Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Учебно-методическое пособие по дисциплине «Управление качеством разработки ПО»

Генельт А.Е.,
ассистент кафедры математического моделирования

Санкт-Петербург
2007

Оглавление

Тема 1. Понятие качества продукта и процесса

> *"... Качество необходимо России: верные, волевые, знающие и даровитые люди; крепкая и гибкая организация; напряженный и добросовестный труд; выработанный первосортный продукт; высокий уровень жизни. Новая, качественная эпоха нужна нашей Родине, эпоха, которая довершила бы все упущенное за время перегруженности и беспечности, которая исцелила бы, зарастила бы все язвы революционного времени. <...> И готовить восстановление России - значит, прежде всего, готовить себя самого к качественному служению Родине; готовить свой характер, свой разум, свое чувство, свою волевую идею. Имя этой волевой идеи - русское качество."*
>
> *ИВАН АЛЕКСАНДРОВИЧ ИЛЬИН* [1]
> *(1882–1954), русский философ и правовед*

Качество продукции – понятие очень широко используемое, но весьма трудно формализуемое. Дело в том, что разные потребители одного и того же изделия рассматривают его качество с разных точек зрения. Для некоторых изделие является качественным, если оно надежно, другие обращают внимание, прежде всего на удобство использования, третьи на приятный дизайн. Таким образом, понятие качества оказывается весьма субъективным.

В незавершённой работе [2] “Диалектика природы” в 1878 году Фридрих Энгельс писал, что: “… у двух различных вещей всегда имеются известные общие качества (но крайней мере свойство телесности), другие качества отличаются между собой по степени, наконец, иные качества могут совершенно отсутствовать у одной из вещей…”.

В отличие от блестящих философских изысканий мыслителей прошлого сегодняшние представления о качестве тесно связаны с понятиями качества продукции и систем управления (менеджмента) качеством. Отношение общества к качеству продукции привело к формированию концепции всеобщего качества (TQM - Total Quality Management"). В работе “ИСТОРИЯ МЕТРОЛОГИИ, СТАНДАРТИЗАЦИИ, СЕРТИФИКАЦИИ И УПРАВЛЕНИЯ КАЧЕСТВОМ" [3] показано, что возникновению интереса общества к менеджменту качества предшествовали развитие метрологии:

- Этап стандартизации и сертификации в Древнем мире;
- Этап стандартизации и сертификации в X-XVII вв. на Руси;
- Этап стандартизация и метрология в период правления Петра I в XVIII в.;

- метрологическая реформа Петра I;
- старинные русские меры длины, веса и объема;
- история развития и внедрения метрической системы во Франции;
- Этапы развития отечественной метрологии в XIX-XX вв.;
- Этап подписание метрической конвенции 20 мая 1875 г.;
- Менделеевский этап развития отечественной метрологии;
- Нормативный этап развития отечественной метрологии;
- Метрология в Российской Федерации;

и развитие стандартизации:

- стихийной сертификации;
- организационной национальной сертификации и стандартизации систем качества;
- Государственные испытания в бывшем СССР – прообраз сертификации;
- Этап международной сертификации и управления качеством.

В разделе “Краткий обзор этапов развития всеобщего управления качеством” авторы работы выделили четыре этапа:

- Этап контроля качества;
- Этап технического управления качеством;
- Этап обеспечения качеством;
- Этап всеобщего управления качеством (TQM – Total Quality Management).

Рассмотрим примеры применения систем менеджмента качества на этих этапах.

1.1. Этап контроля качества.

Для этапа контроля качества характерно создание бригад контролеров для испытания продукции, сравнения ее характеристик с установленными требованиями и выявления брака. Хорошая продукция поступала на склад и далее к потребителю. Плохая продукция или признавалась окончательным браком и уничтожалась, или признавалась не окончательным браком и ее ремонтировали, снижали класс качества, а затем реализовывали по более низкой цене. Продукция с признанным окончательным браком частично использовалась.

Недостатки этапа контроля качества:

- дефектная продукция попадала к потребителю;
- ответственность за качество возлагалась на контролеров, несмотря на то, что на самом деле качество создавали рабочие основного производства.

Пример из этапа контроля качества:

“Петр I.Указ Царя
от II января 1723 года [4]

Повелеваю хозяина Тульской фабрики Корнилу Белоглазова бить кнутом и сослать на работу в монастыри, понеже он, подлец, осмелился войску Государеву продавать негодные пищали и фузеи, старшину Альдермала [[1]] *Флора Фукса бить кнутом и сослать в Азов, пусть не ставит клейма на плохие ружья. Приказано оружейной канцелярии из Петербурга переехать в Тулу и денно и нощно блюсти исправность ружей. Пусть дьяки и подьячие смотрят, как альдермалы клейма ставят, буде сомнение возьмет, самим проверить и осмотром и стрельбою. А два ружья каждый месяц стрелять, пока не испортится. Буде заминка в войске приключаться при сражении, по недогляду дьяков и подьячих, бить оных кнутьями и нещадно по оголенному месту. Хозяину 25 кнутов и пени по червонцу за ружье, Старшине Альдермалу - бить до бесчувствия. Старшего дьяка отдать в унтерофицеры. Дьяка - в писари. Подьячего лишить чарки сроком на один год. Новому хозяину ружейной фабрики Демидову повелеваю построить дьякам и подьячим избы не хуже хозяйской были, буде хуже, пусть Демидов не обижается, повелеваю живота лишить.”*

[1] Альдемарл - фамилия старшины, которому также тогда "досталось на орехи" от царя. "Северная Таврида" № 48 (869), 7 декабря 2006 г.

Рисунок 1. Пётр Великий

Несмотря на устоявшуюся практику того времени - применять клеймление изделий с целью сбора таможенной пошлины [5], в лаконичном указе Петра I прежде всего говорится о контроле качества оружия. За свою многовековую историю клеймление, являясь товарным знаком, стало мощнейшим маркетинговым инструментом, определяющим жизнеспособность товара на рынке. В разное время этот вид товарного знака исполнял различные функции, однако всегда являлся важным движущим мотивом покупки и своеобразной гарантией качества товара. Что же касается рассмотренного выше стиля менеджмента качества тех лет, то нам следует лишь посочувствовать старшине Альдермалу. Ему было совсем не смешно.

1.2. Этап технического управления качеством.

При этапе технического управления качеством внимание уделяется сбору информации, применению технических систем с обратной связью и к промежуточным элементам контроля качества выпускаемой продукции. Окончательный контроль рассматривался как основная защита интересов потребителя. Контролеры выявляли бракованную продукцию, и информация об этом безотлагательно передавалась в производственные подразделения.

Как и на этапе контроля качества, хорошая продукция поступала на склад и далее к потребителю. Плохая продукция также признавалась окончательным браком, ремонтировалась и затем реализовывалась по более низкой цене.

Известный детальностью своих произведений, писатель Артур Хейли описывает действительное положение дел с качеством продукции автомобильного концерна Дженерал Моторс (General Motors) в середине 70-х годов прошлого века: "...А понедельники и пятницы на автомобильных заводах из-за прогулов – самые тяжелые для начальства дни. Каждый понедельник куда больше рабочих, чем в любой другой день, не является на работу; а за понедельником по числу прогульщиков следует пятница. Дело в том, что по четвергам обычно выдают жалованье, и многие рабочие предаются трехдневному запою или принимают наркотики, а в понедельник отсыпаются или приходят в себя. Таким образом, по понедельникам и пятницам все проблемы отступают на второй план, кроме одной, самой главной: как обеспечить

выпуск продукции, несмотря на критическую нехватку рабочих. Людей переставляют, точно пешки на шахматной доске. Некоторых перебрасывают с той работы, к которой они привыкли, на ту, которой они раньше никогда не выполняли. Рабочим, обычно завинчивающим гайки на колесах, могут поручить установку передних крыльев, дав перед этим лишь минимальные инструкции, а то и никаких. Даже рабочих без особой квалификации, например, занятых погрузкой машин или уборкой помещений или взятых прямо из конторы по найму, могут направить туда, где образуется прорыв. И они иногда быстро осваиваются со своими временными обязанностями, а иногда целую смену пытаются приладить вверх тормашками шланг обогревателя или какую-нибудь другую деталь. Это, естественно, не может не сказаться на качестве. Поэтому большинство машин, выпущенных в понедельник и пятницу, собраны кое-как, с “запланированными” дефектами, и те, кто в курсе дела, избегают их, как гнилого мяса. Некоторые наиболее крупные оптовики, которым известно это обстоятельство и с которыми фирмы считаются ввиду объема их закупок, обычно требуют для наиболее уважаемых клиентов машины, собранные во вторник, среду или четверг, и сведущие покупатели обращаются именно к таким оптовикам, чтобы получить приличную машину. Сборка автомобилей для служащих компании и их друзей производится только в те же дни..." [6]

Пример из этапа технического управления качеством:

В знаменитой работе “The Principles of Scientific Management” [7], почти четверть объема которой состоят из подробных примеров практик менеджмента качества, Фредерик Уинслоу Тейлор приводит пример системного исследования труда чернорабочих на Вифлеемской Стальной Компании по уборке территории и разгрузки вагонов и платформ и складировании сыпучих материалов.

Рисунок 2. Фредерик Уинслоу Тейлор, 1910 г.

К моменту начала исследования в перечисленных процессах было занято примерно 600 чернорабочих, а объём выгребаемого вручную материала (железная руда, уголь), размещённого на различных покрытиях (деревянный или железный настил, в сухой или в сырой среде) составлял до 9600 тонн/день. Кроме того, чернорабочие были заняты перегрузкой сырья и материалов (руды, кокса, известняка, песка, угля и т. д.) и их транспортировкой к

доменным и мартеновским печам, транспортировкой и погрузкой готовой продукции (чугун в чушках) от печей и заготовок от прокатных станов на железнодорожные платформы, предназначенные для вывоза продукции за пределы завода. Важно также отметить, что к конкретному процессу производства в единицу учётного времени мог относиться любой из рабочих, который получал $1,15/день.

Тейлор поступил на работу в Вифлеемскую Стальную Компанию в 1893 г.. Убеждённый в том, что знание технических деталей процессов производства и наличие метрик, а также применение методов материального поощрения персонала и жёсткого контроля нормативов выполнения работ может существенно повысить степень эффективности производства, Тейлор взялся это доказать. Он использовал свою систему управления заданиями, разбив выявленные виды работ на отдельные элементы и прохронометрировав выполнение каждого из них. Тейлор организовал исследование вопросов о влиянии вариантов нагрузки на лопату в 5 (англ.) фунтов[2], 10 фунтов, 15 фунтов, 20, 25, 30 или 40 фунтов на объём выполняемой работы, а результате чего в качестве эталонной расчётной величины был принят размер нагрузки в 21 фунт. Для каждого рабочего было изготовлено от 8 до 10 различных типов лопат, а также и тщательно разработанные и фиксированные в своих нормальных размерах (стандартизованные) рабочие орудия (ломы, ворота и пр.). Это дало возможность предоставить каждому рабочему лопату, которая вмещала бы нагрузку, весом в 21 фунт, для любого рода материала, с которым ему пришлось бы иметь дело, например маленькую лопату для железной руды, и большую — для золы. Им были проделаны тысячи измерений с секундомером в руках для изучения вопроса о том, с какой скоростью рабочий, снабженный в каждом данном случае лопатой надлежащего типа, может погружать свою лопату в кучу материала, а затем поднимать ее с надлежащей нагрузкой. Эти наблюдения были сделаны, при погружении лопаты в середину кучи, при работе лопатой по грязному дну на внешнем краю кучи, а затем отдельно при работе лопатой по деревянному дну и по железному дну. Исследовались продолжительности движения заноса лопаты назад, а затем выбрасывания груза вперед на определенное горизонтальное расстояние при определенной высоте броска. Это измерение времени было проделано для различных возможных комбинаций расстояния и высоты.

