.

6. ПРОФЕССИЯ - программные инженеры будут улучшать целостность и репутацию своей профессии соответственно с интересами общества.

4. КОЛЛЕГИ - программные инженеры будут честными по отношению к своим коллегам и будут всячески их поддерживать.

8. ЛИЧНОСТЬ - программные инженеры в течение всей своей жизни будут учиться практике своей профессии и будут продвигать этический подход к практике своей профессии.

Полная версия кодекса: IEEE-CS/ACM Software Engineering Ethics and Professional Practices. http://www.computer.org/tab/seprof/code.htm#Public.

Стандарты программной инженерии

Как отмечалось, по происхождению программные продукты бывают двух типов: заказные (под заказ конкретного потребителя) и коробочные (для массовой продажи на рынке). Для заключения контракта заказчик должен быть уверен, что разработчик справится и не завалит проект. В мировой практике промышленного производства гарантией успеха являются стандарты на производство продуктов и услуг и сертификация производителей на соответствие этим стандартам.

Процесс стандартизации и сертификации давно вошел и в программную инженерию, где он составляет основу промышленного производства программных продуктов. При изготовлении коробочных продуктов стандартизация имеет не меньшее значение, т.к. она обеспечивает качество продуктов и продвижение их на рынок.

Какие бывают стандарты?

Среди всего многообразия стандартов принято выделять следующие основные типы стандартов:

Корпоративные стандарты разрабатываются крупными фирмами (корпорациями) с целью повышения качества своей продукции. Такие стандарты разрабатываются на основе собственного опыта и с учетом требований мировых стандартов. Корпоративные стандарты не сертифицируются, но являются обязательными для применения внутри корпорации. В условиях рыночной конкуренции могут иметь закрытый характер.

Отраслевые стандарты действуют в пределах организаций некоторой отрасли (министерства). Например, СНИП – строительные нормы и правила. Разрабатываются с учетом требований мирового опыта и специфики отрасли. Являются, как правило, обязательными для отрасли. Подлежат сертификации.

Государственные стандарты (ГОСТы) принимаются государственными органами, в некоторых случаях имеют силу закона. Разрабатываются с учетом мирового опыта или на основе отраслевых стандартов. Могут иметь как рекомендательный, так и обязательный характер (стандарты безопасности). Для сертификации создаются государственные или лицензированные органы сертификации.

Международные стандарты. Разрабатываются, как правило, специальными международными организациями на основе мирового опыта и лучших корпоративных стандартов. Имеют сугубо рекомендательный характер. Право сертификации получают организации (государственные и частные), прошедшие лицензирование в международных организациях.

Кто разрабатывает стандарты программной инженерии?

Основными разработчиками международных стандартов являются следующие организации:

ISO - International Organization for Standardization – Международная организация по стандартизации. Наиболее представительная и влиятельная организация, разрабатывающая стандарты почти во всех областях деятельности, в том числе и в IT.

ACM - Association for Computing Machinery –Ассоциация по вычислительной технике. Всемирная научная и образовательная организация в области вычислительной технике. Известна также и разработкой образовательных стандартов.

SEI - Software Engineering Institute - Институт Программной Инженерии. Исследования в области программной инженерии с упором на разработку методов оценки и повышения качества ПО. Стандарты по качеству ПО и зрелости организаций, разрабатывающих ПО.

PMI - Project Management Institute - Международный Институт Проектного Менеджмента (Управления Проектами). Некоммерческая организация, целью которой является продвижение, пропаганда, развитие проектного менеджмента в разных странах. PMI разрабатывает стандарты проектного менеджмента, занимается повышением квалификации специалистов.

IEEE - Институт инженеров по электронике. Поддержка научных и практических разработок в области электроники и вычислительной техники. Большие вложения в разработку стандартов в этой области.

См. также: Васютович В.В. Стандартизация в области информационных технологий. http://inform.alee.ru/item_541.html

Основные стандарты программной инженерии

Наиболее известными стандартами программной инженерии являются:

- ISO/IEC 12207 - Information Technology - Software Life Cycle Processes - Процессы жизненного цикла программных средств. Стандарт содержит определения основных понятий программной инженерии (в частности программного продукта и жизненного цикла программного продукта), структуры жизненного цикла как совокупности процессов, детальное описание процессов жизненного цикла.
- SEI CMM - Capability Maturity Model (for Software) - модель зрелости процессов разработки программного обеспечения. Стандарт отвечает на вопрос: «Какими признаками должна обладать профессиональная организация по разработке ПО?». Профессионализм организации определяется через зрелость процесса, применяемого этой организацией. Выделяются пять уровней зрелости процесса.
- ISO/IEC 15504 - Software Process Assessment - Оценка и аттестация зрелости процессов создания и сопровождения ПО. Является развитием и уточнением ISO 12207 и SEI CMM. Содержит расширенное по отношению ISO 12207 количество процессов жизненного цикла и 6 уровней зрелости процессов. Дается подробное описание схемы аттестации процессов, на основе результатов которой может

быть выполнена оценка зрелости процессов и даны рекомендации по их усовершенствованию.

- PMBOK - Project Management Body of Knowledge - Свод знаний по управлению проектами. Содержит описания состава знаний по следующим 9 разделам (областям знаний) управления проектами
- SWBOK - Software Engineering Body of Knowledge - Свод знаний по программной инженерии - содержит описания состава знаний по 10 разделам (областям знаний) программной инженерии.
- ACM/IEEE CC2001 - Computing Curricula 2001 – Академический образовательный стандарт в области компьютерных наук. Выделены 4 основных раздела компьютерных наук: Computer science, Computer engineering, Software engineering и Information systems, по каждому из которых описаны области знаний соответствующего раздела, состав и планы рекомендуемых курсов

ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА

Начало стандартизации жизненного цикла ПО

Методологическую основу любой инженерии составляет понятие жизненного цикла (ЖЦ) изделия (продукта) как совокупности всех действий, которые надо выполнить на протяжении всей «жизни» изделия. Смысл жизненного цикла состоит во взаимосвязанности всех этих действий. Жизненный цикл промышленного изделия определяется как последовательность этапов (фаз, стадий), состоящих из технологических процессов, действий и операций. Обычно к таким этапам относят: проектирование, изготовление образца, организацию производства, серийное производство, эксплуатацию, ремонт, вывод из эксплуатации. Организация промышленного производства с позиции жизненного цикла позволяет рассматривать все его этапы во взаимосвязи, что ведет к сокращению сроков, стоимости и трудозатрат.

Как отмечалось выше, впервые о необходимости рассматривать разработку ПО с позиций его жизненного цикла заговорили в 1968 г. Исторически основными стандартами на ЖЦ ПО являлись:

1985 (уточнен в 1988 г.) DOD-STD-2167 A – Разработка программных средств для систем военного назначения. Первый формализованный и утвержденный стандарт жизненного цикла для проектирования ПС систем военного назначения по заказам Министерства обороны США. Этим документом регламентированы 8 фаз (этапов) при создании сложных критических ПС и около 250 типовых обязательных требований к процессам и объектам проектирования на этих этапах.

1994г. MIL-STD-498. Разработка и документирование программного обеспечения. Принят Министерством обороны США для замены DOD-STD-2167 A и ряда других стандартов. Он предназначен для применения всеми организациями и предприятиями, получающими заказы Министерства обороны США. В 1996 г. утверждено очень подробное (407 стр.) руководство “Применение и рекомендации к стандарту MIL-STD-498”. Основную часть составляют 75 подразделов — рекомендаций по обеспечению и реализации процессов ЖЦ сложных критических ПС высокого качества и надежности, функционирующих в реальном времени.

1995г. IEEE 1074. Процессы жизненного цикла для развития программного обеспечения. Охватывает полный жизненный цикл ПС, в котором выделяются шесть крупных базовых процессов. Эти процессы детализируются 16 частными процессами. В

последних имеется еще более мелкая детализация в совокупности на 65 процессов-работ. Содержание каждого частного процесса начинается с описания общих его функций и задач и перечня действий — работ при последующей детализации. Для каждого процесса в стандарте представлена входная и результирующая информация о его выполнении и краткое описание сущности процесса. В стандарте внимание сосредоточено преимущественно на непосредственном создании ПС и на процессах предварительного проектирования. В приложении представлены четыре варианта адаптации максимального состава компонентов ЖЦ ПС к конкретным особенностям типовых проектов.

Между тем, разработка стандартов ЖЦ и их практическое применение сталкивались с рядом проблем:

- Внедрение стандартов требовало вложения значительных средств, что не всегда окупалось.
- Было неясно, все ли требуемые процессы надо выполнять и в какой мере.
- Различные типы ПО (ИС, реального времени, бизнес системы), различные требования.
- Высокая динамика отрасли и устаревание стандартов.
- Терминологическая неоднозначность различных государственных и корпоративных стандартов.
- Во многих случаях применение стандартов было вызвано только требованиями заказчиков, хотя на практике часто тормозило выполнение проектов.

Стандарт ISO/IEC 12207 – процессы жизненного цикла ПП

В 1995 году ISO совместно с IEC (International Electrotechnical Commission - Международная электротехническая комиссия) был принят международный стандарт ISO/IEC 12207 - Information Technology - Software Life Cycle Processes. В 2000 г. он был принят как ГОСТ (ИСО/МЭК) 12207 - Процессы жизненного цикла программных средств [4].

Стандарт ISO/IEC 12207 разрабатывался с учетом лучшего мирового опыта на основе вышеперечисленных стандартов. Основными результатами стандарта ISO 12207 являются:

- Введение единой терминологии по разработке и применению ПО (предназначен не только для разработчиков, но и для заказчиков, пользователей, всех заинтересованных лиц).

- Разделение понятий ЖЦ ПО и модели ЖЦ ПО. ЖЦ ПО в стандарте вводится как полная совокупность всех процессов и действий по созданию и применению ПО, а модель ЖЦ – конкретный вариант организации ЖЦ, обоснованно (разумно) выбранный для каждого конкретного случая.
- Описание организации ЖЦ и его структуры (процессов).
- Выделение процесса адаптации стандарта для построения конкретных моделей ЖЦ.

Стандарт ISO/IEC 12207 определяет организацию ЖЦ программного продукта как совокупность процессов, каждый из которых разбит на действия, состоящие из отдельных задач; устанавливает структуру (архитектуру) ЖЦ программного продукта в виде перечня процессов, действий и задач.

В соответствии со стандартом ISO/IEC 12207 процессы ЖЦ делятся на три группы:

- Основные
- Вспомогательные
- Организационные

Отдельно описан процесс адаптации стандарта, содержащий основные работы, которые должны быть выполнены при адаптации настоящего стандарта к условиям конкретного программного проекта.

К числу основных относятся процессы:

- Заказа. Определяет работы заказчика, то есть организации, которая приобретает систему, программный продукт или программную услугу.
- Поставки. Определяет работы поставщика, то есть организации, которая поставляет систему, программный продукт или программную услугу заказчику.
- Разработки Определяет работы разработчика, то есть организации, которая проектирует и разрабатывает программный продукт.
- Эксплуатации. Определяет работы оператора, то есть организации, которая обеспечивает эксплуатационное обслуживание вычислительной системы в заданных усло-виях в интересах пользователей.
- Сопровождения. Определяет работы персонала сопровождения, то есть организации, которая предоставляет услуги по сопровождению программного продукта, состоящие в контролируемом изменении программного продукта с целью сохранения его исходного состояния и функциональных возможностей.

Данный процесс охватывает перенос и снятие с эксплуатации программного продукта.

Вспомогательными процессами являются:

- Документирования. Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.
- Управления конфигурацией. Определяет работы по управлению конфигурацией.
- Обеспечения качества. Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных планов. Совместные анализы, аудиторские проверки, верификация и аттестация могут использоваться в качестве методов обеспечения качества.
    - Верификации. Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.
    - Аттестации. Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.
    - Совместного анализа. Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.
    - Аудита. Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).
- Решения проблем. Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

Организационные процессы жизненного цикла:

- Управления. Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.
- Создания инфраструктуры. Определяет основные работы по созданию основной структуры процесса жизненного цикла.

- Усовершенствования. Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.
- Обучения. Определяет работы по соответствующему обучению персонала.

Процессы жизненного цикла стандарта ISO/IEC 15504

Стандарт ISO/IEC 12207 разрабатывался 9 лет и достаточно быстро устарел. В 1998 г. выходит новый стандарт ISO/IEC TR 15504: Information Technology - Software Process Assessment (Оценка процессов разработки ПО) [5]. В этом документе рассматриваются вопросы аттестации, определения зрелости и усовершенствования процессов жизненного цикла ПО. Один из разделов документа содержит новую классификацию процессов жизненного цикла, являющуюся развитием стандарта ISO/IEC 12207.

Связь ISO/IEC TR 15504 со стандартом ISO 12207 состоит в том, что все процессы стандарта ISO/IEC 15504 принадлежат к одной из следующих типов:

- базовый — процесс из 12207;
- расширенный — расширение процесса из 12207;
- новый — процесс, не описанный в 12207;
- составляющий — часть процесса из 12207;
- расширенный составляющий — расширенная часть проц. из 12207

В соответствии с новой классификацией в трех группах процессов вводятся пять категорий процессов:

- Основные процессы: категория CUS: Потребитель-поставщик и категория ENG: Инженерная
- Вспомогательные процессы: категория SUP: Вспомогательная
- Организационные процессы: категория MAN Управленческая и категория ORG: Организационная.

Категория **Потребитель-Поставщик** состоит из процессов, непосредственно влияющих на потребителя, поддерживающих процесс разработки программного средства и его передачи потребителю и обеспечивающих возможность корректного использования программного средства или услуги. Включает следующие процессы:

- CUS.1 Процесс приобретения (Acquisition process)

  - CUS.1.1 Процесс подготовки приобретения (Acquisition preparation process)
  - CUS.1.2 Процесс выбора поставщика (Supplier selection process)
  - CUS.1.3 Процесс мониторинга поставщика (Supplier Monitoring process)
  - CUS.1.4 Процесс приемки (Customer Acceptance process)
- CUS.2 Поставки (Supply process)
- CUS.3 Процесс выявления требований (Requirements process)
- CUS.4 Эксплуатации (Operation process)
  - CUS.4.1 Процесс эксплуатационного использования (Operational use process)
  - CUS.4.2 Процесс поддержки потребителя(Customer support process)

Инженерная категория процессов состоит из процессов, которые непосредственно определяют, реализуют или поддерживают программный продукт, его взаимодействие с системой и документацию на него. В тех случаях, когда система целиком состоит из программных средств, инженерные процессы имеют отношение только к созданию и поддержанию этих программных средств. Включает следующие процессы:

- ENG.1 Процесс разработки (Development process)
  - ENG.1.1 Процесс анализа требований и разработки системы (Systemrequirements analysis and design process)
  - ENG.1.2 Процесс анализа требований к программным средствам (Software requirements analysis process)
  - ENG.1.3 Процесс проектирования программных средств (Software design process)
  - ENG.1.4 Процесс конструирования программных средств (Software construction process)
  - ENG.1.5 Процесс интеграции программных средств (Software integration process)
  - ENG.1.6 Процесс тестирования программных средств (Software testing process)
  - ENG.1.7 Процесс интеграции и тестирования системы (System integration andtesting process)
- ENG.2 Процесс сопровождения системы и программных средств(System and software maintenance process)

Вспомогательная категория состоит из процессов, которыми могут пользоваться любые другие процессы (включая другие вспомогательные процессы) в различные моменты жизненного цикла программных средств. Включает следующие процессы:

- SUP.1 Процесс документирования (Documentation process)
- SUP.2 Процесс управления конфигурацией (Configuration management process)
- SUP.3 Процесс обеспечения качества (Quality assurance process)
- SUP.4 Процесс верификации (Verification process)
- SUP.5 Процесс проверки соответствия (Validation process)
- SUP.6 Процесс совместных проверок (Joint review process)
- SUP.7 Процесс аудита (Audit process)
- SUP.8 Процесс разрешения проблем (Problem resolution process)

Управленческая категория состоит из процессов, содержащих практики общего характера, которые могут быть использованы каждым, кто управляет любым проектом или процессом в ходе жизненного цикла программных средств. К управленческой категории относятся следующие процессы:

- MAN.1 Процесс административного управления (Management process)
- MAN.2 Процесс управления проектами (Project management process)
- MAN.3 Процесс управления качеством (Quality Management process)
- MAN.4 Процесс управления рисками (Risk Management process)

Организационная категория процессов состоит из процессов, которые устанавливают цели функционирования организации и создают активы процессов, продуктов и ресурсов, которые, будучи использованы в проектах организации, способствуют выполнению ее целей. Хотя организационные практики в целом относятся не только к процессам, относящимся к программным средствам, последние выполняются в общем контексте организации, и для их эффективного использования необходимо соответствующее окружение. Кратко, эти организационные процессы создают инфраструктуру организации; используют все лучшее из того, что имеется (передовой опыт) во всех частях организации (эффективные процессы, лучшие навыки, качественный программный код, хорошие средства поддержки); делают это общедоступным в рамках всей организации; создают базу для постоянного совершенствования во всей организации.

К организационной категории принадлежат процессы:

- ORG.1 Процесс организационных установок (Organizational alignment process)
- ORG.2 Процесс усовершенствования (Improvement process)

    - ORG.2.1 Процесс создания процессов (Process establishment process)
    - ORG.2.2 Процесс аттестации процессов (Process assessment process)
    - ORG.2.3 Процесс усовершенствования процессов (Process improvement process)
- ORG.3 Процесс административного управления кадрами (Human resource management process)
- ORG.4 Процесс создания инфраструктуры (Infrastructure process)
- ORG.5 Процесс измерения (Measurement process)
- ORG.6 Процесс повторного использования (Reuse process)

Подробнее: В. Липаев. Стандарты, регламентирующие жизненный цикл сложных программных комплексов. http://www.pcweek.ru/year1998/N24/CP1251/Reviews/chapt1.htm

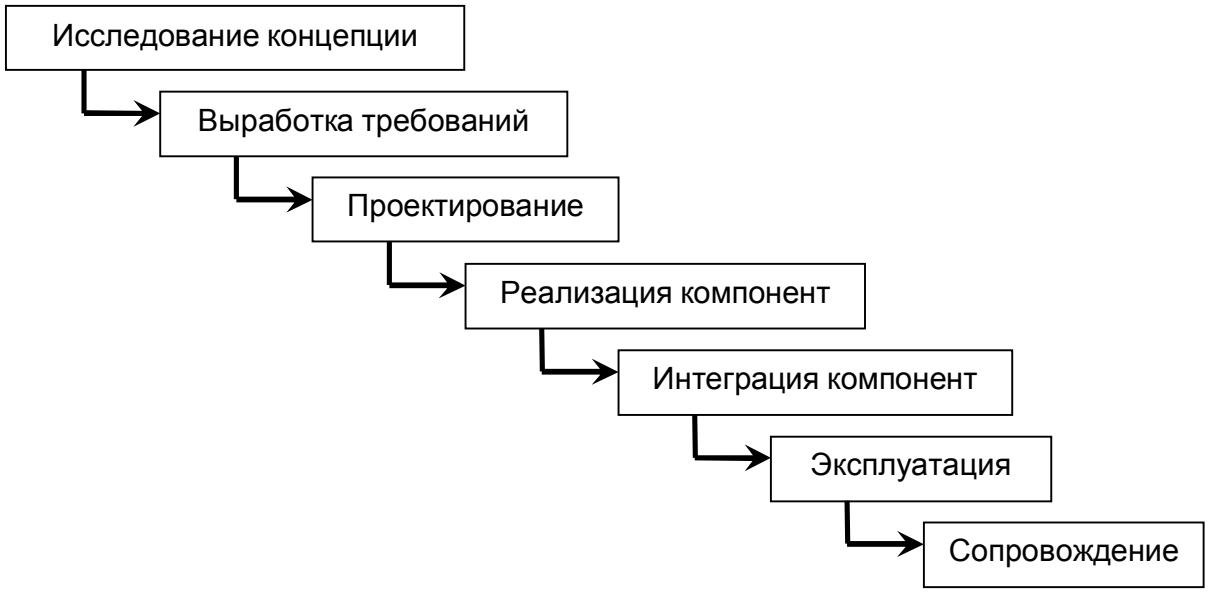
Модель жизненного цикла программного продукта

Приведенные выше стандарты жизненного цикла устанавливают состав и организацию процессов, которые могут выполняются на различных этапах жизненного цикла ПО. Во многих случаях эти процессы оказываются избыточными и приведенные стандарты адаптируются к условиям выполнения конкретных проектов в соответствии с выбираемыми моделями жизненного цикла, учитывающими специфику этих проектов.