Тейлор предложил увеличить на 60% (до $1,85) дневной заработок специально отобранных рабочих. Приходя утром на работу, рабочий получал два листка бумаги, на одном из которых было указано - какие инструменты он должен был получить со склада и в каком месте он

[2] Стандартный американский и английский фунт, равен 16 унциям или 453,592374495300 грамма

должен был приступить к работе, на другом - описание истории его работы накануне (подсчет произведенной им работы, заработанной платы и др.).

За два года применения системы менеджмента качества на Вифлеемской Стальной Компании были получены следующие результаты:

	При старой системе	При новой системе урочной работы
Число (занятых) рабочих	от 400 до 600 чел	приблиз.140 чел.
Среднее число тонн выработки на человека в день составляло	16	59
Средний заработок одного рабочего в день составлял	$1,15	$1,88
Средняя себестоимость переработки одной тонны, равной 2240 фунтов	$0,072	$0,033

В расчете себестоимости (0,033 доллара на тонну) учтены издержки на постройку конторы и склада инструментов, на вознаграждение всех агентов по организации труда, надсмотрщиков, клерков, счетчиков рабочего времени и. т.д..

В течение последнего года применения системы менеджмента качества (СМК) на Вифлеемской Стальной Компании, общая сумма экономии, составила 38417,69 долларов, а в течение следующих шести месяцев, когда вся работа на заводском дворе была целиком переведена на урочную систему, сбережение выразилось в размере от 75000 до 80000[3] долларов в год.

На классическом примере технического управления качеством видно, что организация производства строится на принципах системного управления с учётом особенностей изученных бизнес-процессов. В популярной экономической работе, имеющей, к сожалению, оттенки социалистической риторики 60-х годов [8], известный русско-американский экономист В.И. Терещенко пишет: "...Годы 1896 и 1911, когда вышли две основные работы Тейлора "Сдельная система" и "Принципы управления", считаются годами зарождения новой дисциплины. Тейлор высказал мысль о том, что если в прошлом в вопросах организации и управления на первом месте стоял организатор, то в настоящее время на первое место выступает "система". Иначе говоря, роль человека-организатора должна сводиться, по мысли Тейлора, лишь к установлению организации самой системы...".

[3] ~$2 857 593,75 сегодня.

1.3. Этап обеспечения качеством.

На этапе обеспечения качеством центр внимания с выявления дефектов был перенесен на их предупреждение. В редакции 1994 г. в модели ИСО 9001:94 введён элемент системы качества "Корректирующие и предупреждающие действия". Для этапа обеспечения качеством характерно применение первых руководств по качеству, программ качества, а также технологических и рабочих инструкций. Этап соответствует уровню предприятия, имеющего сертифицированную систему качества, которая направлена на предупреждение дефектов, а акцент качества продукции перенесен на качество процессов и систем. На этап обеспечения качеством существенно влияют результаты, полученные от внедрения разработок концепции TQM. Эдвард Деминг призывает японцев применять к решению проблем системный подход, известный как "цикл Деминга" PDCA (Plan, Do, Check, Action) – "планирование, выполнение, проверка, действие" [9].

Рисунок 3. Цикл Деминга

Опыт развития индустрии в Японии приводит к исключению промышленного брака и к непрерывному циклу улучшения продукции. Основной целью внедрения систем менеджмента качества (СМК) становится создание таких условий в организации, когда происходит постоянное улучшение каждого из ее процессов.

Пример этапа обеспечения качеством:

В качестве примера этапа обеспечения качеством возьмем проект Linux. Операционная система Linux поставляется потребителю целиком, включая исходные тексты программ модулей системы и инструментальные средства по её модификации. Пользователь Linux получает возможность “собрать” собственное программное обеспечение применительно к решаемым задачам, используя при этом конкретные аппаратные особенности его оборудования. В отличие от операционных систем, которые не поставляют исходных текстов потребителю, Linux представляет собой программное обеспечение серверного типа с открытым, не спрятанным от конкурентов исходным кодом. Эта особенность неожиданно для разработчиков принесла значительные качественные преимущества. Усилиями пользователей участвующих в разработках для Linux в режиме обратной связи и высокий профессионализм организаторов сотрудничества позволили получить постоянный эффект добавленной ценности. Разработчики Linux практически избавилась от производственного брака. Огромное количество профессионалов со всего мира непрерывно проявляет ежедневную заинтересованность в качестве ядра операционной системы и принимают практическое участие в его усовершенствовании.

Рисунок 4. Линус Торвальдс, автор Linux

В интервью каналу CNN Линус Торвальдс[4] [10] сказал, что фактически он работает с небольшим количеством сотрудников (10 – 20 человек), а они, в свою очередь, взаимодействуют с другими людьми, их может быть 5000 и более. Основным каналом является e-mail, и неважно, где находится человек. Даже если все остальные операционные системы поддержат принципы открытого программного обеспечения - OSS (Open Source Software - программное обеспечение с открытым исходным кодом), им вряд ли удастся организовать столь масштабный процесс по непрерывному улучшению качества программного обеспечения. Поддерживать тенденцию усиления производительности при разработке операционной системы позволяет следующий процесс. Многочисленные конкурирующие дистрибьюторы Linux (Debian, Gentoo, Mandriva, Red Hat, Slackware, Fedora, SuSE, Ubun и другие), предлагают рынку потребителей различные “модели” Linux независимо от разработчиков. Разработчики Linux пишут код операционной системы и обеспечивают к нему доступ в сети Интернет. Любой дистрибьютор выбирает некоторый поднабор

[4] Создатель Linux

доступного кода, объединяет в пакеты и продает это клиентам под своей маркой. Пользователи выбирают нужный им дистрибутив и получают возможность дополнить дистрибутив, загружая код непосредственно с сайтов разработчиков. В работе [11], опубликованной под общей редакцией Тито Конти, Ёсло Кондо и Грегори Ватсона, приведено описание исследования под названием "Обеспечение качества программного продукта на примере создания системы Linux", которое провели Роберт И. Коул, Гвендолин К. Ли несколько лет назад.

Рисунок 6. Д-р Тито Конти. Экс-президент Европейской организации по качеству (EOQ), член Международной академии качества и Американского общества по качеству (ASQ)

Рисунок 5. Д-р Грегори Ватсон. Секретарь-казначей Международной академии качества.

Член Американского общества качества (ASQ), член Британского института обеспечения качества, Австралийской организации качества, Австралийско-азиатского общества качества и Всемирного совета по научным основам производительности. Автор семи книг по качеству. Награжден ASQ медалью Ланкастера за глобальный вклад в развитие системы знаний о качестве. В 2001 году в Японском союзе ученых и инженеров читал 50-ю, юбилейную, ежегодную лекцию в память Э. Деминга. Диплом «Мастера методологии «Шесть сигм»

1.4. Этап всеобщего управления качеством

Поскольку мы добрались до ключевой темы в данной главе, то сначала необходимо определить, что представляет собой этап всеобщего управления качеством.

“Отцом” концепции всеобщего качества (TQM - Total Quality Management) является Уильям Эдвард Деминг (William Edwards Deming).

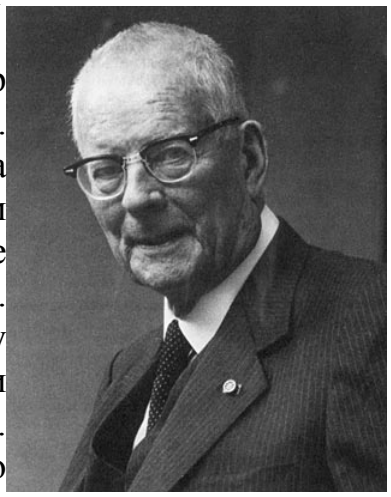

Рисунок 7. Уильям Эдвард Деминг

Его также называют наставником номер один по менеджменту качества (Гуру). Подробности о жизни и вкладе доктора Деминга в науку о всеобщем управлении качеством (TQM) можно найти в www.deming.ru. Развитие TQM началось с середины XX века. В.А. Лапидус[5] [12] указывает, что: “…На смену тейлоризму с его четким разделением функций и обязанностей стали приходить новые подходы. Высшее руководство компаний уже не могло просто делегировать одному из отделов задачу обеспечения качества. Оно вынуждено было само возглавить работы по качеству, создавая специальные системы качества, вовлекать весь персонал компаний в работы по качеству, обеспечивая при этом четкость и ясность в вопросах ответственности, полномочий и взаимодействия всего персонала и прежде всего высшего менеджмента. Одновременно качество становилось стратегией компаний, затрагивая все более долговременные задачи развития организации[6] с целью достижения долгосрочного успеха на основе постоянного, непрерывного улучшения качества.

Рисунок 8. Лапидус Вадим Аркадьевич, д.т.н., Академик Международной Академии Качества

[5] Вадим Аркадьевич Лапидус, д. т. н , первый заместитель генерального директора АО НИЦ КД, генеральный директор СМЦ «Приоритет», профессор Государственного университета Высшей школы экономики (г. Н. Новгород); Председатель ИСО/ТК 69/ПК4 "Применение статистических методов" (1991–1999); Председатель ТК 125 "Статистические методы в управлении качеством продукции" Госстандарта РФ (1991-2002); Ответственный редактор журнала "Методы менеджмента качества"; Член Американского общества по качеству (ASQ) (с 1996 г.); Член Южно–Итальянского общества по качеству (AICQ) (с 1995 г.); Академик Международной Академии Качества (IAQ) (с 2003 г.); Президент Международной Гильдии профессионалов качества; Академик Академии проблем качества РФ; Аудитор AFAQ ASCERT International.

[6] Группа работников и необходимых средств с распределением ответственности, полномочий и взаимоотношений. ГОСТ Р ИСО 9000-2001.

Это в свою очередь привело к пониманию того, что новый менеджмент качества, который получил международно-принятую аббревиатуру TQM (Total Quality Management), должен стать основой новой корпоративной культуры предприятий …". В этой же работе В.А. Лапидус приводит слова Арманда Фейгенбаума[7]: "…два вида деятельности (внедрение ИСО 9000 и принципов TQM) являются партнерами в достижении единой цели, но на разных стадиях движения предприятий к качеству. При этом базисом систем качества являются стандарты ИСО серии 9000, а эволюционным развитием — TQM…".

Рисунок 9. Арманд В. Фейгенбаум Академик Международной Академии Качества

Развитие TQM будет продолжаться долго. За последние годы возникли различные методики, направленные на достижение всеобщего менеджмента качества, некоторые из которых, как, например методика евро-американского опыта - "Шесть сигм", уже конкурируют с TQM [13].

Как пишет в уже упомянутой выше работе [11] д-р Грегори Ватсон: "...Основы методологии "Шесть сигм" были заложены инженером корпорации Motorola Биллом Смитом, который применил ее в качестве средства экспериментальной проверки стратегии совершенствования работы корпорации, предложенной ее генеральным директором Бобом Гелвином и нацеленной на повышение эффективности компании в 100 раз за пять лет. Чтобы создать методологию, позволяющую решить столь сложную задачу, Motorola взаимодействовала с большим числом компаний — поставщиков комплектующих полупроводниковых приборов, участвовавших в работе созданного корпорацией научно-исследовательского института (Six Sigma Research Institute), возглавляемого д-ром Майклом Дж. Харри. Разработанную методологию затем применил генеральный директор AlliedSignal Лари Боссиди (Larry Bossidy), совершив с ее помощью коренной переворот в работе своей организации. Но после того как Джек Уэлш адаптировал методологию "Шесть сигм" к решению задач, стоявших перед возглавляемой им корпорацией General Electric, она стала главной методологией менеджмента, применяемой многими сотнями компаний по всему миру...".

Отметим, что подход методологии "Шесть сигм" к совершенствованию бизнеса интересен тем, что он позволяет найти и исключить причины ошибок или дефектов в бизнес-процессах организации.

[7] Арманд В. Фейгенбаум (Armand W. Feigenbaum) - Академик Международной Академии Качества (IAQ). Наряду с Э. Демингом, Дж. Джуран, Ф. Кросби, К. Исикава, Т. Тагути и Т. Сейфис - удостоен именоваться гуру в области менеджмента качества.

Как мы видим, идеи всеобщего менеджмента качества являются крупнейшей научной и практической школой.

Этап всеобщего управления качеством характеризуется применением множества моделей качества и стандартов, прежде всего - ИСО для создания комплексных систем управления с учетом особенностей конкретного предприятия в конкретной стране, определившего направление своего развития, которое отличает его от соперников и отвечает реальным нуждам потребителей.

В качестве примера рассмотрим стандарт ИСО, используя ГОСТ Р ИСО 9000-2001. ОСНОВНЫЕ ПОЛОЖЕНИЯ И СЛОВАРЬ:

Серия стандартов ISO 9000:2000 устанавливает общие требования к системе менеджмента качества любой организации, желающей продемонстрировать свою способность стабильно давать продукцию, отвечающую требованиям потребителя и соответствующим нормативным требованиям, и способствующую повышению степени удовлетворенности потребителя. Стандарт основан на трех основополагающих идеях:

- ориентация на потребителя,
- процессный подход,
- постоянное улучшение.

Ориентация на потребителя означает, что качество (продукта, услуги) определяется как степень удовлетворенности потребителя. Вопросы организации взаимодействия с потребителем занимают важное место в стандарте.

Процессный подход подразумевает, что качественно работающая организация имеет установленные процессы, и что качество процесса предопределяет качество продукта. Требования стандарта определяют, что процессы в организации должны быть установлены (то есть надежно воспроизводиться, не быть случайными и рассчитанными на удачу) и должны быть определены (то есть должны быть детально документированы используемые операции, регламенты, формы документов и т.д.).

Принцип постоянного улучшения требует, чтобы все основные процессы были измеримыми и чтобы были установлены и определены процессы постоянного улучшения основных процессов.

Стандарт имеет развитую систему поддержки сертификации, то есть подтверждения соответствия стандарту. Сертификация производится независимыми специализированными органами по сертификации, имеющими международную признанную аккредитацию. Аудиторы органа по сертификации проверяют, что все процессы в организации установлены и документированы, и что процессы фактически выполняются в точном соответствии с тем, как это описано в документах организации. В обязанности аудиторов ни в коем случае не

входит определение того, какие процессы являются «правильными» для данной организации. Сертификация удостоверяет, что организация делает ровно то, что обещает, не более, но и не менее. Поэтому стандарты этой серии являются гибкими, динамичными и наиболее широко используемыми во всем мире.

Следует подчеркнуть, что применение стандартов серии ISO 9000 в организациях по разработке программного обеспечения является довольно трудоемким. Стандарты этой серии ориентированы на систему менеджмента качества любой организации в целом, и практически не касаются специальных вопросов управления разработкой программных продуктов. Организации надлежит самостоятельно детально проработать все вопросы разработки программного продукта в соответствии с буквой и духом стандарта. С другой стороны, сертификация по ISO 9000 действительно подтверждает не только то, что организация может однократно произвести качественный программный продукт, но и то, что организация постоянно прогрессирует, ориентирована не на себя, а на внешний мир и работает по строгим, прозрачным для внешнего наблюдателя правилам.

Стандарты ИСО (ISO/IEC) 9000:2000 вводят следующее достаточно общее определение качества продукции:

Качество – это степень удовлетворенности потребителя[8].

Такое определение, несмотря на его чрезвычайно общий характер, делающий его трудным в практическом применении (оно не дает никаких практических способов измерить качество продукции в виде численного показателя) вводит важный общий принцип современного управления качеством – *ориентацию на потребителя*.

С момента появлением стандартов ИСО 9000:2000 мы уже не можем отделять качество создаваемого продукта от качества процесса его производства. Успешное руководство организацией, производящей продукт, и эффективность её функционирование достигаются за счет постоянного улучшения деятельности организации с учетом потребностей всех заинтересованных сторон. Создание или модернизация предприятия всегда связано с внедрением и поддержанием в рабочем состоянии системы управления предприятием. Качество системы управления предприятия прямо связано с качеством производимой продукции.

Как всё же нам подойти к измерениям и количественной оценке качества продукции? Эти оценки необходимы для любой системы управления качеством, так как для того, чтобы управлять каким-либо процессом, надо, прежде всего, уметь измерять его параметры. Дело в

[8] Качество: Степень соответствия присущих характеристик требованиям. ГОСТ Р ИСО 9000-2001.

том, что проблема измерения и количественной оценки качества продукции в настоящее время является научной проблемой о качестве продукции. Отрасль науки, изучающая и реализующая методы количественной оценки качества продукции, называется квалиметрией [14]. Назначение квалиметрии состоит в определении количественных оценок качественным характеристикам товара. Квалиметрия исходит из того, что качество зависит от большого числа свойств рассматриваемого продукта. Для того чтобы судить о качестве продукта, недостаточно только данных о его свойствах. Нужно учитывать и условия, в которых продукт будет использован. Согласно квалиметрии, численные значения параметров качества могут быть получены, если потребитель в состоянии группировать свойства в порядке их важности. Качество в связи с этим – величина измеримая и, следовательно, несоответствие продукта предъявляемым к нему требованиям может быть выражено через какую-либо постоянную меру, которой обычно являются деньги.
С позиций стандартов серии ИСО 9000 следует усвоить, что:

- Всякая работа выполняется как процесс[9];
- Каждый процесс имеет вход;
- Результатом процесса является выход;
- Всегда имеются благоприятные возможности для выполнения измерений на входе и в различных местах процесса, а также на выходе из процесса.

Рисунок 10. Всякая работа выполняется как процесс

[9] Совокупность взаимосвязанных и взаимодействующих видов деятельности, преобразующая входы в выходы. ГОСТ Р ИСО 9000-2001.

Процесс сам по себе является преобразованием, которое добавляет ценность. Каждый процесс включает определенным образом трудовые и (или) другие ресурсы. Выход из процесса представляет собой материальную или нематериальную продукцию. Выходом может быть, например, фактура, программный продукт, жидкое топливо, прибор для клиники, банковская услуга или промежуточная продукция любой общей категории.

Рисунок 11. Модель системы менеджмента качества

На графическом изображении модели системы менеджмента качества, основанной на процессах, заметна роль потребителей при определении входных данных.

Традиционное изображёние модели системы менеджмента качества, показанное здесь, охватывает все требования стандарта ИСО, несмотря на то, что на ней детально не обозначены процессы. Мы видим, что деятельность организации и соответствующие ресурсы представлены процессами. Управление процессами и позволяет эффективно управлять предприятием.

В стандарте ИСО определены восемь принципов менеджмента качества, краткое рассмотрение которых позволяет уточнить представление о качестве продукта и определить роль и место процессного подхода для применения в системе управления предприятием.

Эти восемь принципов менеджмента качества образуют основу для стандартов на системы менеджмента качества, входящих в семейство ИСО 9000[15]:

- Ориентация на потребителя

Организации зависят от своих потребителей, и поэтому должны понимать их текущие и будущие потребности, выполнять их требования и стремиться превзойти их ожидания.

- Лидерство руководителя

Руководители обеспечивают единство цели и направления деятельности организации. Им следует создавать и поддерживать внутреннюю среду, в которой работники могут быть полностью вовлечены в решение задач организации.

- Вовлечение работников

Работники всех уровней составляют основу организации, и их полное вовлечение дает возможность организации с выгодой использовать их способности.

- Процессный подход

Желаемый результат достигается эффективнее, когда деятельностью и соответствующими ресурсами управляют как процессом.

Эффективное функционирование организации возможно при идентификации множества взаимосвязанных между собой видов деятельности и управления ими. Виды деятельности, использующие ресурсы и управляемые в определенном порядке, позволяющем преобразовать “входы” в “выходы”, рассматриваются как процессы. Поскольку “выходы” одних процессов связаны с “входами” других процессов, то у нас имеется возможность достижения высокой степени детализации при описании бизнес-процессов организации. Применение системы процессов совместно с их идентификацией и взаимодействием в рамках организации, а также управление этими процессами, представляет собой “процессный подход”. Преимущество

процессного подхода заключается в непрерывном управлении, которое обеспечивает хорошую взаимосвязь, как между отдельными процессами, так и их комбинацией и взаимодействием.

Применение процессного подхода в рамках системы менеджмента[10] качества организации позволяет достигать:

- понимания и выполнения требований;
- необходимости рассмотрения процессов в терминах “добавленной ценности”;
- получения результатов выполнения процессов и результативности, и непрерывного улучшения процессов, основанного на объективных измерениях.

- Системный подход к менеджменту

Выявление, понимание и менеджмент взаимосвязанных процессов как системы содействуют результативности и эффективности организации при достижении ее целей.

- Постоянное улучшение

Постоянное улучшение деятельности организации в целом следует рассматривать как ее неизменную цель.

- Принятие решений, основанное на фактах

Эффективные решения основываются на анализе данных и информации.

- Взаимовыгодные отношения с поставщиками

Организация и ее поставщики взаимозависимы, и отношения взаимной выгоды повышают способность обеих сторон создавать ценности.

[10] Система менеджмента для руководства и управления организацией применительно к качеству. ГОСТ Р ИСО 9000-2001.