В стандарте ISO/IEC 12207 модель жизненного цикла (life cycle model) определяется как структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения ее использования. При этом, конкретные модели определяются особенностью задач, ограничениями на ресурсы, опытом разработчиков и т.д.

Известны некоторые типовые модели ЖЦ ПО, которые проявили себя в определенных условиях, имеют определенные преимущества, недостатки и условия применимости. Эти типовые модели устанавливают некоторые принципы организации модели жизненного цикла ПО. К числу основных моделей жизненного цикла ПО следует отнести каскадную и спиральную модели. На практике часто используют итерационную, V-образную, инкрементную и модель быстрого прототипирования.

Каскадная модель жизненного цикла ПО



Представленная на схеме каскадная модель (водопад - waterfall) включает выполнение следующих фаз:

- **исследование концепции** — происходит исследование требований, разрабатывается видение продукта и оценивается возможность его реализации.
- **определение требований** — определяются программные требования для информационной предметной области системы, предназначение, линии поведения, производительность и интерфейсы.
- **разработка проекта** — разрабатывается и формулируется логически по-следовательная техническая характеристика программной системы, включая структуры данных, архитектуру ПО, интерфейсные представления и процессуаль-ную (алгоритмическую) детализацию;
- **реализация** — эскизное описание ПО пре-вращается в полноценный программный продукт. Результат: исходный код, база данных и документация. В реализации обычно выделяют два этапа: реализацию компонент ПО и интеграцию компонент в готовый продукт. На обоих этапах выполняется кодирование и тестирование, которые тоже иногда рассматривают как два подэтапа.
- **эксплуатация и поддержка** - подразумевает запуск и текущее обеспечение, включая предоставление технической помощи, обсуждение возникших вопросов

с пользователем, регистрацию запросов пользователя на модернизацию и внесение изменений, а также корректирование или устранение ошибок;

- **сопровождение —** устранение программных ошибок, неис-правностей, сбоев, модернизация и внесение изменений. Состоит из итераций разработки.

Основными принципами каскадной модели являются:

- Строго последовательное выполнение фаз
  - Каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы
  - Каждая фаза имеет определенные критерии входа и выхода: входные и выходные данные.
  - Каждая фаза полностью документируется
  - Переход от одной фазы к другой осуществляется посредством формального обзора с участием заказчика
- Основа модели – сформулированные требования (ТЗ), которые меняться не должны
- Критерий качества результата – соответствие продукта установленным требованиям.

Каскадная модель имеет следующие преимущества:

- Проста и понятна заказчикам, т.к часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО
- Проста и удобна в применении:
  - процесс разработки выполняется поэтапно;
  - ее структурой может руководствоваться даже слабо подготовленный в техническом плане или - неопытный персонал;
  - она способствует осуществлению строгого контроля менеджмента проекта;
- Каждую стадию могут выполнять независимые команды (все документировано).
- Позволяет достаточно точно планировать сроки и затраты.

При использовании каскадной модели для «неподходящего» проекта могут проявляться следующие ее основные недостатки:

- Попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо проблему или недостаток, приведет к значительному увеличению затрат и сбою в графике.

- Интеграция компонент, на которой обычно выявляется большая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость устранения ошибок.
- Запаздывание с получением результатов – если в процессе выполнения проекта требования изменились, то получится устаревший результат

Недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования или требования могут меняться в процессе выполнения проекта. В этом случае разработка ПО имеет принципиально циклический характер.

Каскадная модель впервые четко сформулирована в 1970 году Ройсом [6]. На начальном периоде она сыграла ведущую роль как метод регулярной разработки сложного ПО. В семидесятых-восьмидесятых годах XX века модель была принята как стандарт министерства обороны США. Со временем недостатки каскадной модели стали проявляться все чаще и возникло мнение, что она безнадежно устарела. Между тем, каскадная модель не утратила своей актуальности при решении следующих типов задач:

- Требования и их реализация максимально четко определены и понятны; используется неизменяемое определение продукта и вполне понятные технические методики. Это задачи типа:
  - научно-вычислительного характера (пакеты и библиотеки научных программ типа расчета несущих конструкций зданий, мостов, …);
  - операционные системы и компиляторы;
  - системы реального времени управления конкретными объектами;
- Повторная разработка типового продукта (автомати-зированного бухгалтерского учета, начисления зарплаты, …).
- Выпуск новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы (перенос уже существующего продукта на новую платформу).
- И наконец, принципы каскадной модели находят применение как элементы моделей других типов, о чем речь пойдет ниже.

Спиральная модель жизненного цикла ПО

На практике, при решении достаточно большого количества задач, разработка ПО имеет циклический характер, когда после выполнения некоторых стадий приходится возвращаться на предыдущие. Можно указать две основные причины таких возвратов:

- Ошибки разработчиков, допущенные на ранних стадиях и выявленные на поздних стадиях – ошибки анализа, проектирования, кодирования, выявляемые, как правило, на стадии тестирования.
- Изменение требований в процессе разработки («ошибки» заказчиков). Это или неготовность заказчиков сформулировать требования («Сказать, что должна делать программа я смогу только после того, как увижу как она работает»), или изменения требований, вызванные изменениями ситуации в процессе разработки (изменения рынка, новые технологии, …).

Циклический характер разработки ПО отражен в спиральной модели ЖЦ, описанной Б. Боэмом в 1988 году [7]. Спиральная модель была предложена как альтернатива каскадной модели, учитывающая повторяющийся характер разработки ПО. Основные принципы спиральной модели можно сформулировать следующим образом:

- Разработка вариантов продукта, соответствующих различным вариантам требований с возможностью вернуться к более ранним вариантам
- Создание прототипов ПО как средства общения с заказчиком для уточнения и выявления требований
- Планирование следующих вариантов с оценкой альтернатив и анализом рисков, связанных с переходом к следующему варианту
- Переход к разработке следующего варианта до завершения предыдущего в случае, когда риск завершения очередного варианта (прототипа) становится неоправданно высок.
- Использование каскадной модели как схемы разработки очередного варианта
- Активное привлечение заказчика к работе над проектом. Заказчик участвует в оценке очередного прототипа ПО, уточнении требований при переходе к следующему, оценке предложенных альтернатив очередного варианта и оценке рисков.

Схема работы спиральной модели выглядит следующим образом. Разработка вариантов продукта представляется как набор циклов

раскручивающейся спирали. Каждому циклу спирали соответствует такое же количество стадий, как и в модели каскадного процесса. При этом, начальные стадии, связанные с анализом и планированием представлены более подробно с добавлением новых элементов. В каждом цикле выделяются четыре базовые фазы:

- определение целей, альтернативных вариантов и ограничений.
- оценка альтернативных вариантов, идентификация и разрешение рисков.
- разработка продукта следующего уровня.
- планирование следующей фазы.

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку ПО. Здесь на первой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативы подходов к решению задачи. Далее проводится оценка подходов, устанавливаются их риски. На шаге разработки создается концепция (видение) продукта и путей его создания.

Следующий цикл начинается с планирования требований и деталей ЖЦ продукта для оценки затрат. На фазе определения целей устанавливаются альтернативные варианты требований, связанные с аранжировкой требований по важности и стоимости их выполнения. На фазе оценки устанавливаются риски вариантов требований. На фазе разработки – спецификация требований (с указанием рисков и стоимости), готовится демо-версия ПО для анализа требований заказчиком.

Следующий цикл – разработка проекта – начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам, …), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков, … На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демо-версия, отражающая основные проектные решения.

Следующий цикл – реализация ПО – также начинается с планирования. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков на этом цикле определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом – действующим вариантном (прототипом) продукта.

Отметим некоторые особенности спиральной модели:

- До начала разработки ПО есть несколько полных циклов анализа требований и проектирования.
- Количество циклов модели (как в части анализа и проектирования, так и в части реализации) не ограничено и определяется сложностью и объемом задачи
- В модели предполагаются возвраты на оставленные варианты при изменении стоимости рисков.

Спиральная модель (по отношению к каскадной) имеет следующие очевидные преимущества:

- Более тщательное проектирование (несколько начальных итераций) с оценкой результатов проектирования, что позволяет выявить ошибки проектирования на более ранних стадиях.
- Поэтапное уточнение требований в процессе выполнения итераций, что позволяет более точно удовлетворить требованиям заказчика
- Участие заказчика в выполнении проекта с использованием прототипов программы. Заказчик видит, что и как создается, не выдвигает необоснованных требований, оценивает реальные объемы финансирования.

- Планирование и управление рисками при переходе на следующие итерации позволяет разумно планировать использование ресурсов и обосновывать финансирование работ.
- Возможность разработки сложного проекта «по частям», выделяя на первых этапах наиболее значимые требования.

Основные недостатки спиральной модели связаны с ее сложностью:

- Сложность анализа и оценки рисков при выборе вариантов.
- Сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий)
- Сложность оценки точки перехода на следующий цикл
- Бесконечность модели – на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Спиральную модель целесообразно применять при следующих условиях:

- Когда пользователи не уверены в своих потребностях или когда требования слишком сложны и могут меняться в процессе выполнения проекта и необходимо прототипирование для анализа и оценки требований.
- Когда достижение успеха не гарантировано и необходима оценка рисков продолжения проекта.
- Когда проект является сложным, дорогостоящим и обоснование его финансирования возможно только в процессе его выполнения
- Когда речь идет о применении новых технологий, что связано с риском их освоения и достижения ожидаемого результата
- При выполнении очень больших проектов, которые в силу ограниченности ресурсов можно делать только по частям.

Другие типы моделей жизненнго цикла ПО

Каскадная и спиральная модели устанавливают некоторые принципы организации жизненного цикла создания программного продукта. Каждая из них имеет преимущества, недостатки и области применимости. Каскадная модель проста, но применима в случае, когда требования известны и меняться не будут. Спиральная модель учитывает такие важные показатели проекта как изменяемость требований, невозможность оценить заранее объем финансирования, риски выполнения проекта. Но спиральная модель сложна и требует больших затрат на сопровождение.

Существуют некоторые другие типы моделей, которое можно рассматривать как «промежуточные» между каскадной и спиральной моделями. Эти модели используют отдельные преимущества каскадной и спиральной моделей и достигают успеха для определенных типов задач.

Итерационная модель

Итерационная модель жизненного цикла является развитием классической каскадной модели и предполагает возможность возвратов на ранее выполненные этапы. При этом, причинами возвратов в классической итерационной модели являются выявленные ошибки, устранение которых и требует возврата на предыдущие этапы в зависимости от типа (природы) ошибки – ошибки кодирования, проектирования, спецификации или определения требований. Реально итерационная модель является более жизненной, чем классическая (строгая) каскадная модель, т.к. создание ПО всегда связано с устранением ошибок. Следует отметить, что уже в первой статье, посвященной каскадной модели, Боэм отмечал это обстоятельство и описал итерационный вариант каскадной модели. практически все применяемые модели жизненного цикла имеют итерационный характер, но цели итераций могут быть разными.

V-образная модель

V-образная модель была создана как итерационная разновидность каскадной модели. Целями итераций в этой модели является обеспечение процесса тестирования. Тестирование продукта обсуждается, проектируется и планируется на ранних этапах жизненного цикла разработки. План испытания приемки заказчиком разрабатывается на этапе планирования, а компоновочного испытания системы - на фазах анализа, разработки проекта и т.д. Этот процесс разработки планов испытания обозначен пунктирной линией между прямоугольниками V-образной модели. Помимо планов, на ранних этапах разрабатываются также и тесты, которые будут выполняться при завершении параллельных этапов.

Инкрементная (пошаговая) модель

Инкрементная разработка представляет собой процесс поэтапной реализации всей системы и поэтапного наращивания (приращения) функциональных возможностей. На первом шаге необходим полный заранее сформулированный набор требований, которые делятся по некоторому признаку на части. Далее выбирается первая группа требований и выполняется полный проход по каскадной модели. После того, как первый вариант

системы, выполняющий первую группу требований сдан заказчику, разработчики переходят к следующему шагу (второму инкременту) по разработке варианта, выполняющего вторую группу требований.

Особенностью инкрементной модели является разработка приемочных тестов на этапе анализа требований, что упрощает приемку варианта заказчиком и устанавливает четкие цели разработки очередного варианта системы.

Инкрементная модель особенно эффективна в случае, когда задача разбивается на несколько относительно независимых подзадач (разработка подсистем «Зарплата», «Бухгалтерия», «Склад», «Поставщики»). Кроме того, инкрементная модель может для внутренней итерации может использовать не только каскадную, но и другие типы моделей.

Модель быстрого прототипирования

Модель быстрого протитипирования предназначена для быстрого создания прототипов продукта с целью уточнения требований и поэтапного развития прототипов в конечный продукт. Скорость (высокая производительность) выполнения проекта обеспечивается планированием разработки прототипов и участием заказчика в процессе разработки.

Начало жизненного цикла разработки помещено в центре эллипса. Совместно с пользователем разрабатывается предварительный план проекта на основе предварительных требований. Результат начального планирования - документ, описывающий в общих чертах примерные графики и результативные данные.

Следующий уровень – создание исходного прототипа на основе быстрого анализа, проекта база данных, пользовательского интерфейса и некоторых функций. Затем начинается итерационный цикл быстрого прототипирования. Разработчик проекта демонстрирует очередной прототип, пользователь оценивает его функционирование, совместно определяются проблемы и пути их преодоления для перехода к следующему прототипу. Этот процесс продолжается до тех пор, пока пользователь не согласится, что очередной прототип в точности отображает все требования.

Получив одобрение пользователя, быстрый прототип преобразуют детальный проект, и систему настраивают на производственное использование. Именно на этом этапе настройки ускоренный прототип становится полностью действующей системой.

При разработке производственной версии программы, может понадобиться более высокий уровень функциональных возможностей, различные системные ресурсы, необходимых для обеспечения полной рабочей нагрузки, или ограничения во времени.

После этого следуют тестирование в предельных режимах, определение измерительных критериев и настройка, а затем, как обычно, функциональное сопровождение.

См. также: Итеративная и инкрементальная разработка: краткая история. http://www.sibinfo.ru/news/03_10_14/iid_history.shtml

Модели жизненного цикла MSF, RUP, XP

В настоящее время широкое применение получают так называемые промышленные технологии создания программного продукта. Эти технологии были разработаны фирмами, накопившими большой опыт создания ПО. Технологии представлены описаниями принципов, методов, применяемых процессов и операций. Такие технологии, как правило, поддерживаются набором CASE-средств, охватывают все этапы жизненного цикла продукта и успешно применяются для решения практических задач.

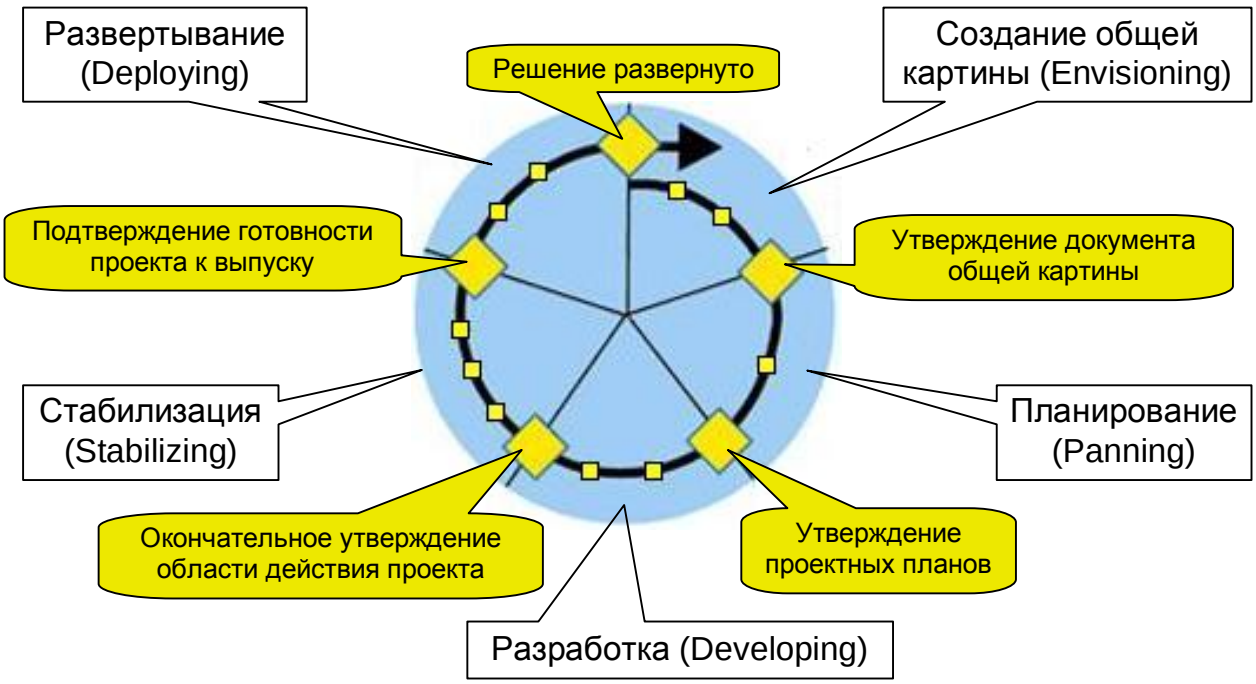
Хороший обзор моделей жизненного цикла промышленных технологий можно прочитать: Скопин И.Н.Основы менеджмента программных проектов. Лекция №11: Модели жизненного цикла в некоторых реальных методологиях программирования. http://www.intuit.ru/department/se/msd/11/1.html. Здесь мы рассмотрим особенности моделей жизненного цикла трех наиболее известных промышленных технологий:

- Microsoft Solution Framework (MSF) – разработка фирмы Microsoft, предназначенная для решения широкого круга задач. Технология масштабируема, т.е. настраиваема на решение задач любой сложности коллективом любой численности.
- Rational Unified Process (RUP) – разработка фирмы Rational, долгое время успешно занимавшейся созданием CASE-средств, применяемых на различных этапах жизненного цикла продукта от анализа до тестирования и документирования. Аналогично MSF, RUP универсальна, масштабируема и настраиваема на применение в конкретных условиях.
- Extreme Programming (XP) – активно развивающаяся в последнее время технология, предназначенная для решения относительно небольших задач, относительно небольшими коллективами профессиональных разработчиков в условиях жестко ограниченного времени.