1.5. Термины и определения ИСО (ISO/IEC) 9000:2000

Термин	Определение	Отношение к
Анализ	Деятельность, предпринимаемая для установления пригодности, адекватности, **результативности** рассматриваемого объекта для достижения установленных целей. Примечание — Анализ может также включать определение **эффективности**. Примеры: анализ со стороны руководства, анализ проектирования и разработки, анализ требований потребителей и анализ несоответствий.	**оценке**
Аудит (проверка)	Систематический, независимый и документированный **процесс** получения **свидетельств аудита (проверки)** и объективного их оценивания с целью установления степени выполнения согласованных **критериев аудита (проверки)**. Примечание — Внутренние аудиты (проверки), иногда называемые «аудиты (проверки) первой стороной», проводятся обычно самой **организацией** или от ее имени для внутренних целей могут служить основанием для декларации о **соответствии**. Внешние аудиты (проверки) включают аудиты, обычно называемые «аудиты (проверки) второй стороной» или «аудиты (проверки) третьей стороной». Аудиты (проверки) второй стороной проводятся сторонами, заинтересованными в деятельности организации, например потребителями или другими лицами от их имени. Аудиты (проверки) третьей стороной проводятся внешними независимыми организациями. Эти организации осуществляют сертификацию или регистрацию на соответствие требованиям, например требованиям ГОСТ Р ИСО 9001 и ГОСТ Р ИСО 14001. Если **системы менеджмента** качества и охраны окружающей среды вместе подвергаются аудиту (проверке), это называется «комплексным аудитом». Если две или несколько организаций проводят совместно аудит (проверку) **проверяемой организации**, это называется «совместным аудитом».	**аудиту (проверке)**

аудитор	Лицо, обладающее **компетентностью** для проведения **аудита (проверки)**.	**аудиту (проверке)**
Валидация	Подтверждение на основе представления **объективных свидетельств** того, что **требования**, предназначенные для конкретного использования или применения, выполнены. Примечания: 1. Термин «подтверждено» используется для обозначения соответствующего статуса, 2. Условия применения могут быть реальными или смоделированными.	**оценке**
Верификация	Подтверждение на основе представления **объективных свидетельств** того, что установленные **требования** были выполнены. Примечания: 1. Термин “верифицировано” используется для обозначения соответствующего статуса. 2. Деятельность по подтверждению может включать: -осуществление альтернативных расчетов; - сравнение **научной и технической документации** по новому проекту с аналогичной документацией по апробированному проекту, - проведение **испытаний** и демонстраций; - анализ документов до их выпуска.	**оценке**
Возможности	Способность **организации**, **системы** или **процесса** производить **продукцию**, которая будет соответствовать **требованиям** к этой продукции.	**качеству**
Выпуск	Разрешение на переход к следующей стадии процесса. Примечание — В английском языке, в контексте компьютерных программных средств, термином «release» часто называют версию самих программных средств.	**соответствию**
Высшее руководство	Лицо или группа работников, осуществляющих направление деятельности и управление **организацией** на высшем уровне.	**менеджменту**

Градация	Класс, сорт, категория или разряд, присвоенные различным **требованиям** к качеству **продукции**, **процессов** или **систем**, имеющих то же самое функциональное применение. Пример: класс авиабилета или категория гостиницы в справочнике гостиниц. Примечание — При определении требования к качеству градация обычно устанавливается.	**качеству**
Группа по аудиту (проверке)	Один или несколько аудиторов, проводящих аудит **(проверку)**. Примечания: 1. Один из аудиторов в группе по аудиту (проверке), как правило, назначается руководителем группы по аудиту. 2. Группа по аудиту может включать стажеров и, в случае необходимости, **технических экспертов**. 3. В работе группы могут принимать участие наблюдатели без полномочий членов группы по аудиту.	**аудиту (проверке)**
Дефект	Невыполнение **требования**, связанного с предполагаемым или установленным использованием. Примечания: 1. Различие между понятиями дефект и **несоответствие** является важным, так как имеет подтекст юридического характера, связанный с вопросами ответственности за качество продукции. Следовательно, термин «дефект» надо использовать чрезвычайно осторожно. 2. Использование, предполагаемое **потребителем**, может зависеть от характера информации, такой как инструкции по использованию и техническому обслуживанию, предоставляемые поставщиком.	**соответствию**

Документ	**Информация** и соответствующий носитель. Примеры: **записи**, **нормативная и техническая документация**, процедурный документ, чертеж, отчет, стандарт. Примечания: 1. Носитель может быть бумажным, магнитным, электронным или оптическим компьютерным диском, фотографией или эталонным образцом, или комбинацией из них. 2. Комплект документов, например технических условий и записей, часто называется «документацией». 3. Некоторые требования (например, требование к разборчивости) относятся ко всем видам документов, однако могут быть иные требования к техническим условиям (например требование к управлению пересмотрами) и записям (например требование к восстановлению).	**документации**
заинтересованная сторона	Лицо или группа, заинтересованные в деятельности или успехе организации. Примеры: потребители, владельцы, работники организации, поставщики, банкиры, ассоциации, партнеры или общество. Примечание — Группа может состоять из организации, ее части или из нескольких организаций.	**организации**
Заказчик аудита (проверки)	**Организация** или лицо, заказавшие аудит **(проверку)**.	**аудиту (проверке)**
Заключения по результатам аудита (проверки)	Выходные данные аудита, предоставленные **группой по аудиту (проверке)** после рассмотрения целей аудита и всех **наблюдений аудита**.	**аудиту (проверке)**
Запись	**Документ**, содержащий достигнутые результаты или свидетельства осущест-вленной деятельности. Примечания: 1. Записи могут использоваться, например, для документирования **прослеживаемости**, свидетельства проведения **верификации**, **предупреждающих действии** и **корректирующих действий**. 2. Обычно пересмотры записей не нуждаются в управлении.	**документации**

Измерительное оборудование	Средства измерения, программные средства, эталоны, стандартные образцы, вспомогательная аппаратура или комбинация из них, необходимые для выполнения **процесса измерения**.	**обеспечению качества процессов измерения**
Информация	Значимые данные.	**документации**
Инфраструктура	<организация> Совокупность зданий, оборудования и служб обеспечения, необходимых для функционирования **организации.**	**организации**
Испытание	Определение одной или нескольких **характеристик** согласно установленной **процедуре**.	**оценке**
Качество	Степень соответствия присущих характеристик **требованиям**. Примечания: 1. Термин “качество” может применяться с такими прилагательными, как плохое, хорошее или отличное. 2. Термин “присущий” в отличие от термина “присвоенный” означает имеющийся в чем-то. Прежде всего, это относится к постоянным характеристикам.	**качеству**
Компетентность	Выраженная способность применять свои знания и умение.	**аудиту (проверке)**
Контроль	Процедура оценивания соответствия путем наблюдения и суждений, сопровождаемых соответствующими измерениями, испытаниями или калибровкой.	**оценке**
Корректирующее действие	Действие, предпринятое для устранения причины обнаруженного **несоответствия** или другой нежелательной ситуации. Примечания: 1. У несоответствия может быть несколько причин. 2. Корректирующее действие предпринимается для предотвращения повторного возникновения события, тогда **как предупреждающее действие** — для предотвращения возникновения события. 3. Существует различие между **коррекцией** и корректирующим действием.	**соответствию**
Коррекция	Действие, предпринятое для устранения обнаруженного **несоответствия**. Примечания: 1. Коррекция **может** осуществляться в сочетании с **корректирующим действием**. 2. Коррекция может включать, например, **переделку** или **снижение градации**.	**соответствию**

Критерии аудита (проверки)	Совокупность политики, процедур или **требований**, которые применяются в виде ссылок.	**аудиту (проверке)**
Менеджмент	Скоординированная деятельность по руководству и управлению **организацией**. Примечание — В английском языке термин “management” иногда относится к людям, т.е. к лицу или группе работников, наделенных полномочиями и ответственностью для руководства и управления организацией. Когда “management” используется в этом смысле, его следует всегда применять с определяющими словами с целью избежания путаницы с понятием “management”, определенным выше. Например, не одобряется выражение “руководство должно ...”, в то время как “высшее руководство должно ...” — приемлемо.	**менеджменту**
Менеджмент качества	Скоординированная деятельность по руководству и управлению **организацией** применительно **к качеству**. Примечание — Руководство и управление применительно к качеству обычно включает разработку **политики в области-качества и целей в области качества**, **планирование качества**, **управление качеством**, **обеспечение качества** и **улучшение качества**.	**менеджменту**
Метрологическая служба	Организационная структура, несущая ответственность за определение и внедрение **системы управления измерениями**.	**обеспечению качества процессов измерения**
Метрологическая характеристика	Отличительная особенность, которая может повлиять на результаты измерения. Примечания: 1. Измерительное оборудование обычно имеет несколько метрологических характеристик. 2. Метрологические характеристики могут быть предметом калибровки.	**обеспечению качества процессов измерения**

Метрологическое подтверждение пригодности	Совокупность операций, необходимая для обеспечения соответствия **измерительного оборудования требованиям**, отвечающим его назначению. Примечания: 1. Метрологическое подтверждение пригодности обычно включает калибровку или верификацию, любую необходимую юстировку или ремонт и последующую перекалибровку, сравнение с метрологическими требованиями для предполагаемого использования оборудования, а также требуемое пломбирование и маркировку. 2. Метрологическое подтверждение пригодности не выполнено до тех пор, пока пригодность измерительного оборудования для использования по назначению не будет продемонстрирована и задокументирована. 3. Требования к использованию по назначению включают такие характеристики, как диапазон, разрешающая способность, максимально допустимые погрешности и т.д. 4. Требования к метрологическому подтверждению пригодности обычно отличаются от требований на продукцию и в них не регламентируются.	**обеспечению качества процессов измерения**
Наблюдения аудита (проверки)	Результат оценки **свидетельства аудита (проверки)** в зависимости от **критериев аудита (проверки)**. Примечание — Наблюдения аудита (проверки) могут указывать на соответствие или несоответствие критериям аудита (проверки) или на возможности улучшения.	**аудиту (проверке)**
Надежность	Собирательный термин, применяемый для описания свойства готовности и влияющих на него свойств безотказности, ремонтопригодности и обеспеченности технического обслуживания и ремонта. Примечание — Надежность применяется только для общего неколичественного описания свойства.	**характеристикам**
Несоответствие	Невыполнение **требования**.	**соответствию**

Нормативная и техническая документация	**Документы**, устанавливающие **требования**. Примечание: Нормативные документы могут относиться к деятельности (например, документированная процедура, технологическая документация на процесс или методику испытаний) или продукции (например, технические условия на продукцию, эксплуатационная документация и чертежи).	**документации**
Обеспечение качества	Часть **менеджмента качества**, направленная на создание уверенности, что **требования** к качеству будут выполнены.	**менеджменту**
Объективное свидетельство	Данные, подтверждающие наличие или истинность чего-либо. Примечание — Объективное свидетельство может быть получено путем наблюдения, измерения, **испытания** или другими способами.	**оценке**
Организационная структура	Распределение ответственности, полномочий и взаимоотношений между работниками. Примечания: 1. Распределение обычно бывает упорядоченным. 2. Официально оформленная организационная структура часто содержится в **руководстве по качеству** или в **плане качества проекта**. 3. Область применения организационной структуры может включать соответствующие взаимодействия с внешними **организациями**.	**организации**
Организация	Группа работников и необходимых средств с распределением ответственности, полномочий и взаимоотношений. Примеры: компания, корпорация, фирма, предприятие, учреждение, благотворительная организация, предприятие розничной торговли, ассоциация, а также их подразделения или комбинация из них. Примечания: 1. Распределение обычно бывает упорядоченным. 2. Организация может быть государственной или частной. 3. Настоящее определение действительно применительно к стандартам на **системы менеджмента качества**.	**организации**

Переделка	Действие, предпринятое в отношении несоответствующей **продукции**, с тем, чтобы она соответствовала **требованиям**. Примечание — В отличие от переделки **ремонт** может воздействовать на части несоответствующей продукции или изменять их.	**соответствию**
План качества	**Документ**, определяющий, какие **процедуры** и соответствующие ресурсы, кем и когда должны применяться к конкретному проекту, **продукции**, **процессу** или контракту.	**документации**
Планирование качества	Часть **менеджмента качества**, направленная на установление **целей в области качества** и определяющая необходимые операционные **процессы** жизненного цикла продукции и соответствующие ресурсы для достижения целей в области качества. Примечание — Разработка **планов качества** может быть частью планирования качества.	**менеджменту**
Политика в области качества	Общие намерения и направление деятельности **организации** в области **качества**, официально сформулированные **высшим руководством**. Примечания: 1. Как правило, политика в области качества согласуется с общей политикой организации и обеспечивает основу для постановки целей в области качества. 2. Принципы менеджмента качества, изложенные в настоящем стандарте, могут служить основой для разработки политики в области качества.	**менеджменту**
Поставщик	Организация или лицо, предоставляющие продукцию. Примеры: производитель, оптовик, предприятие розничной торговли или продавец продукции, исполнитель услуги, поставщик информации. Примечания: 1. Поставщик может быть внутренним или внешним по отношению к организации. 2. В контрактной ситуации поставщика иногда называют “подрядчиком”.	**организации**

Постоянное улучшение	Повторяющаяся деятельность по увеличению способности выполнить **требования**. Примечание — **Процесс** установления целей и поиска возможностей улучшения является постоянным процессом, использующим **наблюдения аудита (проверки)** и **заключения по результатам аудита (проверки)**, анализ данных, **анализ** со стороны руководства или другие средства и обычно ведущим **к корректирующим действиям** или **предупреждающим действиям**.	**менеджменту**
Потребитель	**Организация** или лицо, получающие продукцию. Примеры: клиент, заказчик, конечный пользователь, розничный торговец, бенефициар и покупатель. Примечание — Потребитель может быть внутренним или внешним по отношению к организации.	**организации**
Предупреждающее действие	Действие, предпринятое для устранения причины потенциального **несоответствия** или другой потенциально нежелательной ситуации. Примечания: 1. У потенциального несоответствия может быть несколько причин. 2. Предупреждающее действие предпринимается для предотвращения возникновения события, тогда как **корректирующее действие** — для предотвращения повторного возникновения события.	**соответствию**
Проверяемая организация	**Организация**, подвергающаяся **аудиту (проверке)**.	**аудиту (проверке)**
Программа аудита (проверки)	Совокупность одного или нескольких **аудитов (проверок)**, запланированных на конкретный период времени и направленных на достижение конкретной цели.	**аудиту (проверке)**

Продукция	Результат **процесса**. Примечания: 1. Имеются четыре общие категории продукции: - услуги (например, перевозки); - программные средства (например, компьютерная программа, словарь); - технические средства (например, узел двигателя); - перерабатываемые материалы (например, смазка). Многие виды продукции содержат элементы, относящиеся к различным общим категориям продукции. Отнесение продукции к услугам, программным или техническим средствам или перерабатываемым материалам зависит от преобладающего элемента. Например, поставляемая продукция «автомобиль» состоит из технических средств (например, шин), перерабатываемых материалов (горючее, охлаждающая жидкость), программных средств (программное управление двигателем, инструкция водителю) и услуги (разъяснения по эксплуатации, даваемые продавцом). 2. Услуга является результатом, по меньшей мере, одного действия, обязательно осуществленного при взаимодействии поставщика и потребителя, она, как правило, нематериальна. Предоставление услуги может включать, к примеру, следующее: - деятельность, осуществленную на поставленной потребителем материальной продукции (например, автомобиль, нуждающийся в ремонте); - деятельность, осуществленную на поставленной потребителем нематериальной продукции (например, заявление о доходах, необходимое для определения размера налога); - предоставление нематериальной продукции (например, информации в смысле передачи знаний); - создание благоприятных условий для потребителей (например, в гостиницах и ресторанах). Программное средство содержит информацию и обычно является нематериальным, может также быть в форме подходов, операций или процедуры. Техническое средство, как правило, является материальным, и его количество выражается исчисляемой характеристикой. Перерабатываемые материалы обычно являются материальными, и их количество выражается непрерывной характеристикой. Технические средства и перерабатываемые материалы часто называются товарами. 3. Обеспечение качества направлено главным образом на предполагаемую продукцию.	**процессам и продукции**

Проект	Уникальный **процесс**, состоящий из совокупности скоординированной и управляемой деятельности с начальной и конечной датами, предпринятый для достижения цели, соответствующей конкретным **требованиям**, включающий ограничения сроков, стоимости и ресурсов. Примечания: 1. Отдельный проект может быть частью структуры более крупного проекта. 2. В некоторых проектах цели совершенствуются, а характеристики продукции определяются соответственно по мере развития проекта. 3. Выходом проекта может быть одно изделие или несколько единиц продукции.	**процессам и продукции**
Проектирование и разработка	Совокупность **процессов**, переводящих **требования** в установленные **характеристики** или **нормативную и техническую документацию** на **продукцию**, **процесс** (3.4.1) или **систему**. Примечания: 1. Термины «проектирование» и «разработка» иногда используют как синонимы, а иногда — для определения различных стадий процесса проектирования и разработки в целом. 2. Для обозначения объекта проектирования и разработки могут применяться определяющие слова (например, проектирование и разработка продукции или проектирование и разработка процесса).	**процессам и продукции**
Производственная среда	Совокупность условий, в которых выполняется работа. Примечание — Условия включают физические, социальные, психологические и экологические факторы (такие как температура, системы признания и поощрения, эргономика и состав атмосферы).	**организации**
Прослеживаемость	Возможность проследить историю, применение или местонахождение того, что рассматривается. Примечания: При рассмотрении продукции прослеживаемость может относиться к: - происхождению материалов и комплектующих; - истории обработки; - распределению и местонахождению продукции после поставки.	**характеристикам**

Процедура	Установленный способ осуществления деятельности или **процесса**. Примечания: 1. Процедуры могут быть документированными или недокументированными. 2. Если процедура документирована, часто используется термин «письменная процедура» или «документированная процедура». Документ, содержащий процедуру, может называться «документированная процедура».	**процессам и продукции**
Процесс	Совокупность взаимосвязанных и взаимодействующих видов деятельности, преобразующая входы в выходы. Примечания: 1. Входами к процессу обычно являются выходы других процессов. 2. Процессы в организации, как правило, планируются и осуществляются в управляемых условиях с целью добавления ценности. 3. Процесс, в котором подтверждение **соответствия** конечной продукции затруднено или экономически нецелесообразно, часто относят к “специальному процессу”.	**процессам и продукции**
Процесс измерения	Совокупность операций для установления значения величины.	**обеспечению качества процессов измерения**
Процесс квалификации	**Процесс** демонстрации способности выполнить установленные **требования**. Примечания: 1. Термин «квалифицирован» используется для обозначения соответствующего статуса. 2. Квалификация может распространяться на работников, **продукцию**, процессы или **системы**. Пример: квалификация аудиторов (экспертов по сертификации систем качества), квалификация материала.	**оценке**
Разрешение на отклонение	Разрешение на использование или **выпуск продукции**, которая не соответствует установленным **требованиям**. Примечание — Разрешение на отклонение обычно распространяется на поставку продукции с несоответствующими характеристиками для установленных согласованных ограничений по времени или количеству данной продукции.	**соответствию**

Разрешение на отступление	Разрешение на отступление от исходных установленных **требований к продукции** до ее производства. Примечание — Разрешение на отступление, как правило, дается на ограниченное количество продукции или период времени, а также для конкретного использования.	**соответствию**
Результативность	Степень реализации запланированной деятельности и достижения запланированных результатов.	**менеджменту**
Ремонт	Действие, предпринятое в отношении несоответствующей **продукции**, чтобы сделать ее приемлемой для предполагаемого использования. Примечания: 1. Ремонт включает действие по исправлению, предпринятое в отношении ранее соответствовавшей продукции для ее восстановления с целью использования, например, как часть технического обслуживания. 2. В отличие от **переделки** ремонт может воздействовать на части несоответствующей продукции или изменять их.	**соответствию**
Руководство по качеству	**Документ**, определяющий **систему менеджмента качества организации**. Примечание — Руководства по качеству могут различаться по форме и детальности изложения, исходя из соответствия размеру и сложности организации.	**документации**
Свидетельство аудита (проверки)	**Записи**, изложение фактов или другой **информации**, связанной с **критериями аудита (проверки)**, которая может быть перепроверена. Примечание — Свидетельство аудита (проверки) может быть качественным или количественным.	**аудиту (проверке)**
Система	Совокупность взаимосвязанных и взаимодействующих элементов.	**менеджменту**
Система менеджмента	**Система** для разработки политики и целей и достижения этих целей. Примечание — Система менеджмента организации может включать различные системы менеджмента, такие как система менеджмента качества, система менеджмента финансовой деятельности или система менеджмента охраны окружающей среды.	**менеджменту**

Система менеджмента качества	Система менеджмента для руководства и управления организацией применительно к качеству.	**менеджменту**
Система управления измерениями	Совокупность взаимосвязанных или взаимодействующих элементов, необходимых для достижения **метрологического подтверждения пригодности** и постоянного управления **процессами измерения**.	**обеспечению качества процессов измерения**
Снижение градации	Изменение **градации** несоответствующей **продукции**, чтобы она соответствовала **требованиям**, отличным от исходных.	**соответствию**
Соответствие	Выполнение **требования**. Примечание: В английском языке термин «conformance» является синонимом, но он вызывает возражения.	**соответствию**
Технический эксперт	<аудит> Лицо, обладающее специальными знаниями или опытом применительно к объекту, подвергаемому аудиту. Примечания: 1. Специальные знания или опыт включают знания или опыт применительно **к организации**, **процессу** или деятельности, подвергаемым аудиту, а также знание языка и культуры страны, где проводится аудит. 2. Технический эксперт не имеет полномочий **аудитора** в группе **по аудиту** (проверке).	**аудиту (проверке)**
Требование	Потребность или ожидание, которое установлено, обычно предполагается или является обязательным. Примечания: 1. “Обычно предполагается” означает, что это общепринятая практика организации, ее потребителей и других заинтересованных сторон, когда предполагаются рассматриваемые потребности или ожидания. 2. Для обозначения конкретного вида требования могут применяться определяющие слова, например, требование к продукции, требование к системе качества, требование потребителя. 3. Установленным является такое требование, которое определено, например, в документе. 4. Требования могут выдвигаться различными заинтересованными сторонами.	**качеству**