Каждая из этих технологий имеет свои особенности организации модели жизненного цикла создания продукта.

Модель Microsoft Solution Framework

Одна из особенностей технологии MSF состоит в том, что она ориентирована не просто на создание программного продукта, удовлетворяющего перечисленным требованиям, а на поиск решения проблем, стоящих перед заказчиком. Различие состоит в том, что перечисляемые заказчиком требования являются проявлениями некоторых более глубоких проблем и неточность, неполнота, изменение требований в процессе разработки – следствие недопонимания проблем. Поэтому, в технологии MSF большое внимание уделяется анализу проблем заказчика и разработке вариантов системы для поиска решения этих проблем.



Модель жизненного цикла MSF является некоторым гибридом каскадной и спиральной моделей, сочетая простоту управления каскадной модели с гибкостью спиральной. Схема модели жизненного цикла MSF (модели процессов) представлена на слайде.

Модель жизненного цикла MSF ориентирована на “вехи” (milestones) – ключевые точки проекта, характеризующие достижение какого-либо существенного результата. Этот результат может быть оценен и проанализирован, что подразумевает ответ на вопрос: “А достигли ли мы целей, поставленных на этом шаге?». В модели предусматривается наличие основных вех (завершение главных фаз модели) и промежуточных, отражающих внутренние этапы главных фаз.

Основными фазами модели MSF являются:

- Создание общей картины приложения (Envisioning).На этом этапе решаются следующие основные задачи: оценка существующей ситуации; определение состава команды, структуры проекта, бизнес-целей, требований и профилей пользователей; разработка концепции решения и оценка риска. Устанавливаются две промежуточные вехи: "Организован костяк команды" и "Создана общая картина решения".
- Планирование (Panning). Включает планирование и проектирование продукта. На основе анализа требований разрабатывается проект и основные архитектурные решения, функциональные спецификации системы, планы и календарные графики, среды разработки, тестирования и пилотной эксплуатации. Этап состоит из трех стадий: концептуальное, логическое и физическое проектирование. На стадии **концептуального** проектирования задача рассматривается с точки зрения пользовательских и бизнес-требований и заканчивается определением набора сценариев использования системы. При **логическом** проектировании задача рассматривается с точки зрения проектной команды, решение представляется в виде набора сервисов. И уже на стадии **физического** проектирования задача рассматривается с точки зрения программистов, уточняются используемые технологии и интерфейсы.
- Разработка (Developing). Создается вариант решения проблемы, в виде кода и документации очередного прототипа, включая спецификации и сценарии тестирования. Основная веха этапа - "Окончательное утверждение области действия проекта". Продукт готов к внешнему тестированию и стабилизации. Кроме того, заказчики, пользователи, сотрудники службы поддержки и сопровождения, а также ключевые участники проекта могут предварительно оценить продукт и указать все недостатки, которые нужно устранить до его поставки.
- Стабилизация (Stabilizing). Подготовка к выпуску окончательной версии продукта, доводка его до заданного уровня качества. Здесь выполняется комплекс работ по тестированию (обнаружение и устранение дефектов), проверяется сценарий развертывания продукта. Когда решение становится достаточно устойчивым, проводится его пилотная эксплуатация в тестовой среде с привлечением пользователей и применением реальных сценариев работы.

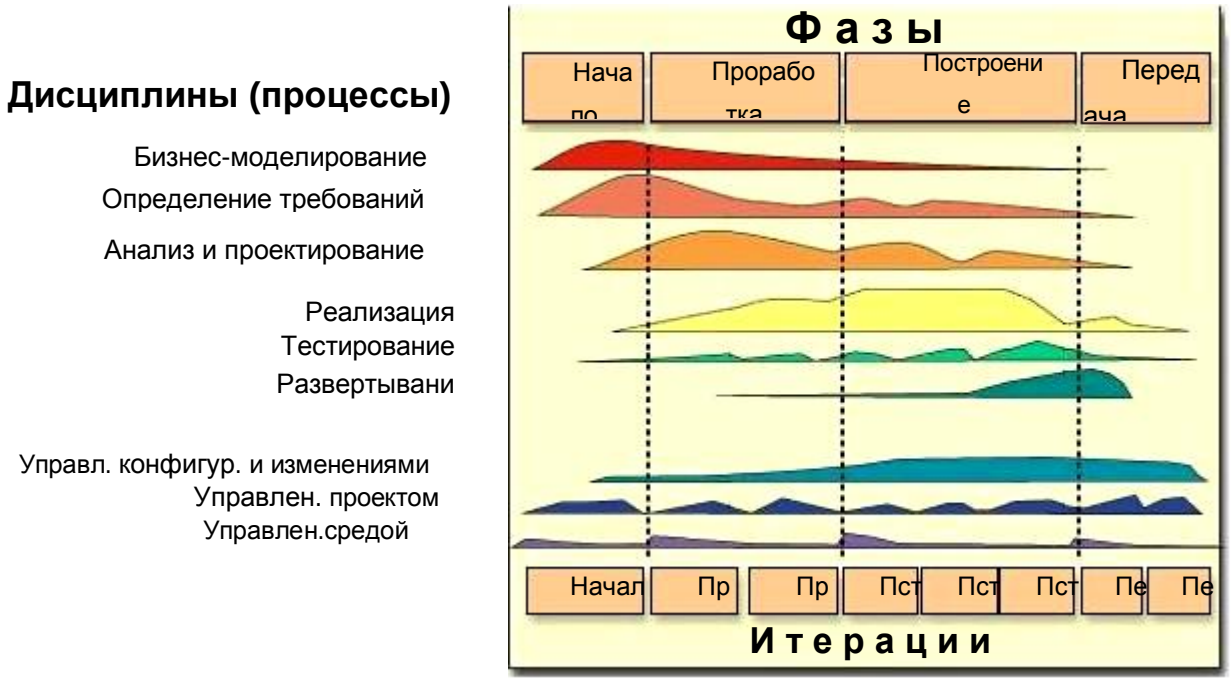
- Развертывание (Deploying). Выполняется установка решения и необходимых компонентов окружения, проводится его стабилизация в промышленных условиях и передача проекта в руки группы сопровождения. Кроме того, анализируется проект в целом на предмет уровня удовлетворенности заказчика.

Подробнее:

1. Модель процессов MSF, версия 3.1. http://www.microsoft.com/Rus/Download.aspx?file=/Msdn/Msf/MSF_process_model_rus.doc
2. Андрей Колесов. Введение в методологию Microsoft Solutions Framework. http://www.bytemag.ru/Article.asp?ID=2866

Модель Rational Unified Process

Модель жизненного цикла RUP является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели. В модели RUP выделяются 4 основные фазы, 9 видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта. RUP ориентирован на поэтапное моделирование создаваемого продукта с помощью языка UML.



Основными фазами RUP являются:

- **Фаза начала проекта (Inception).** Определяются основные цели проекта, бюджет проекта, основные средства его выполнения — технологии, инструменты, ключевой персонал, составляются предварительные планы

проекта. Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта.

- **Фаза проработки (Elaboration).** Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы.
- **Фаза построения (Construction).** Основная цель этой фазы — детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.
- **Фаза передачи (Transition).** Цель фазы — сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

В рамках каждой фазы возможно проведение нескольких итераций, количество которых определяется сложностью выполняемого проекта.

Деятельности (основные процессы) RUP делятся на пять рабочих и четыре поддерживающие. К рабочим деятельностям относятся:

- Моделирование предметной области (бизнес-моделирование, Business Modeling). Цели этой деятельности — понять бизнес-контекст, в котором должна будет работать система (и убедиться, что все заинтересованные лица понимают его одинаково), понять возможные проблемы, оценить возможные их решения и их последствия для бизнеса организации, в которой будет работать система.
- Определение требований (Requirements). Цели — понять, что должна делать система, определить границы системы и основу для планирования проекта и оценок ресурсозатрат в нем.
- Анализ и проектирование (Analysis and Design). Выработка архитектуры системы на основе ключевых требований, создание проектной модели, представленной в виде диаграмм UML, описывающих продукт с различных точек зрения.
- Реализация (Implementation). Разработка исходного кода, компонент системы, тестирование и интегрирование компонент.

- Тестирование (Test). Общая оценка дефектов продукта, его качество в целом; оценка степени соответствия исходным требованиям.

Поддерживающими деятельностями являются:

- Развертывание (Deployment). Цели — развернуть систему в ее рабочем окружении и оценить ее работоспособность.
- Управление конфигурациями и изменениями (Configuration and Change Management). Определение элементов, подлежащих хранению и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.
- Управление проектом (Project Management). Включает планирование, управление персоналом, обеспечения связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.
- Управление средой проекта (Environment). Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в проекте.

Модель Extreme Programming

Экстремальное программирование является примером так называемого метода «живой» разработки (см. Agile Development Method http://www.agilemodeling.org.ua/home.html). В группу «живых» входят, помимо экстремального программирования, входит еще ряд методов, о чем подробнее можно прочитать: Мартин Фаулер. Новые методологии программирования. http://www.maxkir.com/sd/newmethRUS.html.

Модель жизненного цикла XP является итерационно-инкрементной моделью быстрого создания (и модификации) протопопов продукта, удовлетворяющих очередному требованию (user story). Особенности этой модели представлены на схеме. Основными фазами модели можно считать:

- «Вброс» архитектуры – начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.
- Истории использования (User Story) – этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций. User Story являются требованиями для планирования очередной версии и одновременной разработки приемочных тестов (Acceptance tests) для ее проверки.
- Планирование версии (релиза). Проводится на собрании с участием заказчика путем выбора User Stories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования - получение оценок того, что и как можно сделать за 1-3 недели создания следующей версии продукта.
- Разработка проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.
- Тестирование проводится с участием заказчика, который участвует в составлении тестов.
- Выпуск релиза – разработанная версия передается заказчику для использования или бета-тестирования.

По завершению цикла делается переход на следующую итерацию разработки

Особенности модели жизненного цикла XP проясняют следующие принципы этого метода. Прежде всего, это принципы «живой» разработки ПО, зафиксированные в манифесте «живой» разработки:

- Люди их общение более важны, чем процессы и инструменты
- Работающая программа более важна, чем исчерпывающая документация
- Сотрудничество с заказчиком более важно, чем обсуждение деталей контракта
- Отработка изменений более важна, чем следование планам

Кроме того, в XP есть несколько правил (техник), характеризующих особенности модели его жизненного цикла:

- Живое планирование (planning game) - как можно быстрее определить объем работ, который нужно сделать до следующей версии ПО. Решение принимается на основе, в первую очередь, бизнес-приоритетов заказчика и, во-вторую, технических оценок. Планы изменяются, как только они начинают расходится с действительностью или пожеланиями заказчика.
- Частая смена версий (small releases) - первая работающая версия должна появиться как можно быстрее, и тут же должна начать использоваться. Следующие версии подготавливаются через достаточно короткие промежутки времени.
- Простые проектные решения (simple design) - в каждый момент времени система должна быть сконструирована так просто, насколько это возможно. Новые функции добавляются только после ясной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.
- Разработка на основе тестирования (test-driven development) - сначала пишутся тесты, потом реализуются модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.
- Постоянная переработка (refactoring) - системы для устранения излишней сложности, увеличения понятности кода, повышения его гибкости. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат.
- Программирование парами (pair programming) - весь код пишется двумя программистами на одном компьютере, что повышает его качество (отсутствие ошибок, понятность, читаемость,…).

Постоянная интеграция (continuous integration) - система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.

- 40-часовая рабочая неделя - сверхурочная работа рассматривается как признак больших проблем в проекте. Не допускается сверхурочная работа 2 недели подряд — это истощает программистов и делает их работу значительно менее продуктивной.

Подробности: http://www.xprogramming.ru/index.html.

УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

Проект и управление проектом

В различных источниках можно найти различные определения понятия «управление»:

- элемент, функция организованных систем различной природы (биологических, социальных, технических), обеспечивающая сохранение их определенной структуры, поддержание режима деятельности, реализацию их программ и целей. (СЭС);
- руководство, направление чей-либо деятельности. (www.mega.km.ru);
- изменение состояния объекта, системы или процесса, ведущее к достижению поставленной цели (словарь по кибернетике).

С точки зрения последнего (наиболее приемлемого для нас) определения, существенным является:

- наличие измеряемой цели управления (функции цели);
- наличие (возможность) управления (управляющего воздействия, влияющего на изменение функции цели);
- наличие измерений состояния объекта или процесса;
- ограниченность управления.

Что такое проект?

Слово «проект» происходит от латинского projectus, что означает «брошенный вперед». В последнее время слово «проект» употребляется достаточно часто (и часто всуе): проект озеленения улиц города, проект повышения квалификации сотрудников, проект реорганизации деятельности фирмы и т.д. Известны несколько определений проекта [2]:

Проект – это произвольный ряд действий или задач, имеющий определенную цель, которая будет достигнута в рамках выполнения некоторых заданий, характеризующимися определенными датами начала и окончания, пределами финансирования и ресурсами (Г. Керцнер).

Проект – одноразовая работа, которая имеет определенные даты начала и окончания, ясно определенные цели, возможности и, как правило, бюджет (Д. Льюис).

Проект – временное усилие, применяемое для того, чтобы создать уникальный продукт или услугу с определенной датой начала и окончания действия, отличающегося от продолжающихся, повторных действий и требующего прогрессивного совершенствования характеристик (PMI).

В этих определениях в той или иной степени отражаются следующие определяющие характеристики проекта:

- Цель проекта. Наличие четко выраженного конечного результата, выхода, продукции, определяемых в терминах затрат, качества и времени реализации.
- Уникальность. Проект - это разовое начинание, которое не будет повторяться. Даже “повторяющиеся” проекты, например, по строительству еще одного предприятия по той же проектной документации, значительно отличаются друг от друга использующимися ресурсами и средой реализации.
- Ограниченность во времени. Проект имеет начало и конец. Для его реализация необходима временная концентрация ресурсов. По минованию надобности, ресурсы используются на другие цели.
- Ограниченность ресурсов, выделяемых на выполнение проекта (финансовых, людских, материальных).
- Сложность. Для достижения целей проекта необходимо решить множество задач. Отношения между задачами могут быть довольно сложными, особенно, если в проекте много задач.
- Неопределенность. Возможность достижения цели в указанные сроки с выделенными ресурсами заранее не гарантирована.
- Предсказуемость. По мере реализации проекта, изменяется потребность в тех или иных ресурсах. Это изменение идет в определенной предсказуемой последовательности, определяемой жизненным циклом проекта.

Иными словами, проект – это достаточно сложный вид деятельности, которым сложно управлять в силу его уникальности и ограниченности ресурсов и времени. Это обстоятельство вносит в проект элемент неопределенности, а правильно организованное управление делает результаты предсказуемыми. Кстати, предсказуемый – не значит успешный. Это значит – во время завершенный (успешно) или во время прекращенный (неуспешно).

Следует отметить, что к проектам можно отнести далеко не любой вид деятельности. К непроектам обычно относят:

- **Программа** –широкомасштабное усилие, направленное на достижение некоторой комплексной цели: программа космических исследований, программа мелиорации земель Средней Азии. Цель программы не конкретна, сроки и ресурсы не определены.

Но программа может разбиваться на отдельные конкретные цели, для которых устанавливаются сроки и выделяются ресурсы. Т.е. программа может разбиваться на отдельные проекты: проект Апполон, проект Союз, проект строительства канала Волга - Амударья, ...

- **Выполнение установившегося процесса** – деятельность, которая выполняется многократно и постоянно: конвейерное производство, обработка заказов, ведение бухгалтерии. Имеет конкретную цель (выпуск запланированного количества продукции, получение установленной прибыли, ведение отчетности) и выделенные ресурсы, но не является уникальной или сложной и не связана с конкретными сроками. Управление такими повторяющимися процессами относительно простое.

  Изменение параметров установившихся процессов может превратиться в проект (повышение прибыльности фирмы) с конкретными целями (до 45%), сроками и выделенными ресурсами.
- **Решение творческой задачи** (научной или художественной). Здесь есть конкретная цель, уникальность и сложность, но, как правило, нет ограничений по времени и ресурсам (объединим усилия нашего коллектива и докажем теорему к Новому году!). Слишком велика степень неопределенности.

  Решение сложных научных проблем может разбиваться на отдельные исследования (эксперименты) с конкретно установленными целями, сроками и ресурсами.

Управление проектами

Среди различных определений управления проектами [2], наиболее полным следует считать определение, сформулированное PMI в Своде знаний по управлению проектами (PMBOK) [8]: «Управление проектом (Project Management - PM) – это наука и искусство руководства и координации людских и материальных ресурсов на протяжении жизненного цикла проекта путем применения современных методов и техники управления для достижения определенных в проекте результатов по составу и объему работ, стоимости, времени, качеству и удовлетворению участников проекта».

Управление проектом основано на двух китах (принципах):

- Умение – знание принципов и методов управления проектом (планирования, организация, составление графиков, контроль, управление и отслеживание).

- Навыки – опыт в области управления – применение умения для достижения целей в конкретных условиях.

Хотя разработка методов и приемов управления была начата еще в начале прошлого века, как дисциплина управление проектами начало складываться в 50-х годах XX столетия, что было вызвано необходимостью координации работ в крупных проектах по разработке вооружений и освоению космоса (США). Разрабатывались методы управления крупными проектами, среди которых наиболее известными являются:

- Метод критического пути – МКП (CPM – Critical Path Method)
- Метод анализа и оценки программ PERT (Program Evaluation and Review Technique)

60-80 гг. прошлого века характеризуются широким распространением методов управления проектами, созданием компьютерных программ на базе МКП, PERT и разработкой новых методов и программ правления проектами.

С 90 гг. XX в. главным образом благодаря усилиям PMI (Project Management Institute) управление проектами становится профессией и областью знаний.

По данным [9] в настоящее время в США почти не осталось компаний, которые не используют формальные методы управления проектами. В России формальные методы в проектах использует незначительное число предприятий. Большинство из этих инноваторов работают на рынке информационных технологий. Согласно исследованию, проведенному консалтинговой компанией Interthink, 97,5% компаний в США и Канаде используют формализованные подходы к управлению проектами, а 22,5% компаний используют полностью проектно-ориентированный подход для всех своих проектов.

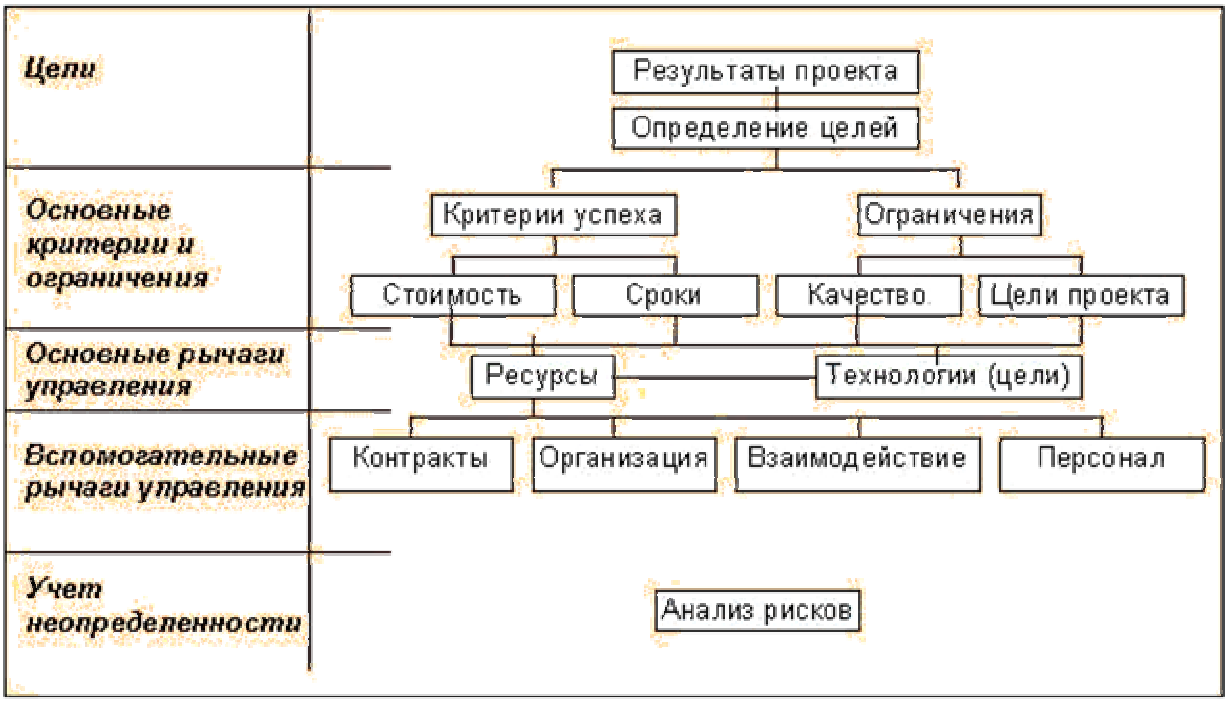
В России же ситуация прямо противоположная. По оценкам экспертов, только 5% компаний используют те или иные формальные подходы к управлению проектами. Между тем, в России также наблюдается взрывообразный интерес к методам и стандартам управления проектами. Любопытно, что большая часть из этих 5% российских предприятий приходится на IT-компании.

См. также:

- Кто и как управляет проектами в России? http://www.pmprofy.ru/content/rus/55/552-article.asp
- Результаты исследования в области управления проектами в Германии http://www.pmprofy.ru/content/rus/89/894-article.asp
- Результаты исследования управления проектами проводившиеся австрийским экспертом Роландом Гарисом http://www.pmprofy.ru/content/rus/65/653-article.asp

Категории управления проектами

Категории (от греч. kategoria высказывание; признак), в философии наиболее общие и фундаментальные понятия, отражающие существенные, всеобщие свойства и отношения явлений действительности и познания. Категории образовались как результат обобщения исторического развития познания и практики: материя и сознание, пространство и время, причинность, необходимость и случайность, возможность и действительность, и др.



Аналогично философским категориям, в области управления проектами существуют категории, отражающие основные понятия этой области. В общем случае выделяют следующие группы категорий:

- Цели, определяемые ожидаемыми результатами проекта.
- Критерии успеха и ограничения: стоимость, сроки, качество.
- Основные рычаги управления: ресурсы (являющиеся также ограничением) и технологии.
- Вспомогательные рычаги управления: контракты, организация, взаимодействие, персонал.
- Неопределенность, связанная с рисками выполнения проекта.

Треугольник ограничений проекта

Ключевой категорией, участвующей в процессе управления проектами, являются ограничения. Известный закон Лермана гласит: "Любую техническую проблему можно преодолеть, имея достаточно времени и денег", а следствие Лермана уточняет: "Вам никогда не будет хватать либо времени, либо денег". Если попросить менеджера описать, как он понимает свою основную задачу в выполнении проекта, то он ответит: "Обеспечить выполнение работ в срок, в рамках выделенных средств, в соответствии с техническим заданием". Именно эти три момента: время, бюджет и качество работ находятся под постоянным вниманием руководителя проекта. Их также можно назвать основными ограничениями, накладываемыми на проект.

Эти три основные ограничения (сроки, расходы и качество результата) взаимосвязаны. Для иллюстрации взаимосвязи используют треугольник ограничений, в котором качество, время и деньги интерпретируются площадями внутренних треугольников. В этом треугольнике центр и верхняя вершины фиксированы, а нижние вершины могут перемещаться. Треугольник иллюстрирует, что любое сокращение финансов или времени ведет к сокращению качества, а увеличение качества может быть достигнуто за счет увеличения финансирования или сроков.

Что должен знать менеджер проекта?

Главное действующее лицо проекта – менеджер. Он должен иметь ЗНАНИЯ и НАВЫКИ. Кто он программного проекта? Программист? Вначале так и было. Играли роль знания в предметной области (проектирования и разработка ПО). Но потом на первое место стали выходить ЗНАНИЯ и НАВЫКИ об управлении.

PMBOK: 9 областей управленческих знаний

PMBOK (Project Management Body of Knowledge - Свод знаний по управлению проектами) – международный стандарт состава знаний по управлению проектами, который разработан и развивается Институтом Проектного Менеджмента (Project Management Institute - PMI) [8]. PMBOK содержит описания состава знаний по следующим 9 разделам (областям знаний) управления проектами, каждый из которых делится на соответствующие подразделы:

1. Управление интеграцией проекта (Integration)
    a. Создание плана проекта (Project Plan Development)
    b. Исполнение плана проекта (Project Plan Execution)
    c. Контроль изменений в проекте (Integrated Change Control)
2. Управление объемом работ (Scope)
    a. Инициирование (Initiation) - формальное принятие решения о начале проекта (следующей фазы проекта)
    b. Планирование объема работ (Scope Planning) - разработка документа, описывающего объем работ
    c. Формализация объема работ (Scope Definition) - декомпозиция всего объема работ (основных необходимых результатов) на мелкие, измеримые задачи
    d. Верификация (Scope Verification) - подтверждение объема работ – формальная проверка приемлемости результатов работы.
    e. Управление изменениями объема работ (Scope Change Control) - контроль и утверждение изменений
3. Управление временем выполнения (Time)
    a. Определение состава работ (Activity Definition)
    b. Определение взаимосвязей работ (Activity Sequencing)
    c. Оценка длительностей работ (Activity Duration Estimating)
    d. Составление расписания проекта (Schedule Development)
4. Управление стоимостью (Cost)
    a. Планирование ресурсов (Resource Planning) - какие ресурсы в каком количестве нужны для работ
    b. Оценка стоимостей (Cost Estimating) - ресурсов, необходимых для работ
    c. Разработка бюджета (Cost Budgeting) - бюджетирование – распределение затрат по компонентам проекта
    d. Контроль стоимости (Cost Control) - управление изменениями бюджета
5. Управление качеством (Quality)
    a. Планирование качества (Quality Planning) - определение стандартов качества и средств для их достижения
    b. Обеспечение качества процесса (Quality Assurance) - плановая, регулярная оценка исполнения – проверка производственных процессов

   c. Контроль качества результатов (Quality Control) - мониторинг результатов проекта, определение их соответствия стандартам, выявление и устранение причин несоответствия качества
6. Управление персоналом (Human Resource)
   a. Организационное планирование (Organizational Planning) - идентификация, документирование, и назначение проектных ролей, обязанностей и структуры отчетности
   b. Подбор кадров (Staff Acquisition) - получение необходимых для проекта человеческих ресурсов, назначение персонала в команду проекта
   c. Развитие команды проекта (Team Development) - повышение производительности труда: индивидуальной и команды в целом (улучшение взаимодействия)
7. Управление коммуникациями (Communications)
   a. Планирование взаимодействия (Communications Planning) - определение потребностей участников проекта в информации и планирование информационных потоков
   b. Распределение информации (Information Distribution) - регулярное и своевременное обеспечение участников проекта необходимой информацией
   c. Оценка исполнения (Performance Reporting) - сбор и распространение отчетности о текущем состоянии проекта, достигнутом прогрессе и ожидаемых результатах
   d. Административное завершение (Administrative Closure) - создание, распространение (уничтожение) информации, необходимые для формального завершения проекта/фазы
8. Управление рисками (Risk)
   a. Планирование управления рисками (Risk Management Planning)
   b. Идентификация рисков (Risk Identification)
   c. Качественный анализ рисков (Qualitative Risk Analysis)
   d. Количественный анализ рисков (Quantitative Risk Analysis)
   e. Планирование реагирования на риски (Risk Response Planning)
   f. Мониторинг и контроль рисков (Risk Monitoring and Control)
9. Управление закупками и поставками (Procurement)
   a. Планирование закупок (Procurement Planning) - определение какие продуктов и услуг нужны извне

b. Планирование предложений (Solicitation Planning) - документирование требований к продуктам и услугам от внешних поставщиков
c. Получение предложений (Solicitation)
d. Выбор поставщиков (Source Selection)
e. Управление контрактами (Contract Administration) - регулирование отношений с поставщиками
f. Завершение контрактов (Contract Closeout) - подтверждение выполнения, разрешение споров

SQI: 34 компетенции IT менеджера

Институтом качества ПО (SQI - Software Quality Institute) разработан руководящей документ (Body of Knowledge) для сертификации менеджеров программных проектов (SWPM – SoftWare Project Management) [2]. В этом документе содержится список 34 компетенций, которыми должен обладать менеджер программного проекта. Список разделен на три основные категории: методика разработки продукта, навыки управления проектами и навыки управления персоналом:

1. Методика разработки продукта
   a. Процессы оценивания - определение критериев для отбора
   b. Знание стандартов процесса
   c. Определение продукта - идентификация клиентской среды и требований, выдвигаемых к продукту
   d. Оценка альтернативных процессов
   e. Управление требованиями- мониторинг изменения требований
   f. Управление субподрядчиками - планирование, управление и осуществление контроля
   g. Выполнение начальной оценки - оценка степени трудности, рисков, затрат и создание графиков
   h. Отбор методов и инструментов - определение процессов отбора
   i. Подгонка процессов - модификация стандартных процессов с целью удовлетворения требований проектов
   j. Отслеживание качества продукта - контроль качества в процессе разработки продукта
   k. Понимание действий по разработке продукта - изучение цикла разработки ПО
2. Навыки управления проектами

a. Создание структуры пооперационного перечня работ
b. Документирование планов - идентификация ключевых компонент
c. Оценка стоимости - стоимости завершения проекта
d. Оценка трудозатрат - необходимых для завершения проекта
e. Менеджмент рисков - идентификация, определение воздействия, обработка рисков
f. Отслеживание процесса разработки - контроль процесса разработки
g. Составление графика - разработка графика и ключевых стадий проекта
h. Выбор метрических показателей
i. Отбор инструментов менеджмента проекта - выбор методик и инструментов
j. Отслеживание процессов - мониторинг совместимости членов команды
k. Отслеживание хода разработки продукта - мониторинг хода разработки по выбранным метрическим показателям

3. Навыки управления персоналом
   a. Оценка производительности - оценка действий команды, направленных на повышение ее производительности
   b. Вопросы интеллектуальной собственности - понимание степени влияния критических проблем
   c. Организация эффективных встреч - планирование и проведение
   d. Взаимодействие и общение - с разработчиками, руководством и другими командами
   e. Лидерство - обучение проектных команд для получения оптимальных результатов
   f. Управление изменениями - обеспечение эффективного управления изменениями
   g. Успешное ведение переговоров - разрешение конфликтов и ведение переговоров
   h. Планирование карьерного роста - структурирование и управление ходом реализации карьеры
   i. Эффективное представление - использование письменных и устных навыков
   j. Набор персонала - вербовка и собеседование с членами команды
   k. Отбор команды - высококомпетентных специалистов
   l. Создание команды - формирование, руководство и поддержка эффективной команды

Управление командой проекта

> Успех проекта напрямую связан с используемыми талантами, и, что более важно, способом, в соответствии с которым руководство использует эти таланты в проекте.
>
> *Джон Макдоналд*

Управление программным проектом включает решение трех основных задач: подбор и управление командой, выбор процесса и выбор инструментальных средств. Хотя все три задачи одинаково важны для успеха проекта, ведущую роль играет правильный подбор и управление командой. Успех проекта во многом зависит от того, насколько состав участников проекта сможет быть преобразован в команду единомышленников, насколько эта команда будет активной и инициативной с одной стороны и управляемой с другой. Из множества вопросов управления командой проекта мы рассмотрим три:

- Ролевая модель команды.
- Модели организации команд.
- Общение в команде.

Ролевая модель команды

Состав команды определяется опытом и уровнем коллектива, особенностями проекта, применяемыми технологиями и уровнем этих технологий. «Классический» вариант состава команды включает следующие позиции [10]:

- **Менеджер проекта** - главное действующее лицо, обладающее знаниями и навыками, необходимыми для успешного управления проектом. Его основные функции:
  - Подбор и управление кадрами
  - Подготовка и исполнение плана проекта
  - Руководство командой
  - Обеспечение связи между подразделениями
  - Обеспечение готовности продукта
- **Проектировщик** - это функция проектирования архитектуры высокого уровня и контроля ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. В больших проектах это может быть целый отдел. Основными функциями проектирования являются:

- Анализ требований
- Разработка архитектуры и основных интерфейсов
- Участие в планировании проекта
- Контроль выполнения проекта
- Участие в подборе кадров

- **Разработчик** – роль, ответственная за непосредственное создание конечного продукта. Помимо собственно программирования (кодирования) в его функции входит:
  - Контроль архитектурных и технических спецификаций продукта
  - Подбор технологических инструментов и стандартов
  - Диагностика и разрешение всех технических проблем
  - Контроль за работой разработчиков документации, тестирования, технологов
  - Мониторинг состояния продукта (ведение списка обнаруженных ошибок)
  - Подбор инструментов разработки, метрик и стандартов. Контроль их использования.
- **Тестировщик** – роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных). В функции тестировщика входит:
  - Составление плана тестирования. План тестирования составляет один из элементов проекта и составляется до начала реализации (разработки) проекта. Время, отводимое в плане на тестирование может быть сопоставимо с временем разработки.
  - Контроль выполнения плана. Важнейшая функция контроля – поддержка целостности базы данных зарегистрированных ошибок. В этой базе регистрируется: кто, когда и где обнаружил, описание ошибки, описание состояния среды; статус ошибки: приоритет, кто разрешает; состояние ошибки: висит, в разработке, разрешена, проблемы

    Эта база должна быть доступна всем, т.к. в тестировании принимают участие все члены команды.
  - Разработка тестов. Самая трудоемкая часть в работе тестировщика. Тестирование должно обеспечить полную проверку функциональности при всех режимах работы продукта.
  - Автоматизация тестирования включает автоматизацию составления тестов, автоматизацию пропуска тестов и автоматизацию обработки

результатов тестирования. В виду важности автоматизации тестирования, иногда вводят нового участника – инженера по автоматизации.

- Выбор инструментов, метрик, стандартов для организации процесса тестирования.
- Организация Бета тестирования - тестирования почти готового продукта внешними тестерами (пользователями). Эту важную процедуру надо продумать и организовать в случае разработки коробочного продукта.

- **Инженер по качеству.** В современном представлении рассматривается три аспекта (уровня) качества:
  - качество конечного продукта – обеспечивается тестированием,
  - качество процесса разработки (тезис: для повышения качества продукта надо повысит качество процесса разработки),
  - качество (уровень) организации (тезис: для повышения качества процесса надо повысить качество организации работ).

  В некоторых случаях функции инженера по качеству возлагаются на тестировщика. На самом деле они шире – два следующих уровня качества. Здесь приведены функции, отличные от функции тестировщика:
  - Составление плана качества. План качества включает все мероприятия по повышению качества (на всех уровнях). Имеет долговременный характер. План тестирования – его оперативная составляющая.
  - Описание процессов. Описание процессов является их формализацией. При описании вводятся метрики процесса, влияющие на качество продукта.
  - Оценка процессов включает регистрацию хода выполнения процессов и оценку значений установленных метрик процессов. Выявление «слабых» мест и выработка рекомендаций по улучшению процессов.
  - Улучшение процессов - переопределение процесса, автоматизация части работ, обучение персонала.

  Повышение качества процессов требует участия всех действующих лиц. Принятое решение должно быть обосновано, всем понятно и всеми принято. При повышении качества организации работа по улучшению процессов проводится по определенной схеме. На каждом шаге повышения уровня организации работ выделяются ключевые процессы и выполняются работы по улучшению этих процессов.

- **Технический писатель** или разработчик пользовательской (и иной) документации как части программного продукта. Функциями технического писателя являются:
    - Разработка плана документирования, который включает состав, сроки подготовки и порядок тестирования документов.
    - Выбор и разработка стандартов и шаблонов подготовки документов
    - Выбор средств автоматизации документирования
    - Разработка документации
    - Организация тестирования документации
    - Участие в тестировании продукта. Технический писатель все время работает с продуктом (его готовыми версиями) и выступая от имени пользователя видит все недочеты и несоответствия.
- **Технолог разработки ПО** обеспечивает выполнение следующих задач:
    - Поддержка модели ЖЦ - создание служб и структур по поддержке работоспособности принятой модели ЖЦ ПО. В поддержке модели ЖЦ принимают участие все. Но контроль возложен на технолога.
    - Создание и сопровождение среды сборки продукта. Функция особенно важна на завершающих этапах разработки или при использовании модели прототипирования. В такой ситуации сборка будет проводиться достаточно часто (в некоторых случаях - ежедневно). Среда сборки должна быть подготовлена заранее, сборка должна проводиться быстро и без сбоев. С учетом сборки версий это не простая задача.
    - Создание и сопровождение процедуры установки с тем, чтобы каждая сборка устанавливалась автоматически с учетом версии и конфигураций сред.
    - Управление исходными текстами - сопровождение и администрирование системы управления версиями исходных текстов.

Выделенные позиции на обязательно представлены конкретными людьми. Это список основных функциональных ролей в команде (ролевая модель команды). В малых командах роли могут совмещаться. В больших – выделяться группы или отделы (отдел проектирования, отдел тестирования, отдел контроля качества, отдел подготовки документации, …).

Состав команды определяется также типом выполняемых работ: под заказ или коробочное производство (продукт на рынок). Инженерный психолог и инженер по маркетингу нужны в последнем случае.

Следует отметить, что ролевые модели проектных команд могут быть самыми разнообразными. Ниже приведена ролевая модель команды, рекомендуемая в методологии MSF (Microsoft Solution Framework) [11]. Эта модель основана на шести ролевых кластерах, каждый из которых имеет свою цель, области компетенции и функции:

- **Управление продуктом (product management)**. Цель: удовлетворенные заказчики.

  Компетенции. Маркетинг. Бизнес-отдача (бизнес-приоритеты). Представление интересов заказчика. Планирование продукта

  Функции. Выступает в роли представителя заказчика. Формирует общее видение/рамки проекта. Организует работу с требованиями заказчика. Развивает сферы применения в бизнесе. Формирует ожидания заказчика. Определяет компромиссы между параметрами “возможности продукта / время / ресурсы”. Организует маркетинг, PR и евангелизацию. Разрабатывает, поддерживает и исполняет план коммуникаций.

- **Управление программой (program management).** Достижение результата в рамках проектных ограничений.

  Компетенции. Управление проектом. Выработка архитектуры решения. Контроль производственного процесса. Административные службы.

  Функции. Управляет процессом разработки с целью получения готового продукта в отведенные сроки. Формулирует спецификацию продукта и разрабатывает его архитектуру. Регулирует взаимоотношения и коммуникацию внутри проектной группы. Следит за временным графиком проекта и готовит отчетность о его состоянии. Проводит в жизнь важные компромиссные решения. Разрабатывает, поддерживает и исполняет сводный план и календарный график проекта. Организует управление рисками.