Удовлетворенность потребителей	Восприятие потребителями степени выполнения **их требований**. Примечания: 1. Жалобы потребителей являются показателем низкой удовлетворенности потребителей, однако их отсутствие не обязательно предполагает высокую удовлетворенность потребителей. 2. Даже если требования потребителей были с ними согласованы и выполнены, это не обязательно обеспечивает высокую удовлетворенность потребителей.	**качеству**
Улучшение качества	Часть **менеджмента качества**, направленная на увеличение способности выполнить **требования** к качеству. Примечание — Требования могут относиться к любым аспектам, таким **как результативность**, **эффективность** или **прослеживаемостъ**.	**менеджменту**
Управление качеством	Часть **менеджмента качества** направленная на выполнение **требований** к качеству.	**менеджменту**
Утилизация несоответствующей продукции	Действие в отношении несоответствующей **продукции**, предпринятое для предотвращения ее первоначального предполагаемого использования. Примеры: переработка, уничтожение. Примечание — В ситуации с несоответствующей услугой применение предотвращается посредством прекращения услуги.	**соответствию**
Характеристика	Отличительное свойство. Примечания: 1. Характеристика может быть собственной или присвоенной. 2. Характеристика может быть качественной или количественной. 3. Существуют различные классы характеристик, такие как: - физические (например, механические, электрические, химические или биологические характеристики); - органолептические (например, связанные с запахом, осязанием, вкусом, зрением, слухом); - этические (например, вежливость, честность, правдивость); - временные (например пунктуальность, безотказность, доступность); - эргономические (например физиологические характеристики или связанные с безопасностью человека); - функциональные (например, максимальная скорость самолета).	**характеристикам**

Характеристика качества	Присущая **характеристика продукции**, **процесса** или **системы**, вытекающая из **требования**. Примечания: 1. «Присущая» означает имеющаяся в чем-то. Прежде всего, это относится к постоянной характеристике. 2. Присвоенные характеристики продукции, процесса или системы (например, цена продукции, владелец продукции) не являются характеристиками качества этой продукции, процесса или системы.	**характеристикам**
Цели в области качества	Цели, которых добиваются или к которым стремятся в области **качества**. Примечания: 1. Цели в области качества обычно базируются на политике организации в области качества. 2. Цели в области качества обычно устанавливаются для соответствующих функций и уровней организации.	**менеджменту**
Эффективность	Связь между достигнутым результатом и использованными ресурсами.	**менеджменту**

1.6. Выводы

Изучая раздел “Понятие качества продукта и процесса”, мы пришли к выводам о том, что:

- Качество продукции является измеримой величиной и может быть выражено через какую-либо постоянную меру;
- Всякая работа выполняется как процесс;
- Каждый процесс имеет вход, результатом процесса является выход. Процесс сам по себе является преобразованием, которое добавляет ценность;
- Модель системы менеджмента качества предполагает, что деятельность организации и соответствующие ресурсы представлены процессами;
- Управление процессами позволят эффективно управлять предприятием.

1.7. Вопросы и задания для самостоятельной работы студента по теме «Понятие качества продукта и процесса»

1) Дайте определение понятию «качества» как философской категории.
2) Какие качественные преимущества, применяемые при разработке и сопровождении операционной системы Linux, можно назвать в сравнении с другими подходами обеспечения качества ПО?
3) Дайте определение понятию «качества» в соответствии стандарту ГОСТ Р ИСО 9000-2000.
4) Дайте определение любому из терминов: «Анализ, Аудит, Аудитор, Валидация, Верификация, Возможности, Выпуск, Высшее руководство, Градация, Группа по аудиту (проверке), Дефект, Документ, Заинтересованная сторона, Заказчик аудита (проверки), Заключения по результатам аудита (проверки), Запись, Измерительное оборудование, Информация, Инфраструктура, Испытание, Качество, Компетентность, Контроль, Корректирующее действие, Коррекция, Критерии аудита (проверки), Менеджмент, Менеджмент качества, Метрологическая служба, Метрологическая характеристика, Метрологическое подтверждение пригодности, Наблюдения аудита (проверки), Надежность, Несоответствие, Нормативная и техническая документация, Обеспечение качества, Объективное свидетельство, Организационная структура,

Организация, Переделка, План качества, Планирование качества, Политика в области качества, Поставщик, Постоянное улучшение, Потребитель, Предупреждающее действие, Проверяемая организация, Программа аудита (проверки), Продукция, Проект, Проектирование и разработка, Производственная среда, Прослеживаемость, Процедура, Процесс, Процесс измерения, Процесс квалификации, Разрешение на отклонение, Разрешение на отступление, Результативность, Ремонт, Руководство по качеству, Свидетельство аудита (проверки), Система, Система менеджмента, Система менеджмента качества, Система управления измерениями, Снижение градации, Соответствие, Технический эксперт, Требование, Удовлетворенность потребителей, Улучшение качества, Управление качеством, Утилизация несоответствующей продукции, Характеристика, Характеристика качества, Цели в области качества, Эффективность».

5) Что понимается под «управлением качеством»?
6) В чем заключается концепция всеобщего качества (TQM)?
7) Что такое «система качества»?

Литература по теме «Понятие качества продукта и процесса»

[1] Ильин А.А., Спасение в качестве. "Русский колокол". 1928. № 4
[2] Энгельс Ф., см. Маркс К. и Энгельс Ф., Соч., 2 изд., т. 20
[3] Мищенко С.В., Пономарев С.В., Пономарева Е.С., Евлахин Р.Н., Мозгова Г.В.. ИСТОРИЯ МЕТРОЛОГИИ, СТАНДАРТИЗАЦИИ, СЕРТИФИКАЦИИ И УПРАВЛЕНИЯ КАЧЕСТВОМ, Тамбов: Изд-во Тамб. гос. техн. ун-та, 2004 г.
[4] Момот А.И. Менеджмент качества: Учебное пособие для вузов. Донецк: ДонГТУ, 2000 г.
[5] Новоторговый устав (Текст), http://school-collection.informika.ru/catalog/res/7A9A40A9-0A01-01B2-0125-EE49A0C34B8B/view/
[6] Хейли Артур. Колеса. АСТ, 2003 г.
[7] Taylor, F. W. (1911) The Principles of Scientific Management, Harper & Row, New York. Смотрите также http://marxists.nigilist.ru/reference/subject/economics/taylor/principles/ch01.htm и http://orel.rsl.ru/nettext/foreign/teilor/tailsod.htm
[8] Терещенко В.И., Организация и управление опыт США, Изд-во: Экономика, 1965 г.
[9] Адлер Ю. П., Хунузиди Е. И., Шпер В. Л., Методы постоянного совершенствования сквозь призму цикла Шухарта-Деминга. "Методы менеджмента качества", №3, 2005 г.

[10] Reclusive Linux founder opens. Friday, May 19, 2006 Posted: 0954 GMT (1754 HKT.
http://edition.cnn.com/2006/BUSINESS/05/18/global.office.linustorvalds/
[11] Качество в XXI веке. Роль качества в обеспечении конкурентоспособности и устойчивого развития: пер. с англ. / ред.-сост. Конти Тито, Кондо Ёсло и Ватсон Грегори. - М.: Стандарты и качество, 2005 г.
[12] Лапидус В.А.. Всеобщее качество (TQM) в российских компаниях. ОАО Типография Новости, 2002 г.
[13] Пэнди Питер С., Ньюмен Роберт П., Кэвенег Роланд Р., Курс на Шесть Сигм: Как General Electric, Motorola и другие ведущие компании мира совершенствуют свое мастерство, "ЛОРИ", 2002 г.
[14] Хамханова Д.Н. ОСНОВЫ КВАЛИМЕТРИИ. Учебное пособие для студентов специальностей 190800 «Метрология и метрологическое обеспечение», 072000 «Стандартизация и сертификация (по отраслям пищевой промышленности)» и 340100 «Управление качеством». Издательство ВСГТУ Улан-Удэ, 2003 г.
[15] ГОСТ Р ИСО 9000-2001

Тема 2. Атрибуты качества продукта и их характеристики

2.1. Общие положения о зарактеристиках качества ПО.

Практика выполнения работ по разработке и внедрению программного обеспечения характерна отсутствием в технических заданиях и реализованных проектах сведений или упоминаний о характеристиках качества программного продукта и как их следует измерять и сравнивать с требованиями, отраженными в контракте, техническом задании или спецификациях. Некоторые из характеристик часто отсутствуют в требованиях на разрабатываемые программные средства. Нечеткое декларирование в документах понятий и требуемых значений характеристик качества программных средств вызывает конфликты между заказчиками-пользователями и разработчиками-поставщиками из-за разной трактовки одних и тех же характеристик.

Некоторые программы предназначены для получения результата – например, математические или физические расчеты. Даже в таких случаях важно, как выполнялась разработка программы, на основе какой технологии, на каком языке она написана, была ли она документирована. Если же программа будет долго эксплуатироваться, или ее сопровождение будет сложным и важна сама программа, а не единичный результат выполнение группы расчетов, - и в этом случае нам необходимо применять термин «качество программы».

Для определения качества функционирования, наличия технических возможностей программных средств к взаимодействию, совершенствованию и развитию необходимо использовать стандарты в области оценки характеристик их качества.

Основой регламентирования показателей качества программных средств в Российской Федерации является международный стандарт ISO 9126:1991 (ГОСТ Р ИСО / МЭК 9126-93) "Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению". В России в области обеспечения жизненного цикла и качества сложных комплексов программ в основном применяется группа устаревших ГОСТов, которые отстают от мирового уровня на 5-10 лет.

В международном комитете ИСО формализован проект стандарта состоящего из четырех частей ISO 9126-1--4 для замены небольшой редакции 1991 года. Проект состоит из следующих частей под общим заголовком "Информационная технология - характеристики и метрики качества программного обеспечения": "Часть 1. Характеристики и субхарактеристики качества" Часть 2. Внешние метрики качества"

"Часть 3. Внутренние метрики качества" "Часть 4. Метрики качества в использовании".

Часть 1 стандарта - ISO 9126-1 - распределяет атрибуты качества программных средств по шести характеристикам, используемым в остальных частях стандарта. Все характеристики качества могут быть объединены в три группы, к которым применимы разные категории метрик:

- Категорийные, или описательные (номинальные) метрики. Категорийные метрики - наиболее адекватны функциональным возможностям программных средств;
- Количественные метрики - применимы для измерения надежности и эффективности сложных комплексов программ;
- Качественные метрики - в наибольшей степени соответствуют практичности, сопровождаемости и мобильности программных средств.

В части стандарта ISO 9126-1 даются определения с уточнениями из остальных его частей для каждой характеристики программного средства, а также для субхарактеристик качества.

Вторая и третья части стандарта - ISO 9126-2 и ISO 9126-3 - посвящены формализации соответственно внешних и внутренних метрик характеристик качества сложных программных средств. Все таблицы содержат унифицированную рубрикацию, где отражены имя и назначение метрики; метод ее применения; способ измерения, тип шкалы метрики; тип измеряемой величины; исходные данные для измерения и сравнения; а также этапы жизненного цикла программного средства (по ISO 12207), к которым применима метрика.

Четвертая часть стандарта - ISO 9126-4 - предназначена для покупателей, поставщиков, разработчиков, сопровождающих, пользователей и менеджеров качества программных средств. В ней обосновываются и комментируются выделенные показатели сферы (контекста) использования программных средств и группы выбранных метрик для пользователей.