- **Разработка (development).** Цель: создание продукта в соответствии со спецификацией.

  Компетенции. Технологическое консультирование. Проектирование и осуществление реализации. Разработка приложений. Разработка инфраструктуры.

Функции. Определяет детали физического дизайна. Оценивает необходимые время и ресурсы на реализацию каждого элемента дизайна. Разрабатывает или контролирует разработку элементов. Подготавливает продукт к внедрению. Консультирует команду по технологическим вопросам.

- **Тестирование (test).** Цель: одобрение выпуска продукта только лишь после того, как все дефекты выявлены и улажены.

  Компетенции. Планирование тестов. Разработка тестов. Отчетность по тестам.

  Функции. Обеспечивает обнаружение всех дефектов. Разрабатывает стратегию и планы тестирования. Осуществляет тестирование.

- **Удовлетворение потребителя (user experience**). Цель: повышение эффективности пользователя, увеличение потребительской ценности продукта.

  Компетенции. Обеспечение технической поддержки. Обучение. Эргономика. Графический дизайн. Интернационализация. Общедоступность (обеспечение возможности работы для пользователей с ограниченными физическими возможностями).

  Функции. Представляет интересы потребителя в команде. Организует работу с требованиями пользователя. Проектирует и разрабатывает системы поддержки производительности. Определяет компромиссы, относящиеся к удобству использования и потребительским качествам продукта. Определяет требования к системе помощи и её содержание. Разрабатывает учебные материалы и осуществляет обучение пользователей.

- **Управление выпуском (release management).** Цель: беспроблемное внедрение и сопровождение продукта.

  Компетенции. Инфраструктура. Сопровождение. Бизнес-процессы. Управление выпуском готового продукта.

  Функции. Представляет интересы отделов поставки и обслуживания продукта. Организует снабжение проектной группы. Организует внедрение продукта. Вырабатывает компромиссы в управляемости и удобстве сопровождения продукта. Организует сопровождение и инфраструктуру поставки. Организует логистическое обеспечение проектной группы.

Модели организации команд

Peopleware – человеческий фактор

Как организовать работу команды? Команды из 10 человек и команды из 500 человек? Есть ли различия и в чем они состоят? Надо ли организовывать работу по жесткой технологии или надо предоставить свободу действий? Можно ли найти методологию (технологию) выполнения проекта, обеспечивающую успех?

Алистер Коубен [12] – специалист в области технологий выполнения ИТ проектов приводит данные 23 проектов различной степени сложности, выполнявшихся по различным технологиям и имеющие различные результаты. Пытаясь проанализировать результаты применения различных технологий в тех или иных условиях, он приходит к выводу:

- Практически любую методологию можно с успехом применять в каком-нибудь проекте.
- Любая методология может привести к провалу проекта.

Главную причину он видит в том, прямо перед нами всегда находится нечто, чего мы не замечаем: люди. Именно человеческие качества обеспечивают успех тому или иному проекту, именно они являются фактором первостепенной важности, основываясь на котором надо строить прогнозы о проекте.

Исследованию вопросов человеческого фактора (Peopleware) уделяется достаточно много внимания. Наиболее известными работами являются [13-15] и др.

Проблемы человеческого фактора связаны с тем (проявляются в том), что участвующие в проекте люди:

- Все разные – по характеру, темпераменту, активности, целям – нет двух одинаковых людей.
- Все похожие – участие в проекте объединяет людей общностью целей, поиском путей достижения этих целей.
- Различаются по типу: индивидуалисты - члены команды; генераторы идей - исполнители; ответственные – безответственные.
- Постоянны и изменчивы – люди, как правило, проявляют постоянство своих привычек и свойств характера, но при этом способны проявлять «противоположные» качества: индивидуалист – командные качества, исполнитель – генерировать идеи, …

- Многообразны – надо понимать, что многообразие людей является основной гарантией выживания человечества вообще и возможности выполнять ИТ проекты в частности. Если бы все были индивидуалисты или все командники, все генераторы идей или все исполнители, то вряд ли удалось выполнить хотя бы один проект, а мир стал бы ужасен.

Как же управлять такими людьми? В работе [16] приводится обзор трех основных моделей управления командой

Административная модель (теория X)

Это традиционный стиль управления, связанный с иерархической административно-командной моделью, которую используют военные организации. В основе лежит теория X, которая утверждает, что такой подход необходим, поскольку большинство людей по своей природе не любит работу, и будет стремиться избежать ее, если у них есть такая возможность. Однако менеджеры должны принуждать, контролировать, направлять сотрудников и угрожать им, чтобы получить от них максимальную отдачу. Девиз теории и модели: Люди делают только то, что вы контролируете. Или в более мягком варианте: Люди делают то, что они не хотят делать, только если вы их контролируете. В конце концов, теория утверждает, что большинство людей предпочитают, чтобы им говорили, что следует делать и им не придется ничего решать самим.

Характерные черты модели:

- Властная пирамида – решения принимаются сверху-вниз.
- Четкое распределение ролей и обязанностей.
- Четкое распределение ответственности.
- Следование инструкциям, процедурам, технологиям.
- Роль менеджера: планирование, контроль, принятие основных решений.

Преимущества модели: ясность, простота, прогнозируемость. Модель хорошо сочетается с каскадной моделью жизненного цикла и применима в тех же случаях, что и каскадная модель. Модель эффективна в случае установившегося процесса.

Недостатки модели связаны с тем, что административная система стремится самосохранению (стабильности) и плохо восприимчива к изменению ситуации – новые типы проектов, применение новых технологий, оперативная реакция на изменение рынка. Кроме того, в административной модели плохо уживаются индивидуалисты и генераторы идей.

Административная система (модель) – это тяжелый паровоз, идущий в «середине» и не поддающийся на «крайности» поиска новых путей и решений. Она воспринимает новые решения и технологии, но только проверенные, отработанные и стандартизированные. В этом ее сила, слабость и проявление принципа многообразия. Видимо, именно к ней в наибольшей степени применим термин «промышленное программирование».

Модель хаоса (теория Y)

В основе модели хаоса лежит Теория Y, которая является полной противоположностью Теории X. Основной тезис Теории Y: работа — естественная и приятная деятельность и большинство людей, на самом деле, очень ответственны и не увиливают от работы.

Характерными чертами модели хаоса являются:

- Отсутствие явно выраженных признаков власти.
- Роль менеджера – поставить задачу, обеспечить ресурсами, не мешать и следить, чтобы не мешали другие.
- Отсутствие инструкций и регламентированных процедур.
- Индивидуальная инициатива - решения по проблеме принимается там, где проблема обнаружена.
- Процесс напоминает творческую игру участников на основе дружеской соревновательности.

Преимущества такой модели в том, что творческая инициатива участников ничем не связана и потенциал участников раскрывается в полной мере. Это бывает особенно эффективно в случае, когда для решения проблемы требуется поиск новых подходов, методов, идей и средств. Команда становится командой «прорыва», а работа проходит в форме игры, цель которой – поиск наилучшего результата. Процесс напоминает случайный поиск, когда идеи и решения рождаются при живом и как бы случайном обсуждении проблем в коридоре, столовой на пикнике. Собрать такую команду в рабочей комнате и устроить обсуждение по регламенту часто просто не удается – это команда творческих индивидуалистов.

Недостатки модели связаны с тем, что при определенных условиях команда прорыва может стать командой провала. Причинами провала могут быть:

- Творческая соревновательность переходит в конкуренцию сначала идей, а потом - личностей.

- Процесс начинает преобладать над целью проекта – высказанные идеи не доводятся до конца и сменяются новыми идеями, преобладание получают «красивые» идеи, лежащие в стороне от основных целей проекта.
- Люди, способные к генерации идей, редко обладают терпением доведения идей до полной реализации.

Модель хаоса – это то, что нужно для освоения новых земель. Модель хаоса не противоречит административной модели – она ее дополняет и может эффективно с ней соседствовать (но в разных комнатах!). Многие мускулистые корпоративные бегемоты полагаются на исследовательские «отделы скунсов», откуда они черпают новые идеи, технологии и продукты.

Открытая архитектура (теория Z)

Административная и хаотическая модели являются двумя «крайностями», между которыми находятся множество моделей, сочетающих преимущества «крайних» моделей. Одной из таких моделей является модель открытой архитектуры, основанная на Теории Z. Эта теория была сформулирована Уильямом Оучи на основе изучения опыта японского стиля управления (Theory Z: How American Business Can Meet the Japanese Challenge,» Perseus Publishing, 1981). Теория Z предполагает (но не декларирует) наличие внутреннего механизма управления, основанного на влиянии со стороны коллег и группы в целом. Дополнительное воздействие оказывают культурные нормы конкретной корпорации.

Основной принцип модели можно сформулировать так: «Работаем спокойно. Работаем вместе». Особенностями этой модели являются:

- Адаптация к условиям работы – если делаем независимые модули, то расходимся и делаем, если нужна архитектура базы данных, то собираемся вместе и обсуждаем идеи.
- Коллективное обсуждение проблем, выработка консенсуса и принятие решения – не все могут согласится, но принятое решение является коллективным и в силу этого – обязательным для всех.
- Распределенная ответственность – отвечают все, кто обсуждал, вырабатывал, принимал.
- Динамика состава рабочих групп в зависимости от текущих задач.
- Отсутствие специализации – участники меняются ролями и функциями и могут при необходимости заменить друг друга.

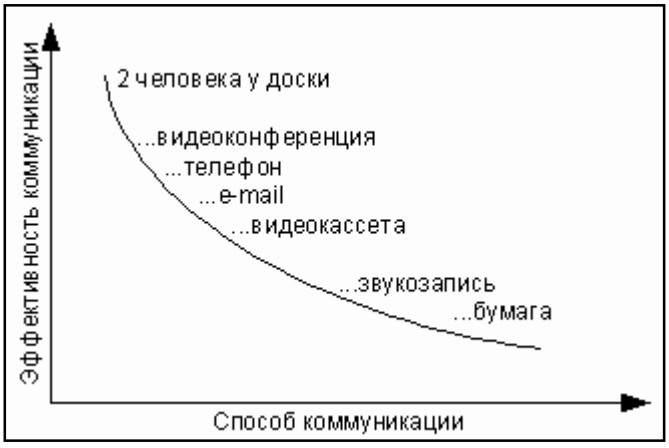
- Задача менеджера – активное (но рядовое, не руководящее) участие в процессе, контроль конструктивности обсуждений, обеспечение возможности активного участия всех.

Открытая архитектура является более гибкой, адаптируемой, настраиваемой на ситуацию. Она дает возможность проявить себя всем членам команды – в ней могут уживаться и индивидуалисты и коллективисты. Коллективное обсуждение высказанных идей позволяет оставлять только прагматичные идеи.

Общение в команде

Коммуникации

Основным фактором в разработке программного обеспечения является возможность коммуникации (общения участников проекта). Общение может проводиться в различных формах от строго формализованного (стандартизированная документация) до полностью неформализованного (вопрос-ответ соседу, обсуждение в неформальной обстановке).



На рисунке изображена некая кривая, иллюстрирующая эффективность различных способов общения [12]. Кривая является обобщением ряда исследований в этой области. Видно, что эффективность общения падает по мере возрастания степени его формализованности. С одной стороны, в этом нет ничего удивительного – старый принцип бюрократа гласит: хочешь получить отказ – пиши письмо, хочешь получить обещание – звони по телефону, хочешь добиться результата – езжай сам.

Но с другой стороны, надо помнить, что коммуникации в команде определяются количеством участников (рабочих связей): при двух участниках – это одна связь, при n участниках – n(n-2)/2. При этом, любая из этих связей может давать сбои и они не транзитивны: из того, что участник А хорошо контактирует с Б, а Б – с В вовсе не следует, что А контактирует с В. Т.е. неформальное, «живое» общение эффективно только в относительно небольших, хорошо организованных (сработавшихся) коллективах.

В [12] можно найти интересный анализ динамики связей отмеченных на приведенном рисунке.

Принятие решений – компромисс и консенсус

Целью общения в команде разработчиков являются обсуждение текущих проблем и вопросов и принятие решений. Далее мы рассмотрим некоторые проблемы организации обсуждений и принятия решений. Начнем с принятия решений.

Итак, принятое в результате обсуждения решение может быть достигнуто в результате компромисса или в результате консенсуса. В чем разница этих результатов?

Начнем с определений (Глоссарий.ру): **Компромисс** - соглашение, достигнутое посредством взаимных уступок. **Консенсус** (коллективное мнение) - общее для конкретной группы мнение. В чем же разница?

Компромисс:

- Это среднее решение, которое может оказаться (и, как правило, оказывается) хуже каждого из вариантов.
- Достигается путем взаимных уступок (мы согласимся с вашим вариантом интерфейса, если вы согласитесь с нашей организацией базы данных).
- Может быть принят большинством (голосованием).

Консенсус:

- Это оптимальное решение, сочетающее лучшее из предложенных вариантов.
- Достигается путем обсуждения, анализа и генерации новых идей.
- Принимается общим согласием (все согласны, что найдено лучшее решение).

Л. Константин [13] приводит следующий пример компромисса и консенсуса. При обсуждении вопроса о размещении кнопок панели инструментов выдвинуты два варианта: горизонтально и вертикально. Компромисс – по диагонали (нелепое решение). Консенсус – настраиваемая пользователем панель (лучшее решение, включающее оба варианта на основе новой идеи – настраиваемая панель).

Как добиться консенсуса?

В отличие от компромисса, который чаще всего достигается в результате политических интриг и подковерных баталий, достижение консенсуса требует конструктивного и плодотворного напряжения всей команды и особого искусства управления командой. При этом рекомендуется придерживаться следующих принципов и правил:

Вера в достижение консенсуса – каждый член команды должен доверять другим в том, что обсуждение приведет к поиску оптимального решения, а не к борьбе личностных мнений. Создание такой атмосферы взаимного доверия является важнейшим в создании

эффективной команды. Следует понимать, что взаимное доверие появляется не само по себе, а является результатом:

- Нескольких удачных консенсусов
- Участием всех в выработке и принятии оптимальных решений
- Созданием у каждого осознания причастности к принятым решениям

Не позиция, а варианты решений – на обсуждение люди должны приходить не со сформированной позицией (ни шагу назад), а с вариантами возможных решений

Объективность принимаемых решений как попытка ограничить проявления чувств и эмоций при обсуждении вопросов. Чувства и эмоции являются неотъемлемым свойством человеческой природы. Избежать их полностью вряд ли удастся, но для приведения их «в норму» можно использовать следующие правила:

- Критерии оценки вариантов – для объективности обсуждения крайне важно заранее договориться о критериях оценки – установить список критериев и выполнить их ранжировку по степени важности.
- Разделение фактов и мнений. Факты – объективные показатели, выраженные в большинстве случаев количественно (но не обязательно): быстродействие, время отклика. Мнения – то, что не основано на фактах. Мнениями не следует пренебрегать, т.к. они часто основаны на опыте, интуиции.

Замена позиций – в случае, когда обсуждение все же заходит в тупик, бывает полезно предложить участникам изменить точку зрения: «перечислите, пожалуйста, сильные стороны варианта Вашего оппонента и слабые стороны Вашего варианта».

Слегка управляя - роль руководителя в достижении консенсуса состоит в том, чтобы дать всем возможность высказаться и предложить свои варианты, оставляя свое мнение напоследок или не высказывать его совсем. Руководитель должен быть нейтрален. Руководитель (лидер) может принимать активное участие в обсуждении, но только на правах равного и поручить в этом случае руководство собранием другому человеку.

Корпоративная политика (наведение мостов)

Представим себе ситуацию, когда одна команда разрабатывает коробочный проект. Проект идет успешно. Даже блестяще: подобралась слаженная команда профессионалов, было найдено красивое архитектурное решение, учитывающее возможность широкого изменения требований, разработан оригинальный интерфейс, успешно использовано большое количество ранее созданных компонент и т.д. и т.д. Но финансирование проекта было прекращено руководством фирмы. Блестящий проект был признан

бесперспективным. Попытки выяснить «истинные» причины успеха не имели. Активным выяснятелям намекнули, что они могут попасть под очередное сокращение кадров.

Что делать в такой ситуации?

Вариант первый – продолжить выполнение проекта в другой обстановке. Например, создать собственную фирму. Отличная идея, но здесь надо быть готовым к ответам на несколько вопросов:

- Деньги на … (на что нужны деньги?). Можно взять кредит, но каков процент и когда мы сможем его погасить?
- Продвижение продукта на рынок:
  - Нужна реклама, а это немалые деньги (плюс к первому вопросу).
  - Репутация фирмы? – молодым фирмам не очень доверяют. Компенсировать можно только усиленной рекламой.
- Конкуренты. Кто работает в этой нише и что от них можно ожидать? Что можно противопоставить конкурентам?
  - Перспективную в плане развития продукта архитектуру? Это далекая перспектива – кредит надо будет возвращать раньше.
  - Оригинальный интерфейс? А если рынок уже привык к стандартным решениям конкурентов?
  - Снижение цены за счет применения готовых компонент? Но теперь за компоненты надо платить.

Вариант второй – научиться играть в корпоративную политику. Что это такое? Начнем с того, что анализ вопросов, возникающих при создании собственного бизнеса, делает решение фирмы о прекращении проекта более прозрачным:

- У нас перспективная архитектура? А мы объяснили это в отделе стратегического планирования? Нет!
- У нас оригинальный интерфейс? А мы сходили к ребятам в недавно созданный отдел People Ware для его оценки? Нет – вместо этого мы много иронизировали по поводу их деятельности.
- Да, цену продукта можно снизить за счет применения готовых компонент нашей фирмы. Но это снижение прибыли фирмы, уже заложенной в ее финансовый план. Пытались мы убедить плановый отдел, что это снижение компенсируется в будущем за счет перспективной архитектуры нашего продукта? И опять же – нет!

Т.е. вы живете не на острове. Ваша фирма (корпорация) – это среда, в которой существует ваш проект и от которой во многом зависит его успех. Да, вы можете создать нечто гениальное. И потомки это оценят (может быть). Но ваша цель все-таки в другом – продать продукт сейчас.

Корпоративная политика – это внешние стратегии команд по учету влияния и воздействию на внешнюю среду вашего проекта.

Корпоративная политика – это не только умение лидера проекта ладить с начальством. Это еще умение всей команды взаимодействовать с другими подразделениями фирмы. В [13] выделяется три измерения, в которых происходят эти взаимодействия: во властной структуре, в структуре задач и в информационной структуре.

Взаимодействие во властной вертикали – это создание репутации, получение поддержки со стороны руководства, получение лучших проектов, оборудования и софта, «прикрытие» от политических бурь. Человек, выполняющий все это должен быть политиком, ориентирующимся в кабинетах власти фирмы.

Координация задач выполняется в горизонтальной плоскости и состоит во взаимодействии с другими подразделениями фирмы. Координаторы обеспечивают поступление проекта и его сдачу, взаимодействие с внешними тестерами, изучение интерфейса с группой анализа человеческого фактора, получение и передачу библиотек компонентов, договариваются с другими группами, выторговывая ресурсы и услуги.

Взаимодействие в информационной структуре предполагает исследование и сбор информации, необходимой для успешного выполнения проекта. Информационные исследователи ищут нужное и просеивают поступающее.

В разных командах складываются разные стили игры в корпоративную политику. Чему следует отдавать предпочтение? Проводились исследования эффективности команд четырех типов: политики, изоляционисты, исследователи и универсалы. В начале исследования политики и изоляционисты считали себя лучше других, хотя с точки зрения руководства лучшими выглядели политики и универсалы. Через полгода оказалось, что самые низкие оценки производительности – у политиков и исследователей (первые много говорили, вторые много собирали информации, но и те и другие мало что делали). Далее шли изоляционисты. Здесь результаты были неоднозначны. Часть команд пришли к полному провалу, часть получили ощутимые результаты. Лучшие результаты были у универсалов.

См. также: Алексей Кайдалов. Команда ИТ-проекта: как избежать проблем. (http://www.silicontaiga.ru/home.asp?artId=3637)

Планирование и контроль

Зачем надо планировать?

Срок завершения небрежно сверстанного проекта в три раза превышает запланированный.

Срок реализации тщательно спланированного проекта превышает установленный в два раза.

Законы управления проектами.

http://www.nwsta.com/Soft/proj/pm01.php

На самом деле: зачем планировать, если запланированные сроки все равно срываются, запланированных ресурсов все равно не хватит, предусмотренный бюджет будет трещать по швам? Стоит ли на планирование тратить время и средства? Стоит потому, что:

- Вы должны убедить Заказчика в том, что с вами можно иметь дело. Как? Наличие плана является одним из наиболее весомых аргументов.
- Проект должен быть предсказуемым. Как сделать его предсказуемым? План – один из основных элементов предсказуемости проекта. Контроль хода выполнения плана позволяет оценить возможность продолжения проекта.
- Проект имеет элемент неопределенности. Т.е. заранее всего предусмотреть нельзя, в силу чего первоначальные планы обычно и не выполняются. Но при возникновении ранее неучтенных обстоятельств, планы можно и нужно корректировать. Для сохранения предсказуемости проекта. ***План – ничто. Планирование – все!***

Задачи планирования

Основными функциями планирования являются:

- Преобразование потребностей в управляемые задачи. Изначально проект выступает в виде требований, разработанных и согласованных с Заказчиком. Цель планирования – представить его в виде совокупности отдельных задач, выполнение которых можно контролировать.
- Определение необходимых ресурсов. Детальные планы позволят Вам определить количество людей, необходимого оборудования и рабочие условия, которые понадобятся для выполнения проекта

- Координация командной работы над проектом. Очень часто выполнение проекта разбивается на отдельные работы, которые можно выполнять параллельно. Планы делают возможной координацию путем определения того кто, что и когда делает.
- Оценка потенциальных рисков. Хотя некоторые риски могут быть выявлены во время формулировки требований, гораздо больше их обнаруживается после осуществления детального планирования. Знание о существовании этих рисков позволит Вам раньше их заметить (если они осуществились) и приготовиться к их адресации.
- Сигнализация о возникновении проблем. Отклонение от плана – сигнал о возникновении проблемы. Планы – это не догма, которой необходимо безоговорочно следовать. Для менеджера проекта они скорее являются предположениями и основой для сравнения. Если выполнение проекта не оправдывает ожиданий, то необходимо провести соответствующую корректировку плана.

Что надо планировать?

При планировании выполнения проекта надо найти ответы на следующие вопросы:

- Что и как надо сделать? Определение целей проекта, стратегии достижения целей, выделение задач.
- Когда это надо сделать? Составление графика выполнения отдельных задач
- Сколько будет это стоить? Планирование бюджета по отдельным задачам и статьям расхода.
- Кто это должен сделать? Планирование ресурсов, распределение ролей и ответственности.
- Насколько хорошо это надо сделать? Планирование качества.
- Что может помешать? Планирование рисков.
- Как проверять и оценивать? Определение метрик проекта.

В относительно небольших проектах план может быть единым. В больших проектах могут составляться планы по отдельным видам работ (процессам): план тестирования, план документирования, план управления качеством, финансовый план и т.д. При наличии нескольких планов составляется также основной план (мастер-план), в котором отражены основные показатели выполнения проекта в целом: основные (без детализации) виды работ, сроки, ресурсы, финансирование.

Как проверять и оценивать?

Прежде всего, определим, что надо проверять и оценивать: общий ход выполнения проекта; выполнение отдельных видов работ; работу отдельных исполнителей, и т.д.

Проверять и оценивать можно по-разному:

- Мы довольны ходом и результатами, потому, что мы умны, нам интересно, …
- Заказчик доволен.
- Заказчик согласен оплатить очередной этап.
- Заказчик недоволен, но он просто ничего не понимает в компонентном программировании.
- Тестирование выполняется (не) нормально потому, что тестирует (не) хороший человек.

Все это – качественные оценки хода выполнения проекта, которые далеко не всегда являются объективными.

Метрики проекта

Объективно оценить и проконтролировать можно только то, что можно измерить. Для объективной оценки необходимо вводить метрики проекта – количественные показатели оценки различных характеристик проекта и процесса его выполнения. Метрики могут вводиться как для всего проекта в целом, так и для отдельных видов работ. Общими метриками проекта являются:

- Количество фаз / действий / работ.
- Продолжительность каждой работы.
- Стоимость ресурсов, стоимость работы, общая стоимость.
- Степень загрузки ресурсов и исполнителей на отдельных этапах.
- Количество завершенных работ.
- Количество изменений в проекте.
- Задержки выпуска.
- Стоимость изменения требований.

Как надо планировать?

Когда начинать планировать?

Когда надо начинать планировать? В самом начале проекта? Когда сформулированы требования и ясен объем работ? Когда выполнение проекта выходит из под контроля и проект надо «ввести в берега»? Начнем с того, что «правильного» ответа на этот вопрос не

существует: есть проект и есть проект. Если у вас небольшая, слаженная команда профессионалов, то … (см. технологию XP, где планирование сведено к разумному минимуму). Если у вас большой проект, большая команда, ограниченные сроки, то планировать придется. И когда начинать?

Иногда разумно начинать планировать и после формулировки требований, если у вас относительно небольшой проект, относительно небольшие ресурсы и есть опыт выполнения аналогичных проектов.

Если проект сложный, много распределенных ресурсов, то планировать надо начинать с самого начала проекта. Могут спросить: как планировать то, что пока еще не известно? Ведь в начале проекта мы еще не знаем даже того, что нам предстоит сделать? Здесь следует вспомнить, что план ИТ проекта – это не догма. Планирование –циклический это процесс составления, оценки и корректировки плана. В сложных случаях этот процесс надо запускать как можно раньше.

Структурная декомпозиция работ

Важнейшим элементом планирования является разбиение проекта на отдельные задачи, подзадачи и действия с дальнейшей оценкой сроков, ресурсов и порядка их выполнения. Этот элемент планирования называют структурной декомпозицией работ (СДР, или WBS - Work Breakdown Structure). СДР – это иерархическая декомпозиция и организация деятельностей (задач, подзадач, действий), необходимых для удовлетворения целей проекта. Организация и уровень детализации деятельности будут способствовать оценке, распределению работ и дальнейшему управлению.

СДР помогает сделать цели проекта управляемыми. Хотя проект может состоять всего из нескольких сотен задач, но уже ими практически невозможно будет руководить, если они будут находиться в одной куче. СДР служит идее организации задач с целью упрощения работ по оцениванию, распределению, координированию и пересмотру.

На деятельностях, определенных в СДР базируются планы проекта, включая:

- Календарный план-график проекта.
- План распределение ресурсов.
- Бюджетный план.
- План управления качеством.
- План управления рисками.

Создание СДР

Ниже перечислены основные шаги процесса, которому можно следовать при построении СДР:

- Определите основные цели проекта.
- Определите функциональные требования, которые удовлетворяют целям проекта.
- Определите основные задачи, соответствующие функциональным требованиям. Чтобы достигнуть более высокой управляемости проекта и убедиться в том, что вы включили все существенные задачи, часто полезно включить промежуточный уровень классификации. Можно систематизировать задачи, используя предлагаемые уровни группировки или их комбинации:
  - системы (основные аппаратные и программные подсистемы);
  - этапы или фазы (как концепция, инициация, проектирование, разработка, сборка и тестирование);
  - организации (отделы и географические дислокации).
- Подразделяйте основные задачи на более мелкие, которые будут отражать то, каким образом планируется завершить работу.
- Составьте графическую схему, создавая столько слоев, сколько необходимо для полного разбиения объема работ на достаточно малые управляемые части с уровнем детализации, который позволяет:
  - оценивать работы и определять их временные рамки;
  - назначать работы исполнителям (группам);
  - видеть и обсуждать продвижение работ.

Критерии СДР

Для достижения поставленных целей (оценка, распределение и контроль выполнения работ) СДР должна удовлетворять следующим критериям:

- Целенаправленность. Все деятельности должны быть направлены на достижение единой цели проекта и вести к конечному результату.
- Независимость. Между деятельностями в рамках проекта очень часто существует множество зависимостей. Управлять и контролировать такие деятельности достаточно сложно. Для повышения управляемости проектом любая деятельность должна быть определена до такого уровня детальности, что она будет завершена без необходимости активной координации с результатами

других деятельностей. Такой ситуацией лучше всего управлять путем разбиения работы на равнозависимые подмножества.

- Определенность продолжительности. Деятельности не должны быть «безлимитными» во времени, так как в этом случае они непременно растянутся, выходя за пределы самых худших ожиданий. Длительность также косвенно устанавливает ожидаемое качество результата.
- Четкость понимания. Деятельность должна предполагать результат, однозначно понимаемый людьми, которые будут выполнять эти работы. Результатом может быть замысел, решение, документ, тест и т.д. Результат должен быть представлен в четко понимаемой форме (общее описание, документ по шаблону, исходный код в соответствии с принятыми правилами оформления, …).
- Достижимость. Планируемый результат должен быть достижим: установленные сроки, выделяемые ресурсы, квалификация исполнителей, организация работ должны быть реальны и достаточны для планируемых результатов отдельных деятельностей с учетом предполагаемого уровня качества.
- Отработанность. Большинство разрабатываемых проектов сопровождаются созданием чего-то нового. Тем не менее, работы, которые приводят к этому созданию, в основном уже проводились ранее (возможно немного по другому, чем это понадобится для нового проекта). Отработанность детальных задач способствует оценке и распределению работ.

Средства управления проектом

Компьютерная система управления проектом обеспечивает доступ к информации минуя бюрократические и географические барьеры. Участники проекта смогут иметь доступ к проектной информации в режиме реального времени, даже если они находятся далеко друг от друга. Система позволяет получить общее представление обо всех проектах, которые имеются в портфеле проектов, наглядно отображает взаимосвязи между задачами, позволяет вовремя заметить проблемы. Средства визуального отображения позволяют организовать обмен информацией между членами проектной команды и клиентами.

Если менеджер проекта не силен в теории управления проектами, то обязательно возникнет вопрос целесообразности использования системы для управления проектами. Программный продукт может только помочь, но решать проблемы будут люди. Кроме

того, программный продукт может реально помочь лишь в том случае, когда в компании разработаны стандарты, методология, инструкции по обучению и использованию программного продукта.

Рынок программных продуктов для управления проектами растет и развивается. Системы все в большей степени ориентированы на Интернет. Такие системы позволяют обеспечить доступ к проектной документации для всех членов команды в режиме реального времени. В будущем все больше организаций будут использовать программные продукты. Ожидается, что в будущем системы от различных производителей будут все более похожи друг на друга. Это объяснятся тем, что базовая основа всех систем календарного планирования одна и та же. Кроме того, пользователи хотят видеть проектные данные в привычном и понятном им формате.

Функции систем управления проектами

Инструментальные средства управления проектом должны поддерживать следующие основные функции:

- Комплекс работ, связей и временных характеристик. Средства описания комплекса работ проекта, связей между работами и их временных характеристик должны включать:
    - Описания глобальных параметров планирования проекта
    - Описание логической структуры комплекса работ
    - Многоуровневое представление проекта
    - Назначение временных параметров планирования задач
    - Поддержка календарей отдельных задач и проекта в целом
- Информация о ресурсах и затратах. Средства поддержки информации о ресурсах и затратах по проекту и назначения ресурсов и затрат отдельным работам проекта должны обеспечивать решение следующих задач:
    - Организационная структура исполнителей
    - Ведение списка наличных ресурсов, номенклатуры материалов и статей затрат
    - Поддержка календарей ресурсов
    - Назначение ресурсов работам
    - Календарное планирование при ограниченных ресурсах
- Контроль за ходом выполнения. Средства контроля за ходом выполнения проекта должны обеспечивать:

  - Фиксацию плановых параметров расписания проекта в базе данных
  - Ввод фактических показателей состояния задач
  - Ввод фактических объемов работ и использования ресурсов
  - Сравнение плановых и фактических показателей и прогнозирование хода предстоящих работ
- Представление структуры проекта, отчетов. Графические средства представления структуры проекта, средства создания различных отчетов по проекту в виде:
  - Диаграмма Гантта (часто совмещенная с электронной таблицей и позволяющая отображать различную дополнительную информацию)
  - PERT диаграмма (сетевая диаграмма)
  - Создание отчетов, необходимых для планирования и контроля
- Дополнительные программные продукты. “Классические” системы календарного планирования, в последнее время, дополняются программными продуктами, которые позволяют:
  - добавить или улучшить отдельные функции управления проектами, например, анализ рисков, учет рабочего времени исполнителей, расчет расписания при ограниченных ресурсах;
  - интегрировать системы управления проектами в корпоративные управленческие системы;
  - настроить универсальное программное обеспечение на специфику управления проектами в конкретной предметной области (например, интеграция со сметными системами для строительных проектов).

УПРАВЛЕНИЕ КАЧЕСТВОМ ИТ ПРОЕКТА

Качество продукта и качество процесса

Качество - это свойство товара (услуги) наиболее полно удовлетворять требованиям и пожеланиям потребителя. Введенное понятие качества есть «качественный» показатель продукта. Можно ли качество измерить? Наиболее общим является подход, при котором вводятся понятия:

- Ценность изделия - способность удовлетворять потребности.
- Качество изделия - соответствие между свойствами изделия и его ценностью.
- Мера качества - соотношение ценности и стоимости.

Исторически обеспечение качества продукта складывалось как контроль (отбраковка) готового продукта. Принципы такого подхода сложились в средние века вместе с зарождением ремесленного производства и продолжали действовать до конца XIX века, когда ремесленное производство стало заменяться промышленным. Одним из элементов этого перехода стало применение контроля комплектующих изделий до окончательной сборки продукта.

Следующим важным шагом в обеспечении качества стал переход к управлению качеством, цель которого состояла не в том, чтобы обнаружить и изъять негодные изделия до их отгрузки покупателю, а в том, чтобы увеличить выход годных изделий в процессе производства. Основными этапами такого перехода стали:

- Управление процессами - переход от контроля к управлению отдельными процессами.
- Управление производством – переход от управления отдельными процессами к управлению производством в целом.

Главная идея перехода от контроля к управлению качеством состояла в том, что качество продукта должно обеспечиваться качеством процесса его изготовления.

В 1957 г. Фейгенбаум опубликовал статью, в которой изложил 8 принципов TQM (Total Quality Management) тотального управления качеством и параллельного (одновременного) инжиниринга. Эти принципы лежат в основе современных систем управления качеством:

1. Ориентация организации на потребителя (Customer-Focused Organization).

   Организации зависят от своих потребителей и, таким образом, должны понимать текущие и будущие потребности потребителей, удовлетворять их требования и стремиться превзойти их ожидания.

2. Лидерство (Leadership)

   Лидеры организаций обеспечивают единство назначения и направления организации. Они должны создать и поддерживать внутреннюю окружающую среду, в которой люди могут в полной мере участвовать в достижении стратегических целей организации.

3. Вовлечение персонала (Involvement of People)

   Люди составляют сущность организации на всех уровнях, и их полная вовлеченность способствует применению их способностей на благо организации.

4. Процессный подход (Process Approach)

   Желаемый результат достигается более эффективно, когда связанные ресурсы и деятельность управляются как процесс.

5. Системный подход к административному управлению (System Approach to Management)

   Выявление, понимание и административное управление системой взаимосвязанных процессов для заданной стратегический цели повышает эффективность и результативность организации.

6. Непрерывное усовершенствование (Continual Improvement)

   Непрерывное усовершенствование должно быть постоянной стратегической целью организации.

7. Основанный на фактах подход к принятию решений (Factual Approach to Decision Making)

   Эффективные решения базируются на анализе данных и информации.

8. Взаимовыгодные отношения с поставщиками (Mutually beneficial supplier relationship)

   Организация и ее поставщики взаимозависимы, и взаимовыгодные отношения повышают способность обоих производить ценности.

Подробнее: Восемь принципов, которые меняют мир. http://www.m2bc.ru/qs_8principles

ISO9000: система управления качеством

Представленные выше принципы TQM составляют методологическую основу (фундаментальные требования) ISO 9000 – серии международных стандартов, регламентирующих организацию системы управления качеством. Стандарты серии ISO 9000 универсальны, т.е.применимы к любым предприятиям независимо от сферы деятельности, формы собственности, размеров предприятия. Первая версия ISO 9000 вышла в 1987, последняя (третья) – в 2000 г. Версия ISO 9000 2000 года содержит 5 базовых стандартов. Позже стали выходить различные методические руководства по применению ISO 9000:2000 к различным видам деятельности. В частности, для применения в программной инженерии вышел стандарт **ISO/IEC 90003**, Guidelines for the Application of ISO 9001:2000 to Computer Software, 2004.

Подробнее: Сущность стандартов ISO. http://www.in4business.ru/?IDA=25

ISO9000. Структура документов СК

Система управления качеством поддерживается следующей структурой документов:

- Заявление о политике и целях в области качества.
- Руководство по качеству.
- Документированные процедуры, требуемые настоящим международным стандартом.
- Документы, необходимые организации для обеспечения эффективного планирования и осуществления процессов и управления ими (положения о подразделениях, должностные инструкции, регламенты, технологические инструкции ..).
- Записи о качестве.

Заявление о политике и целях должно представлять небольшой по объему документ (1-2 листа), отражающий следующие позиции:

- Общие намерения и направления деятельности организации в области качества, официально сформулированные высшим руководством.
- Не должна быть просто декларацией - цели должны быть конкретными, достижимыми, проверяемыми
- Определять приоритеты, поставить 3-4 цели. Не надо все перечисленные направления включать в "Политику". Высшее руководство должно определить

приоритеты, поставить 3-4 цели. Это не значит, что по остальным направлениям ничего не будет делаться, просто заявленные в "Политике" цели будут первоочередными, наиболее важными.

- Добиваться поставленных целей. При этом высшее руководство не только должно подписать "Политику", что является обязательным, но действительно добиваться этих целей.

Руководство по качеству – более детальный документ (20-50 листов), содержащий описание всей системы качества в целом. Структура этого документа должна соответствовать структуре стандарта ISO-9001 или должна быть представлена таблица соответствия разделов руководства и стандарта. Структура стандарта ISO-9001 включает:

- Система менеджмента качества.
  - Общие требования. Требования к документации.
- Ответственность руководства
  - Обязательства руководства. Ориентация на потребителя. Политика в области качества. Планирование. Ответственность, полномочия и обмен информацией. Анализ со стороны руководства.
- Менеджмент ресурсов
  - Обеспечение ресурсами. Человеческие ресурсы. Инфраструктура. Производственная среда
- Процессы жизненного цикла продукции
  - Планирование процессов жизненного цикла продукции. Процессы, связанные с потребителями. Проектирование и разработка. Закупки. Производство и обслуживание. Управление устройствами для мониторинга и измерений
- Измерение, анализ и улучшение
  - Общие положения. Мониторинг и измерение. Управление несоответствующей продукцией. Анализ данных. Улучшение

При внедрении систем качества значительная роль отводится документации. Значимость документации проявляется в нескольких критических случаях, таких как: достижение требуемого уровня качества продукта/услуги и непрерывного улучшения качества; обеспечение повторяемости процессов, протекающих в организации; осуществление требуемого обучения персонала; оценка эффективности системы; проведение аудита и сертификации системы качества.

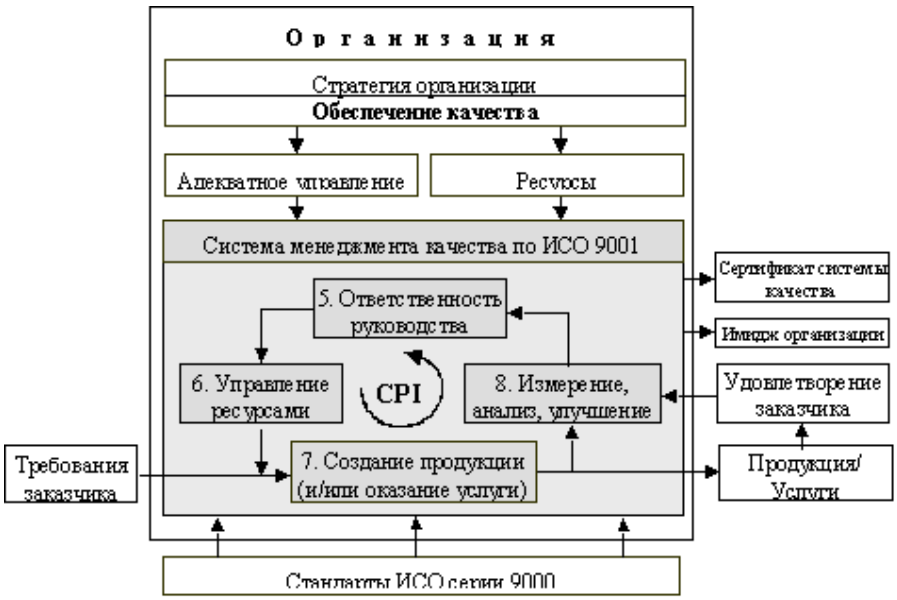
Степень документированности (глубина и подробность описания) определяются самой организацией в зависимости от размера организации и вида деятельности, сложности и взаимодействия процессов, компетентности персонала.

Обязательными для документирования являются:

- процедура по управлению документацией;
- процедура по управлению записями о качестве;
- процедура по проведению внутренних проверок;
- процедура по управлению несоответствующей продукцией;
- процедура по корректирующим действиям;
- процедура по предупреждающим действиям;
    - Состав записей о качества учитывает специфику предприятия, сложившуюся практику, т.е. предприятие само определяет в каком виде вести и хранить эти записи

ISO9000. Как работает система управления качеством

Схема работы системы управления качеством представлена на рисунке [17]. Объектом управления является «производственная линейка»: Требования заказчика – Создание продукции (оказание услуги) - Выпуск продукции (оказание услуги). Обеспечение качества составляет элемент общей стратегии организации, опирается на требуемые для этого ресурсы и адекватное управление, организованными в соответствии со стандартом ISO9001.2000. В соответствии с этим стандартом, адекватное управление включает:



- Выпускаемая продукция контролируется на удовлетворение требований заказчика.

- Этот контроль проводится путем измерений количественных показателей, на основе чего проводится анализ и даются рекомендации по улучшению процессов производства.
- Для выполнения этих рекомендаций подключается руководство, которое достигает результата путем управления соответствующими ресурсами.

Все это вместе способствует повышению имиджа организации и сертификации на соответствие стандарту. Главным в этой схеме является то, что она работает постоянно и непрерывно. Действует принцип CPI: Continuous Process Improvement – Постоянное Улучшение Процессов.

ISO 12207: процессы качества жизненного цикла ПО

В стандарте ISO12207 «Процессы жизненного цикла программного обеспечения» представлены два процесса, относящиеся к управлению качеством ПО: процесс обеспечения качества и процесс усовершенствования. Связано это с тем, что стандарт ISO 12207 разрабатывался вслед за стандартом ISO 9000 и по обеспечению качества содержит ссылки на стандарт ISO 9000.

Процесс обеспечения качества ISO12207 имеет целью обеспечение гарантий того, что программные продукты и процессы в жизненном цикле проекта соответствуют установленным требованиям и утвержденным планам. При обеспечении качества могут использоваться результаты других вспомогательных процессов, таких как верификация, аттестация, совместные анализы, аудит и решение проблем. Процесс состоит из следующих работ: подготовка процесса, обеспечение продукта, обеспечение процесса и обеспечение систем качества, которые должны быть выполнены в соответствии с разделами ГОСТ Р ИСО 9001, указанными в договоре.

Процесс усовершенствования является процессом установления, оценки, измерения, контроля и улучшения любого процесса жизненного цикла программных средств. Процесс состоит из следующих работ: создание процесса усовершенствования, оценка усовершенствуемого процесса и усовершенствование процесса.

При создании процесса усовершенствования определяется набор организационных процессов для всех процессов жизненного цикла в соответствии с имеющимся практическим опытом. Эти организационные процессы и их применение в конкретных ситуациях должны быть задокументированы и определен механизм управления процессом

усовершенствования при разработке, контроле, управлении и улучшении улучшаемых процессов.

Для оценки улучшаемого процесса должна быть разработана, документально оформлена и применена процедура оценки процесса. Должны сохраняться и обновляться отчеты о выполненных оценках процесса. Оценка и анализ улучшаемых процессов должны планироваться и выполняться в установленные сроки.

Для усовершенствования (выполняемых) процессов необходимо:

- Внести (по результатам анализа и оценки) соответствующие улучшения в выполняемый процесс, при этом должны быть внесены соответствующие изменения в документацию выполняемого процесса.
- Для выявления сильных и слабых сторон выполняемых процессов должны быть собраны и проанализированы архивные, технические и оценочные данные. Результаты анализов должны быть использованы для усовершенствования данных процессов, выработки рекомендаций по внесению изменений в реализуемые или планируемые проекты и определения потребности в передовых технологиях.
- Для усовершенствования организационных процессов административной деятельности должны быть собраны, обновлены и использованы данные о расходах. Эти данные должны быть использованы при определении стоимости работ по предотвращению и решению обнаруженных проблем и несоответствий в программных продуктах и услугах.

CMM: зрелость организаций и процессов

CMM SW - Capability Maturity Model for Software – американский стандарт в области качества ПО, разработанный SEI по заказу министерства обороны США. Этот стандарт появился в 1993 году и быстро получил широкое международное признание. Главным образом потому, что, во-первых, этот стандарт предназначен только для разработки ПО и, во-вторых, по отношению к остальным стандартам, управление качеством для разработки ПО в нем прописаны достаточно подробно и детально.

CMM. Причины и история создания

В середине 70-х годов прошлого века министерство обороны США столкнулось с рядом проблем, связанных с разработкой ПО:

- Рост сложности задач, вызванный развитием аппаратной базы. Появление и широкое внедрение интегральных схем существенно повысило производительность вычислительной техники, что позволило переходить к решению качественно более сложных задач. взрывоподобным Рост объема и сложности задач, возлагаемых на программное обеспечение, имел взрывоподобный характер.
- Хронические срывы сроков и качества, как следствие роста сложности задач. Сроки выполнения проектов постоянно срывались, качество ПО (соответствие ожиданиям заказчика) оставалось на неприемлемо низком уровне, и Министерство обороны США начало всерьез беспокоиться об эффективности расходования бюджетных средств.
- Безуспешный поиск методик и инструментов. Усилия были направлены на поиск эффективных методологий и инструментов для разрешения «сугубо технических» (как тогда казалось) проблем программного обеспечения. Почти два десятилетия обещаний поднять производительность и качество работ за счет новых методов и средств разработки ПО ушло на осознание того, что корень зла — не в технике.
- Неспособность организаций управлять процессом разработки ПО как основная причина сложившейся ситуации. В конце концов, был сделан вывод, что фундаментальная проблема «хронического кризиса ПО» состоит в неспособности организаций управлять технологическим процессом разработки программного обеспечения.

И тогда военные приступили к поиску формальных и объективных методов оценки способности организации-разработчика произвести ПО требуемой сложности в установленные сроки и с требуемым уровнем качества. SEI (Software Engineering Institute) получило заказ от министерства обороны США на проведение исследований в этой области. В 1993 году выходит отчет SEI: CMM SW - Capability Maturity Model for Software – Модель технологической зрелости организации-разработчика ПО [18].

CMM. Модель технологической зрелости

Зрелые и незрелые организации

Исследователи SEI пошли достаточно простым путем. Следуя TQM, оценку зрелости организаций они решили проводить на основе анализа выполняемых этой организацией процессов по разработке ПО. При этом считалось (а это – в духе ISO9000), что

организация является тем более зрелой (более предсказуемой), чем более установленными являются применяемые процессы.

Первые исследования организаций с этих позиций показали, что организации находятся на разных уровнях зрелости – была проведена классификация этих уровней, разработаны методы оценки уровня организации и предложены способы повышения уровня зрелости организации. Все это вместе и составляет модель технологической зрелости организации, или модель зрелости ее технологических процессов.

Модель технологической зрелости

В соответствии с CMM модель технологической зрелости - это описание стадий эволюции, которые проходят организации-разработчики по мере того, как они (организации) определяют, реализуют, измеряют, контролируют и совершенствуют процессы создания ПО. Модель помогает выбрать адекватную стратегию усовершенствования процессов, предоставляет методическую основу для определения текущего уровня их совершенства и выявления проблем, критичных для качества разрабатываемого ПО.

Основу модели CMM составляют следующие фундаментальные понятия:

- Process (технология, технологический процесс, процесс) - последовательность шагов (действий), предпринимаемых с заданной целью. Более точно, процесс определяется так: Производственный процесс - набор операций, методов, практик и преобразований, используемых разработчиками для создания и сопровождения ПО и связанных с ним продуктов (например, планов проекта, проектных документов, кодов, сценариев тестирования и руководств пользователя).
- Process Capability (продуктивность, совершенство технологии/процесса) - диапазон результатов, которые можно ожидать от организации, соблюдающей данный технологический процесс. Это понятие имеет отношение к будущим проектам, но базируется на фактических характеристиках технологии, достигнутых на предыдущих проектах.
- Process Performance (производительность технологии/процесса) - фактические результаты, достигнутые организацией, соблюдающей данную технологию/процесс. Это понятие ассоциируется с уже выполненными проектами.

Таким образом, производительность фокусируется на достигаемых результатах, в то время как его продуктивность опирается на ожидаемые результаты.

- Process Maturity (зрелость технологии) - степень определенности, управляемости, наблюдаемости, контролируемости и эффективности процесса, технологии. Фактически, это индикатор полноты технологии и степени последовательности (настойчивости) организации в ее применении на всех проектах. Зрелость определяет потенциал дальнейшего роста совершенства технологии/процесса.

CMM. Пять уровней зрелости

Разработчики модели CMM (SEI) определили пять уровней технологической зрелости, по которым заказчики могут оценивать потенциальных поставщиков (претендентов на получение контракта), а поставщики могут совершенствовать процессы создания ПО Каждому из уровней технологической зрелости внутри модели CMM дано следующее краткое определение:

1. Начальный (Initial). Технология разработки ПО характеризуется как произвольная (импровизированная), в некоторых случаях — даже хаотическая. Лишь некоторые процессы определены, успех всецело зависит от усилий отдельных сотрудников.
2. Повторяемый (Repeatable). Базовые процессы управления проектом ПО установлены для отслеживания стоимости, графика и функциональности выходного продукта. Необходимая дисциплина соблюдения установленных процессов имеет место и обеспечивает возможность повторения успеха предыдущих проектов в той же прикладной области.
3. Определенный (Defined). Управленческие и инженерные процессы задокументированы, стандартизованы и интегрированы в унифицированную для всей организации технологию создания ПО. Каждый проект использует утвержденную, адаптированную к особенностям данного проекта, версию этой технологии.
4. Управляемый (Managed). Детальные метрики (объективные данные) о качестве исполнения процессов и выходной продукции собираются и накапливаются. Управление процессами и выходной продукцией осуществляется по количественным оценкам.

5. Оптимизируемый (Optimized). Совершенствование технологии создания ПО осуществляется непрерывно на основе количественной обратной связи от процессов и плотного внедрения инновационных идей.

CMM. Определение модели зрелости

Модель зрелости должна дать ответ на два вопроса:

- На каком уровне зрелости находится организация:
- Что надо делать, чтобы перейти на следующий уровень?

Модель зрелости CMM является гибкой – она не содержит четких и конкретных (формализованных) указаний на этот счет, а дает некоторую схему поиска ответов на поставленные вопросы и рекомендации по ее использованию. По мнению разработчиков, это расширяет применимость модели.

Структура (схема) модели CMM содержит следующие основные элементы:

- **Группы ключевых процессов**. Каждый уровень зрелости содержит описание группы ключевых процессов, которые должны выполняться на этом уровне.
- **Цели**. Для каждого ключевого процесса определены цели, которые нужно достигнуть для перехода на следующий уровень. Цели (целевые установки):
    - служат критерием эффективной реализации группы ключевых процессов в организации
    - выражают объем, границы и смысл каждой группы ключевых процессов
    - после того, как цели будут реализованы на постоянной основе для всех проектов, можно будет сказать, что организация установила уровень продуктивности своего производственного процесса, характеризующийся данной группой ключевых процессов

  Кроме того, определяется блок связанных работ, после выполнения которых достигаются цели, и круг проблем, которые необходимо решить для достижения следующего уровня зрелости.
- **Разделы**. Описания ключевых процессов организованы по разделам, которые представляют собой атрибуты, указывающие, являются ли эффективными, повторяемыми и устойчивыми реализация и установление групп ключевых процессов. Ниже перечислены пять основных разделов:
    - *Обязательства по выполнению*. Описывают действия, которые должна выполнить организация, чтобы обеспечить установление и стабильность

процесса. Обязательства по выполнению обычно касаются установления организационных политик и поддержки со стороны высшего руководства.

- *Необходимые предпосылки*. Описывают предварительные условия, которые должны выполняться в проекте или организации для компетентного внедрения производственного процесса, обычно касаются ресурсов, организационных структур и требуемого обучения.
- *Выполняемые операции*. В разделе «Выполняемые операции» описаны роли и процедуры, необходимые для внедрения группы ключевых процессов. Выполняемые операции обычно включают в себя создание планов и реализацию процедур, выполнение и отслеживание работ, а также, по мере необходимости, выполнение корректирующих действий.

  Практики раздела «Выполняемые операции» описывают, что должно быть реализовано для получения продуктивного процесса. Все остальные практики вместе формируют базис, с помощью которого организация может внедрить практики, описанные в разделе «Выполняемые операции».
- *Измерения и анализ*. Раздел «Измерения и анализ» описывает, что необходимо для измерения процесса и анализа результатов измерений. В этом разделе обычно приводятся примеры измерений, с помощью которых можно определить статус и эффективность выполняемых операций.
- *Проверка внедрения*. В разделе «Проверка внедрения» описываются шаги, позволяющие убедиться в том, что операции выполняются в соответствии с установленным процессом. В этот раздел обычно входят проверки и аудиты со стороны руководства и работы по обеспечению качества ПО.

- **Ключевые практики**. Описание каждого раздела ключевых процессов выражается ключевыми практиками и подпрактиками, выполнение которых способствует достижению целей группы. Ключевые практики описывают инфраструктуру и операции, которые дают наибольший вклад в эффективное внедрение и установление группы ключевых процессов.

CMM. Группы ключевых процессов

Уровни зрелости включают следующие группы ключевых процессов:

- Начальный
  - Компетентность специалистов, самопожертвование и героизм
- Повторяемый
  - Управление требованиями. Планирование проекта ПО. Отслеживание и контроль проекта ПО. Управление субподрядом. Обеспечение качества ПО. Конфигурационное управление ПО
- Определенный
  - Фокус организации на процессах. Определение процессов в организации. Программа обучения. Интегральное управление ПО. Разработка программной продукции. Координация между группами. Коллегиальное рассмотрение (Peer Review).
- Управляемый
  - Количественное управление процессами. Менеджмент качества ПО.
- Оптимизируемый
  - Предупреждение дефектов. Управление изменениями в технологиях. Управление изменениями в процессах

CMM. Критерии оценки уровня зрелости

В CMM предлагаются следующие критерии оценки соответствия организации тому или иному уровню зрелости:

- Целевые установки группы ключевых процессов считаются удовлетворенными, если действующая в организации практика соответствует всем ключевым элементам практики CMM для данной области или их адекватному эквиваленту.
- Группа ключевых процессов считается удовлетворяющей соответствующему уровню, если все целевые установки CMM в данной области удовлетворены и не удовлетворяющей, если полностью не удовлетворена хотя бы одна ее целевая установка.
- Организация считается соответствующей уровню зрелости, если все ключевые области процессов этого и всех нижестоящих уровней удовлетворены и не считается соответствующей, если хотя бы одна ключевая область процессов этого или любого нижестоящего уровня не удовлетворяет CMM.

CMM. Резюме: CMM в тезисах

Схематично основные принципы CMM можно представить в виде следующих тезисов:

- Зрелость организации есть возможность выполнять сложные проекты
- Зрелость организации определяется через зрелость ее технологических процессов
- Можно выделить уровни зрелости организаций (процессов). В CMM их пять.
- Модель зрелости – описание способа оценки уровня зрелости и путей перехода на следующий уровень
- Модель зрелости описывается:
    - Ключевыми процессами, которые должны выполняться на каждом уровне зрелости
    - Каждый ключевой процесс описывается целями и набором разделов – атрибутов, определяющих различные аспекты выполнения процесса
    - Каждый атрибут описывается в виде ключевых практик – отдельных действий и условий, которые должны выполняться
- Достижение уровня зрелости определяется по критерию:
    - уровень достигнут, если удовлетворены все ключевые процессы этого уровня
    - ключевой процесс удовлетворен, если достигнуты все его цели
    - цели процесса достигнуты, если выполняются все ключевые практики всех разделов или их аналоги

Представленная схема дает способ оценки уровня зрелости организации и определения путей перехода на следующий уровень.

ISO 15504: аттестация, определение зрелости и усовершенствование процессов

ISO 15504. Причины и история создания

Вышедший в 1993 году стандарт CMM SW давал достаточно ясное представление модели зрелости организаций и процессов. Одним из основных свойств этого стандарта являлась недостаточная четкость критериев оценки. При применении стандарта возникали вопросы:

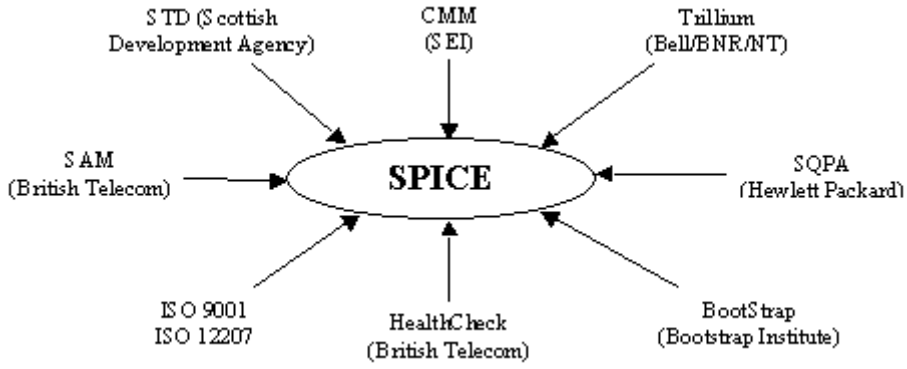
- Каковы могут быть аналоги ключевых практик?
- Как быть, если выполняется только часть ключевых практик?

- Можно ли считать, что ключевая практика частично и как это оценить?

Это качество CMM существенно усложняло сертификацию на соответствие CMM. Причины этого «недостатка» состояла в том, что формально CMM не стандарт, а отчет, имеющий рекомендательный характер. CMM – это модель, являющаяся основой для разработки стандартов.

Надо сказать, что проблемой оценки и выбора организаций, способных выполнять сложные ИТ проекты было озабочено не только министерство обороны США. Процессный подход к организации любого вида работ (в духе TQM и ISO 9000) к тому времени уже сложился окончательно, и в 1991 г. ISO инициировала работу по созданию единого стандарта оценки программных процессов. Первоначально этот стандарт получил рабочее название SPICE - **S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination - определение возможностей и улучшение процесса создания программного обеспечения.

Основная цель SPICE состояла в создании международного стандарта, в котором был бы учтен весь накопленный опыт в области разработки ПО. Стандарт SPICE унаследовал многие черты более ранних стандартов (см. рисунок). Для этого пришлось прибегнуть к повышению уровня детализации стандарта. Следствием такого основательного подхода является большой объем стандарта: документация к нему содержит около 500 страниц.

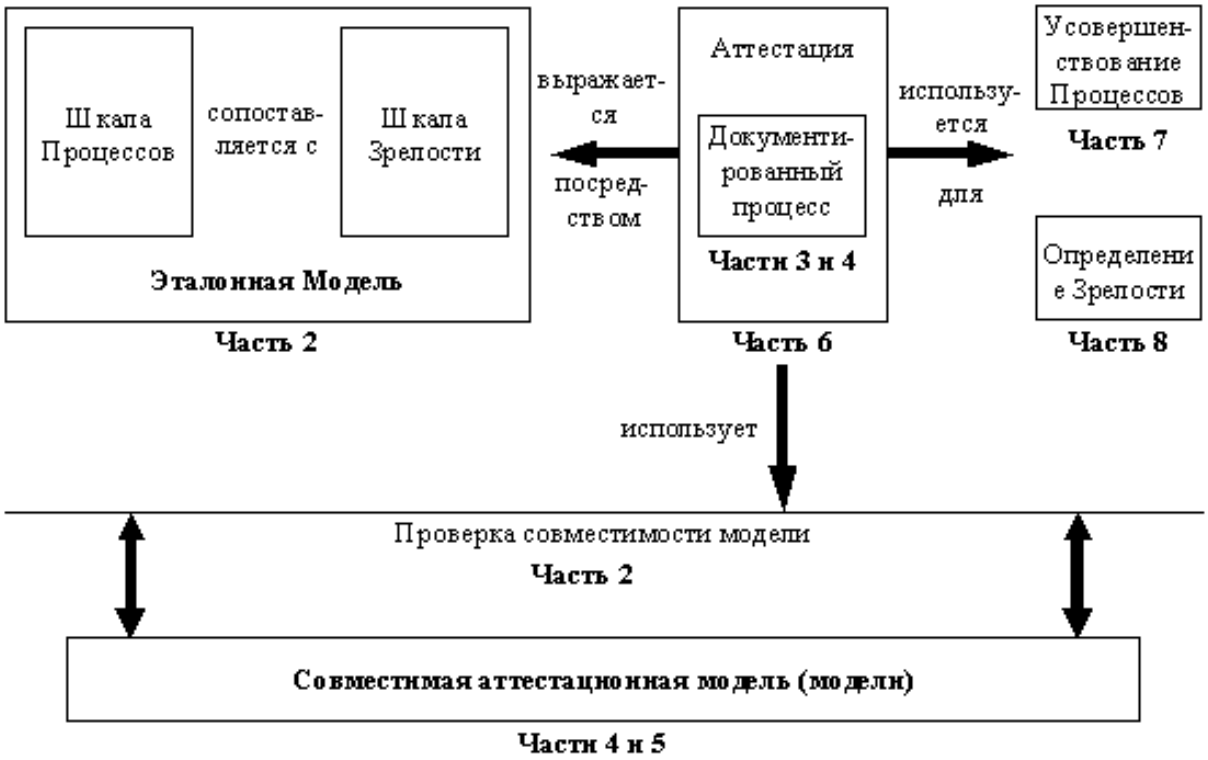


В 1998г. вышла официальная версия стандарта под названием ISO/IEC TR 15504 CMM: Information Technology - Software Process Assessment", которая на данный момент существует в качестве рабочей версии (технического отчета). Издан перевод отчета на русский язык [5].

ISO 15504. Назначение и структура стандарта

ISO/IEC TR 15504 предоставляет основу для аттестации процессов жизненного цикла программных средств, определения зрелости процессов и усовершенствования процессов.

Аттестация исследует процессы, используемые организацией, чтобы определить,



насколько эффективно они достигают своих целей. Аттестация характеризует текущую деятельность организационной единицы в терминах зрелости выбранных процессов.

Результаты аттестации могут быть использованы для усовершенствования процессов или определения зрелости процессов путем анализа результатов аттестации в контексте бизнес-потребностей организации и выявления сильных и слабых сторон процессов, а также сопряженных с ними рисков.

На схеме показаны общий вид зависимостей между аттестацией процессов, усовершенствованием процессов и определением зрелости процессов, а также показано место различных компонентов ISO/IEC TR 15504 в процессах:

- Аттестация: является документированным процессом (описан в ч. 3,4), выраженным в терминах эталонной модели (ч. 2) и может использоваться как в целях Усовершенствования процессов (ч. 7), так и в целях Определения Зрелости (ч. 8).
- Проведение аттестации: требует модели (или моделей), совместимых с эталонной моделью (ч. 4, 5), проверка совместимости которой описана в ч. 2.
- Ведущий аттестатор: отвечает за соответствие аттестации этим правилам; необходимые для этого навыки и компетентность описаны в ч. 6.

ISO 15504.Структура эталонной модели

Эталонная модель состоит из двух измерений: измерения «Процесс», содержащего перечень аттестуемых процессов ЖЦ ПО и измерения «Зрелость», содержащего приложимый к любому процессу набор атрибутов, представляющих собой измеримые характеристики, необходимые для управления процессом и повышения зрелости его выполнения.

Измерение «Процесс»

Измерение «Процесс» представляет расширенный и уточненный вариант процессов, представленных в стандарте ISO 12207 (см. раздел «Процессы жизненного цикла стандарта ISO/IEC 15504» настоящего пособия).

Измерение «Зрелость»

В измерении «Зрелость» эталонной модели мера зрелости основывается на наборе атрибутов процессов (process attribute - PA). Атрибуты процессов используются для определения того, достиг ли процесс определенной способности. Каждый атрибут является мерой конкретного аспекта зрелости процесса. Атрибуты, в свою очередь, оцениваются в процентах, что дает дополнительное понимание конкретных аспектов зрелости процессов, необходимое для усовершенствования процессов и определения их зрелости.

В эталонной модели ISO15504 в отличие от CMM применяется шесть уровней зрелости процессов со следующими атрибутами:

1. Неполный процесс (Incomplete). Процесс не реализован, или не способен достичь итога процесса. На данном уровне доказательства систематического обладания любым из предписанных атрибутов отсутствуют либо недостаточны.
2. Выполняемый (Performed) - реализованный процесс достигает итог процесса.
    - *PA 1.1 Выполнение процесса* - *степень*, в которой процесс достигает соответствующего результата процесса, преобразуя идентифицируемые входные рабочие продукты в идентифицируемые выходные рабочие продукты. В случае обладания данным атрибутом в полной мере:
        - будут понятны объем выполняемых работ и рабочие продукты, которые надо произвести;
        - будут получены рабочие продукты, поддерживающие достижение итога процесса.

3. Управляемый (Managed) - ранее описанный осуществляемый процесс выполняется теперь под управлением, основанном на определенных целевых показателях (т.е., планируется, отслеживается, верифицируется и настраивается).
    - *PA 2.1 Управление выполнением*
    - *PA 2.2 Управление рабочими продуктами*
4. Устоявшийся (Established) - ранее описанный управляемый процесс теперь выполняется на основе заданного процесса, основанного на правильных с точки зрения программной инженерии принципах и способного достичь своего назначения.
    - *PA 3.1 Задание процесса*
    - *PA 3.2 Обеспечение процесса ресурсами*
5. Предсказуемый (Predictable) - ранее описанный устоявшийся процесс теперь устойчиво выполняется в заданных пределах для достижения назначения процесса.
    - *PA 4.1 Измерение*
    - *PA 4.2 Количественное управление процессом*
6. Оптимизируемый (Optimizing) - ранее описанный предсказуемый процесс теперь динамически адаптируется и изменяется для того, чтобы эффективно отвечать соответствующим текущим и проектируемым бизнес–целям организации.
    - *PA 5.1 Изменение процесса*
    - *PA 5.2 Непрерывное усовершенствование*

Рейтинги атрибутов

Атрибут процесса представляет собой измеримую характеристику любого процесса, как и определено выше. Шкала рейтингов представляет собой процентную шкалу от единицы до ста процентов, представляющую степень обладания атрибутом. При этом, устанавливается качественная калибровка шкалы рейтингов:

Код	***Название***	***%***	***Комментарий***
NA	Not Achieved - Не обладает	0% - 15%	Доказательства того, что аттестуемый процесс обладает заданным атрибутом, отсутствуют либо недостаточны
A	Achieved - Обладает частично	16% - 50%	Существуют доказательства разумного систематического подхода к заданному атрибуту и того, что аттестуемый процесс обладает им в некоторой степени. Некоторые аспекты достижения могут быть

			непредсказуемыми.
L	Largely achieved - Обладает в основном	51% - 85%	Существуют доказательства разумного систематического подхода к заданному атрибуту и того, что аттестуемый процесс обладает им в значительной степени. Выполнение процесса может варьироваться в некоторых областях или организационных единицах.
F	Fully achieved - Обладает полностью	86% - 100%	Существуют доказательства полного и систематического подхода к заданному атрибуту и того, что аттестуемый процесс обладает им в полной мере. В заданной организационной единице отсутствуют заметные недостатки

ISO 15504. Процесс аттестации

В ISO/IEC155404 процесс аттестации представлен как документированный процесс в виде набора инструкций и процедуры для проведения аттестации. Основные требования к процессу аттестации состоят в том, что в зависимости от подхода, документированный процесс должен обеспечивать указания по следующим темам:

- Роли и обязанности
- Применение инструментальных средств и методик
- Требуемые ресурсы
- Последовательности видов деятельности и процедур, принадлежащих следующим категориям: планирование; сбор данных; подтверждение данных; формирование рейтингов процесса.

Процесс аттестации начинается с выбора совместимой модели, требования к которой сформулированы во второй главе стандарта. Вкратце, совместимая модель, это модель:

- которая подходит для конкретного назначения аттестации процессов;
- чьи фундаментальные элементы могут быть сопоставлены и сопоставляются с измерениями «процесс» и «зрелость» эталонной модели;
- которая содержит набор показателей для применения во время аттестации для сбора информации о процессах и их атрибутах;
- которая имеет формальный механизм преобразования информации, собранной с применением модели, в рейтинги атрибутов процессов, как описано в стандарте.

Вспомогательные инструменты и инструментальные средства. Документированный процесс аттестации должен поддерживаться различными инструментами и инструментальными средствами для сбора информации, ее обработки и представления. Для некоторых аттестаций вспомогательные инструментальные средства и инструменты

могут быть ручными и основанными на бумажных документах (формы, анкеты, проверочные листы и т. д.). В некоторых случаях объем и сложность аттестационной информации приводят к потребности во вспомогательных инструментальных средствах, ориентированных на применение компьютеров.

Факторы успеха аттестации процессов. Следующие факторы существенны для успешной аттестации процессов:

- Обязательства заказчика и аттестатора. Заказчик аттестации должен взять на себя обязательства по достижению установленных целей аттестации, чтобы обеспечить полномочия для проведения аттестации в организации. Эти обязательства требуют, чтобы для проведения аттестации были предоставлены необходимые ресурсы, время и персонал. Обязательства заказчика аттестации и аттестаторов чрезвычайно важны для достижения целей аттестации.
- Мотивация – поддержка процессов, а не поиск виноватых. Позиция руководства организации и документированный процесс аттестации, посредством которого собирается информация, оказывают существенное влияние на результат аттестации. Таким образом, руководство организации должно мотивировать участников быть открытыми и конструктивными. Аттестация процессов концентрируется на процессе, а не на производительности членов организационной единицы, выполняющих процесс. Ее цель в том, чтобы сделать процессы более эффективными, чтобы поддерживать определенные бизнес-цели, а не в том, чтобы возложить вину на сотрудников.

  Обеспечение обратной связи и поддержание в ходе аттестации атмосферы, способствующей открытому обсуждению предварительных выводов, помогает обеспечить значимость выходных данных аттестации для организационной единицы. Организация должна осознавать, что участники являются главным источником знаний и опыта относительно процесса и что они имеют хорошую возможность выявить потенциальные слабости.
- Конфиденциальность. Внимание к конфиденциальности источников информации и документации, собранных в ходе аттестации, является существенным для обеспечения информационной безопасности. Если используются методики обсуждений, особое внимание должно быть уделено тому, чтобы участники не ощущали, что им что-либо угрожает и не имели бы причин беспокоиться относительно конфиденциальности. Часть предоставленной информации может быть собственностью организации. Таким

образом, важно предусмотреть адекватные рычаги управления такой информацией.

- Релевантность – уверенность в выгоде аттестации. Сотрудники организационной единицы должны быть уверены, что аттестация приведет к определенным выгодам, которые коснутся их прямо или косвенно.
- Доверие. Заказчик аттестации, а также руководство и персонал организационной единицы должны быть уверены, что аттестация принесет результат, являющийся объективным и характеризующим объем аттестации. Важно, чтобы все стороны были уверены в том, что аттестаторы имеют должный опыт аттестации, достаточно беспристрастны и имеют должное понимание организационной единицы и ее бизнеса для того, чтобы проводить аттестацию.

Компетентность аттестаторов

Вопросам компетентности посвящена отдельная глава стандарта. Требования к компетентности аттестаторов формулируются в следующем виде:

- Для проведения аттестации аттестаторы демонстрируют свою компетентность
- Компетентность является результатом:
  - знания процесса, относящегося к программным средства;
  - владения основными технологиями ISO 15504, включая эталонную модель, аттестационные модели, методы и инструментальные средства, а также процессы выставления рейтингов;
  - личных качеств, способствующих эффективной работе;
- Знания, навыки и личные качества приобретаются в результате образования, специальной подготовки и опыта
- Альтернативой демонстрации компетентности является подтверждение образования, специальной подготовки и опыта.

В стандарте также оговаривается, что аттестаторы должны обладать следующими личными качествами: эффективное письменное и устное общение; дипломатичность; ответственность; настойчивость и умение преодолевать сопротивление; рассудительность и лидерство; прямота; способность к взаимопониманию.

ISO15504. Резюме: ISO15504 в тезисах

Схематично основные принципы ISO/IEC15504 можно представить в виде следующих тезисов:

- Назначение стандарта состоит в аттестации, усовершенствовании и определении зрелости процессов создания ПО.
- Основу стандарта составляет эталонная модель процессов и их зрелости. Эталонная модель имеет два измерения: «Процессы» и «Зрелость».
- Измерение «Процессы» содержит классификацию процессов ЖЦ ПО. Эта классификация является развитием стандарта ISO12207 и включает:
  - три группы и пять категорий процессов;
  - разделение процессов (по отношению в ISO12207) на базовые, расширенные, новые, составляющие и расширенные составляющие.
- В отличие от CMM, в измерении «Зрелость» представлено 6 уровней зрелости процессов, по каждому из которых установлены атрибуты, отражающие достижение процессом уровня зрелости. Значения атрибутов оцениваются в процентах от полного достижения атрибута. Для качественной оценки вводятся рейтинги атрибутов.
- Аттестация процессов составляет основу для их оценки и усовершенствования. Аттестация процессов:
  - состоит в определении значений рейтингов атрибутов процессов;
  - начинается в выбора модели процессов и их зрелости аттестуемой организации, совместимой с эталонной моделью стандарта;
  - является документированным процессом, представленным в стандарте в виде инструкций и регламентированной процедуры аттестации;
  - проводятся аттестаторами, требования к компетентности которых также прописаны в стандарте;
- Оценка зрелости и усовершенствование процессов выполняются по результатам аттестации процессов и также являются документированными процессами стандарта.

Литература

1. The Standish Group International The Standish Group International, Inc., Extreme Chaos, 2000 - http://www1.standishgroup.com//sample_research/PDFpages/extreme_chaos.pdf
2. Шафер Д, Фатрел Р, Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат.: Пер. с англ. - М.: Вильямс., 2003. - 1136с.

3. Соммервилл Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ. - М.: Издательский дом "Вильямс", 2002. - 624 с
4. ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств. http://www.staratel.com/iso/InfTech/DesignPO/ISO12207/ISO12207-99/ISO12207.htm.
5. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 CMM) / Пер.с англ. А.С. Агапов, С.В. Зенин, Н.Э. Михайловский, А.А. Мкртумян А.А. - М.: Книга и бизнес, 2001. - 348с. ISBN: 5-212-00884-0.
6. Royce W.W. Managing the Development of Large Software Systems. http://facweb.cti.depaul.edu/jhuang/is553/Royce.pdf.
7. Barry Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol.21, No.5, pp. 61-72, 1988. www.computer.org/computer/homepage/misc/Boehm/r5061.pdf.
8. Руководство к своду знаний по управлению проектами (Руководство PMBOK). Издательства: Институт Управления Проектами, Project Management Institute, 2004 г.
9. Формальные методы управления проектами в России используют только 5% компаний. http://www.mdi.ru/aspnews/body/01.08.2002_61638.html.
10. Салливан Эд. Время – деньги. Создание команды разработчиков программного обеспечения/ Пер.с англ. – М.: Русская редакция, 2002. – 364с.
11. Управление проектами: технология MSF. http://www.microsoft.com/rus/msdn/msf/default.mspx
12. Алистэр Коуберн. Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения. http://www.optim.ru/cs/2002/3/cobern/people.asp.
13. Константин Л. Человеческий фактор в программировании. - Пер. с англ. - СПб: Символ-Плюс, 2004. - 384 с.
14. Weinberg, J., The Psychology of Computer Programming, Silver Edition, Dorset House, 1998.
15. DeMarco, T., Lister, T., Peopleware 2nd Edition, Dorset House, 1999. (Т. ДеМарко, Т.Листер. Человеческий фактор: эффективнын проекты и команды. - Пер. с англ. - СПб: Символ-Плюс, I кв. 2004 г.).
16. Филип Лапланте. Человеческий фактор в управлении ИТ-проектом. http://www.info-system.ru/pj_managment/article/pj_people_factor.html
17. Международные стандарты ISO серии 9000. http://www.m2bc.ru/qs_iso-scheme

18. Марк Паулк и др. Модель зрелости процессов разработки программного обеспечения - Capability Maturity Model for Software (CMM) Интерфейс-Пресс. 2002 г. · 256с.