Методологии и стандартизации оценки характеристик качества программных средств и их компонентов (программного продукта) на различных этапах жизненного цикла посвящен международный стандарт ISO 14598. В соответствии с ISO 14598, общая схема процессов оценки характеристик качества программных систем такова:

- Установка исходных требований для оценки это определение целей испытаний, идентификация типа метрик программного средства, выделение адекватных показателей и требуемых значений атрибутов качества;
- Селекция метрик качества, установление рейтингов и уровней приоритета метрик субхарактеристик и атрибутов,

выделение критериев для проведения экспертиз и измерений;

- Планирование и проектирование процессов оценки характеристик и атрибутов качества в жизненном цикле программного средства;
- Выполнение измерений для оценки, сравнение результатов с критериями и требованиями, обобщение и оценка результатов.

Для каждой характеристики качества рекомендуется формировать меры и шкалу измерений с выделением требуемых, допустимых и неудовлетворительных значений. Реализация процессов оценки должна коррелировать с этапами жизненного цикла конкретного проекта программного средства в соответствии с применяемой, адаптированной версией стандарта ISO 12207.

В работе [16], приведены основные атрибуты качества программного обеспечения, ознакомление с которыми обязательно в рамках данного курса, посредством самостоятельного изучения студентами материалов ГОСТ Р ИСО/МЭК 9126-93:

“…Функциональная пригодность - наиболее неопределенная и объективно трудно оцениваемая субхарактеристика программного средства. Области применения, номенклатура и функции комплексов программ охватывают столь разнообразные сферы деятельности человека, что невозможно выделить и унифицировать небольшое число атрибутов для оценки и сравнения этой субхарактеристики в различных комплексах программ.

Оценка корректности программных средств состоит в формальном определении степени соответствия комплекса реализованных программ исходным требованиям контракта, технического задания и спецификаций на программное средство и его компоненты. Путем верификации должно быть определено соответствие исходным требованиям всей совокупности к компонентов комплекса программ, вплоть до модулей и текстов программ и описаний данных.

Оценка способности к взаимодействию состоит в определении качества совместной работы компонентов программных средств и баз данных с другими прикладными системами и компонентами на различных вычислительных платформах, а также взаимодействия с пользователями в стиле, удобном для перехода от одной вычислительной системы к другой с подобными функциями.

Оценка защищенности программных средств включает определение полноты использования доступных методов и средств защиты программного средства от потенциальных угроз и достигнутой при этом безопасности функционирования информационной системы.

Оценка надежности - измерение количественных метрик атрибутов субхарактеристик в использовании: завершенности,

устойчивости к дефектам, восстанавливаемости и доступности/готовности.

Потребность в ресурсах памяти и производительности компьютера в процессе решения задач значительно изменяется в зависимости от состава и объема исходных данных. Для корректного определения предельной пропускной способности информационной системы с данным программным средством нужно измерить экстремальные и средние значения длительностей исполнения функциональных групп программ и маршруты, на которых они достигаются. Если предварительно в процессе проектирования производительность компьютера не оценивалась, то, скорее всего, понадобится большая доработка или даже замена компьютера на более быстродействующий.

Оценка практичности программных средств проводится экспертами и включает определение понятности, простоты использования, изучаемости и привлекательности программного средства. В основном это качественная (и субъективная) оценка в баллах, однако некоторые атрибуты можно оценить количественно по трудоемкости и длительности выполнения операций при использовании программного средства, а также по объему документации, необходимой для их изучения.

Сопровождаемость можно оценивать полнотой и достоверностью документации о состояниях программного средства и его компонентов, всех предполагаемых и выполненных изменениях, позволяющей установить текущее состояние версий программ в любой момент времени и историю их развития. Она должна определять стратегию, стандарты, процедуры, распределение ресурсов и планы создания, изменения и применения документов на программы и данные.

Оценка мобильности - качественное определение экспертами адаптируемости, простоты установки, совместимости и замещаемости программ, выражаемое в баллах. Количественно эту характеристику программного средства и совокупность ее атрибутов можно (и целесообразно) оценить в экономических показателях: стоимости, трудоемкости и длительности реализации процедур переноса на иные платформы определенной совокупности программ и данных”.

2.2. ГОСТ Р ИСО/МЭК 9126-93. Краткое описание стандарта

Публикуемые аннотации к стандарту ГОСТ Р ИСО/МЭК 9126-93 [17], содержат справку по библиографии[11]:

Обозначение	ГОСТ Р ИСО/МЭК 9126-93
Заглавие на русском языке	Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению
Заглавие на английском языке	Information technology. Software product evaluation. Quality characteristics and guidelines for their use
Дата введения в действие	01.07.1994
Дата огр. срока действия	
ОКС	35.080
Код ОКП	
Код КГС	П85
Код ОКСТУ	4002
Индекс рубрикатора ГРНТИ	5041

[11] http://www.standards.ru/document/4122576.aspx

Аннотация (область применения)	Настоящий стандарт определяет шесть характеристик, которые с минимальным дублированием описывают качество программного обеспечения. Данные характеристики образуют основу для дальнейшего уточнения и описания качества программного обеспечения. Руководства описывают использование характеристик качества для оценки качества программного обеспечения. Настоящий стандарт не определяет подхарактеристики (комплексные показатели) и показатели, а также методы измерения, ранжирования и оценки. Определения характеристик и соответствующая модель процесса оценки качества, приведенные в настоящем стандарте, применимы тогда, когда определены требования для программной продукции и оценивается ее качество в процессе жизненного цикла. Эти характеристики могут применяться к любому виду программного обеспечения, включая программы ЭВМ и данные, входящие в программно-технические средства (встроенные программы). Настоящий стандарт предназначен для характеристик, связанных с приобретением, разработкой, эксплуатацией, поддержкой, сопровождением или проверкой программного обеспечения
Ключевые слова	обработка данных; ЭВМ; программы ЭВМ; качество; характеристики
Термины и определения	
Наличие терминов РОСТЕРМ	
Вид стандарта	Основополагающие стандарты
Вид требований	Открытые системы
Дескрипторы (английский язык)	

Обозначение заменяемого(ых)	
Обозначение заменяющего	
Обозначение заменяемого в части	
Обозначение заменяющего в части	
Примечание	
Гармонизирован с:	
Аутентичный текст с ISO	ISO/IEC 9126:1991
Аутентичный текст с IEC	
Аутентичный текст с ГОСТ	
Аутентичный текст с прочими	
Содержит требования: ISO	
Содержит требования: IEC	
Содержит требования: СЭВ	
Содержит требования: ГОСТ	
Содержит требования: прочими	

Нормативные ссылки на: ISO	ISO/IEC 2382-20:1990;ISO 8402:1986
Нормативные ссылки на: IEC	
Нормативные ссылки на: ГОСТ	
Нормативные ссылки на: Прочие	
EN аутентичен ИСО/МЭК	
EN содержит требования ИСО/МЭК	
Документ внесен организацией СНГ	
Документ принят организацией СНГ	
Номер протокола	
Дата принятия	
Присоединившиеся страны	
Управление Ростехрегулирования	530 - Отдел стандартизации и сертификации информационных технологий, продукции электротехники и приборостроения
Технический комитет России	22 - Информационные технологии
Разработчик МНД	

Межгосударственный ТК	
Дата последнего издания (Дата изменения)	01.11.2004
Номер(а) изменении(й)	переиздание
Входит в сборник	
Количество страниц (оригинала)	12
Организация - Разработчик	
Статус	Действует

1 ОБЛАСТЬ ПРИМЕНЕНИЯ
2 НОРМАТИВНЫЕ ССЫЛКИ
3 ОПРЕДЕЛЕНИЯ

Измерение (measurement)	Действие по применению показателя качества программного обеспечения к конкретной программной продукции.
Качество (quality)	Весь объем признаков и характеристик продукции или услуги, который относится к их способности удовлетворять установленным или предполагаемым определены. Примечание - В сфере контракта потребности определены, тогда как в других сферах предполагаемые потребности должны быть установлены и потребностям (ИСО 8402).
Качество программного обеспечения (software quality)	Весь объем признаков и характеристик программной продукции, который относится к ее способности удовлетворять установленным или предполагаемым потребностям.
Критерий оценки качества программного обеспечения (software quality assessment criteria)	Набор определенных и задокументированных правил и условий, которые используются для решения о приемлемости общего качества конкретной программной продукции. Качество представляется набором установленных уровней, связанных с программной продукцией.

Метрика качества программного обеспечения (software quality metric)	Количественный Масштаб и метод, которые могут быть использованы для определения значения признака, принятого для конкретной программной продукции.
Оценка (assessment)	Действие по применению конкретного задокументированного критерия оценки к конкретному программному модулю, пакету или продукции с целью обусловленной приемки или выпуска программного модуля, пакета или продукции.
Признаки (показатели) (features)	Признаки, определяющие свойства программной продукции, которые могут быть отнесены к характеристикам качества. Примечание - Примеры признаков включают длину маршрута, модульность, структуру программы и комментарии.
Программная продукция (software product)	Программный объект, предназначенный для поставки пользователю.
Программно-аппаратные средства (firmware)	Технические средства, содержащие компьютерную программу и данные, которые не могут изменяться средствами пользователя. Компьютерная программа и данные, входящие в программно-аппаратные средства, классифицируются как программное обеспечение; схемы, содержащие компьютерную программу и данные, классифицируются как технические средства.
Программное обеспечение (software)	Программы, процедуры, правила и любая соответствующая документация, относящиеся к работе вычислительной системы.
Ранжирование (рейтинг) (rating)	Действие по отнесению измеренного значения к соответствующему уровню ранжирования. Используется для определения уровня ранжирования программного обеспечения по конкретной характеристике качества.
Уровень качества функционирования (level of performance)	Степень, в которой удовлетворяются потребности, представленные конкретным набором значений для характеристик качества.
Уровень ранжирования (rating level)	Диапазон значений в масштабе, позволяющем классифицировать (ранжировать) программное обеспечение в соответствии с установленными или предполагаемыми потребностями.

	Соответствующие уровни ранжирования могут быть связаны с различными представлениями о качестве, то есть для пользователей, руководителей или разработчиков. Данные уровни называются уровнями ранжирования. Примечание - Данные уровни ранжирования отличны от "классов", определенных ИСО 8402.
Характеристики качества программного обеспечения (software quality characteristics)	Набор свойств (атрибутов) программной продукции, по которым ее качество описывается и оценивается. Характеристики качества программного обеспечения могут быть уточнены на множестве уровней комплексных показателей (подхарактеристик).

4. ХАРАКТЕРИСТИКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Мобильность (Portability)	Набор атрибутов, относящихся к способности программного обеспечения быть перенесенным из одного окружения в другое. Примечание - Окружающая обстановка может включать организационное, техническое или программное окружение.
Надежность (Reliability)	Набор атрибутов, относящихся к способности программного обеспечения сохранять свой уровень качества функционирования при установленных условиях за установленный период времени. Примечания 1. Износ или старение программного обеспечения не происходит. Ограничения надежности проявляются из-за ошибок в требованиях, проекте и реализации. Отказы из-за этих ошибок зависят от способа использования программного обеспечения и ранее выбранных версий программ. 2. В определении ИСО 8402 "надежность" - "способность элемента выполнять требуемую функцию". В настоящем стандарте функциональная .возможность является только одной из характеристик качества программного обеспечения. Поэтому определение надежности расширено до "сохранения своего уровня качества функционирования" вместо "выполнения требуемой функции".
Практичность (Usability)	Набор атрибутов, относящихся к объему работ, требуемых для использования и индивидуальной оценки такого использования определенным или предполагаемым кругом пользователей.

	Примечания 1. "Пользователи" могут интерпретироваться как большинство непосредственных пользователей интерактивного программного обеспечения. Круг пользователей может включать операторов, конечных пользо-вателей и косвенных пользователей, на которых влияет данное программное обеспечение или которые зависят от его использования. Практичность должна рассматриваться во всем разнообразии условий эксплуатации пользователем, которые могут влиять на программное обеспечение, включая подготовку к использованию и оценку результатов. 2. Практичность, определенная в данном стандарте как конкретный набор атрибутов программной про-дукции, отличается от определения с точки зрения эргономики, где рассматриваются как составные части практичности другие характеристики, такие как эффективность и неэффективность.
Сопровождаемость (Maintainability)	Набор атрибутов, относящихся к объему работ, требуемых для проведения конкретных изменений (модификаций). Примечание - Изменение может включать исправления, усовершенствования или адаптацию программного обеспечения к изменениям в окружающей обстановке, требованиях и условиях функционирования.
Функциональные возможности (Functionality)	Набор атрибутов, относящихся к сути набора функций и их конкретным свойствам. Функциями являются те, которые реализуют установленные или предполагаемые потребности: Примечания 1. Данный набор атрибутов характеризует то, что программное обеспечение выполняет для удовлетворения потребностей, тогда как другие наборы, главным образом, характеризуют, когда и как это выполняется. 2. В данной характеристике для установленных и предполагаемых потребностей учитывают примечание к определению качества.
Эффективность (Efficiences)	Набор атрибутов, относящихся к соотношению между уровнем качества функционирования программного обеспечения и объемом используемых ресурсов при установленных условиях. Примечание - Ресурсы могут включать другие программные продукты, технические

	средства, материалы (например бумага для печати, гибкие диски) и услуги эксплуатирующего, сопровождающего или обслуживающего персонала.

А.2.5.2 Изменяемость (Changeability)
А.2.5.3 Устойчивость (Stability)
А.2.5.4 Тестируемость (Testability)
А.2.6 Мобильность (Portability)
А.2.6.1 Адаптируемость (Adaptability)
А.2.6.2 Простота внедрения (Installability)
А.2.6.3 Соответствие (Conformance)
А.2.6.4 Взаимозаменяемость (Replaceabilily)

Примечания:
1. Взаимозаменяемость используется вместо совместимости для того, чтобы избежать возможной путаницы со способностью к взаимодействию.
2. Взаимозаменяемость с конкретным программным средством не предполагает, что данное средство заменимо рассматриваемым программным средством.
3. Взаимозаменяемость может включать атрибуты простоты внедрения и адаптируемости. Понятие было введено в качестве отдельной подхарактеристики из-за его важности.

2.3. Выводы

Изучая раздел “Атрибуты качества продукта и их характеристики”, мы пришли к выводам о том, что:

- Атрибуты качества программного продукта описаны в ГОСТ Р ИСО/МЭК 9126-93 и состоят из шести характеристик, в том числе: «Мобильность, Надежность, Практичность, Сопровождаемость, Функциональные возможности и Эффективность».
- Каждая из характеристик атрибутов качества может быть оценена в виде составляющих ее подхарактеристик, в том числе: «Мобильность (Адаптируемость, Простота внедрения, Соответствие, Взаимозаменяемость), Надежность (Стабильность, Устойчивость к ошибке, Восстанавливаемость), Практичность (Понятность, Обучаемость, Простота использования), Сопровождаемость (Анализируемость, Изменяемость, Устойчивость, Тестируемость), Функциональные возможности (Пригодность, Правильность, Способность к взаимодействию, Согласованность и Защищенность) и Эффективность (Характер изменения во времени, Характер изменения ресурсов)».
- Стандарт ГОСТ Р ИСО/МЭК 9126-93 уточняет такие определения, как: «Измерение, Качество, Качество программного обеспечения, Критерий оценки качества программного обеспечения, Метрика качества программного обеспечения, Оценка, Признаки (показатели), Программная продукция, Программно-аппаратные средства, Программное обеспечение, Ранжирование (рейтинг), Уровень качества функционирования, Уровень ранжирования и Характеристики качества программного обеспечения», применительно для использования значений характеристик качества программного обеспечения посредством выполнения измерений.

2.4. Вопросы и задания для самостоятельной работы студента по теме «Атрибуты качества продукта и их характеристики»

1) Дайте определение понятию «Атрибуты качества» программного продукта.
2) Перечислите основные атрибуты качества ПО.
3) Дайте определение терминам: «Мобильность, Надежность, Практичность, Сопровождаемость, Функциональные возможности и Эффективность».
4) Выберите разумные измеряемые характеристики качества для проекта разработки биллинговой системы оператора сотовой связи.
5) Задайте предельно допустимые значения атрибутов качества для конкретного проекта разработки ПО и опишите

необходимые корректирующие действия в случае выхода атрибутов качества за допустимый диапазон значений.

Литература по теме «Атрибуты качества продукта и их характеристики»

[1] ГОСТ Р ИСО 9000-2001.

[2] Брауде Э. Технология разработки программного обеспечения. СПб, Питер, 2004.

[3] ГОСТ Р ИСО / МЭК 9126-93.

Тема 3. Управление качеством процесса разработки

Реальности развития информационных технологий таковы, что разработчики сервисов Информационных технологий самостоятельно руководствуются отечественными или зарубежными стандартами, призванными регулировать современные подходы в разработке, внедрении и сопровождении проектов ИТ.

3.1. Обзор практики использования стандартов ИТ в Российской Федерации

3.1.1. Основные задачи стандартизации в области ИТ

В соответствии с Федеральным законом Российской Федерации от 27 декабря 2002 г. N 184-ФЗ «О техническом регулировании» применение стандартизации осуществляется в целях:

- повышения уровня безопасности жизни или здоровья граждан, имущества физических или юридических лиц, государственного или муниципального имущества, экологической безопасности, безопасности жизни или здоровья животных и растений;
- повышения уровня безопасности объектов с учетом риска возникновения чрезвычайных ситуаций природного и техногенного характера;
- обеспечения научно-технического прогресса;
- повышения конкурентоспособности продукции, работ, услуг;
- рационального использования ресурсов;
- технической и информационной совместимости;
- сопоставимости результатов исследований (испытаний) и измерений, технических и экономико-статистических данных;
- взаимозаменяемости продукции.

Кроме перечисленных положений, Закон определяет стандартизацию, как деятельность по установлению правил и характеристик в целях их добровольного многократного использования, направленная на достижение упорядоченности в сферах производства и обращения продукции, работ или услуг.

Применение информационно-коммуникационных технологий (ИКТ) может причинить непосредственный вред жизни и здоровью в тех случаях, когда применение ИКТ используется для управления опасным технологическим оборудованием, системами вооружений, транспортными средствами и т.п. Использование ИКТ для

проектирования и решения подобных задач, степень надежности и функционального соответствия ИТ-составляющей оказывает значительное влияние на безопасность технической или организационной системы в целом. Однако определение требований к подобным системам невозможно без четкой привязки к предметной области (например, атомной, химической, транспортной отрасли).

В тех случаях, когда информационная система непосредственно связана с обеспечением защиты жизни или здоровья граждан, имущества физических или юридических лиц, государственного или муниципального имущества, законом предусмотрено применение технических регламентов, а не стандартов.

Вопросы безопасности применения информационных технологий и понятия безопасности применения и информационной безопасности (защиты информационных систем от несанкционированного доступа) представляют собой различные разделы безопасности, причем информационная безопасность скорее относится собственно к внутренним функциям ИКТ.

Основным ресурсом, необходимым для развития информационных технологий, является человеческий. Рационализация использования человеческих ресурсов возможна за счет организации и управления процессами при производстве и эксплуатации ИТ-продукции, работ или услуг.

Зависимость результатов исследований (испытаний) и измерений, технических и экономико-статистических данных от степени информационной и технологической совместимости ИТ-продукции обязывает соблюдения принятых на нем промышленных стандартов – существующих как де-факто, так и де-юре.

Таким образом, можно выделить два основных направления деятельности по стандартизации в области информационных технологий:

- обеспечение технической и информационной совместимости информационных систем и продуктов и, в первую очередь, стандартизация протоколов информационного обмена.
- обеспечение упорядоченности в сферах производства и обращения продукции и, в первую очередь, унификация и определение понятийного аппарата ИТ. В

Формулируя задачи в области стандартизации, необходимо учитывать, что «Законом о техническом регулировании» особо подчеркивается добровольный принцип применения стандартов. Это накладывает на государственные органы технического регулирования особую ответственность по выбору степени регламентации деятельности в области информационных технологий.

Международное ИТ-сообщество продемонстрировало достаточно высокую способность к самоупорядочиванию и добровольному принятию общих для все отрасли стандартов. В то же время во многих

отношениях стандартизация информационных технологий происходит стихийно, что приводит к непроизводительному расходованию интеллектуальных ресурсов, навязыванию крупными разработчиками информационных проприетарных закрытых стандартов и другим проблемам.

Особую роль в обеспечении научно-технического прогресса могут сыграть стандарты, связанные с локализацией (национализацией) информационных технологий, имплементацией международных стандартов для решения специфических задач, характерных для российских традиций информационного оборота, связанных с обеспечением требований национального законодательства и т.п.

3.1.2. Текущее состояние государственной стандартизации в области ИТ

Согласно Общероссийскому классификатору стандартов (ОКС) в 35 группе («Информационные технологии. Конторские машины») насчитывается около 500 формально действующих стандартов, входящих в следующие подгурппы:

35.020 Информационные технологии (ИТ) в целом

35.040 Наборы знаков и кодирование информации

35.060 Языки, используемые в информационных технологиях

35.080 Документация на разработку программного обеспечения и системная документация

35.100 Взаимосвязь открытых систем

35.110 Организация сети

35.140 Машинная графика

35.160 Микропроцессорные системы

35.180 Информационно-технологические терминалы и другие периферийные устройства

35.200 Интерфейсы и межсоединительные устройства

35.220 Запоминающие устройства

35.240 Применение информационных технологий

35.260 Конторские машины

Преобладание стандартов, относящихся к аппаратному обеспечению информационных технологий (микропроцессорные системы, интерфейсы и соединительные устройства и т.п.), и применяемая терминология устарели и не применяются на практике. Распределение стандартов группы ИТ по системам ГОСТ, ориентированным на различные отрасли деятельности неравномерно.

То же касается нормативов, регулирующих общие вопросы:

- ГОСТ 6 – Унифицированная система документации. В частности, к этой системе отнесены стандарты,

регламентирующие порядок придания юридической силы документам на машинном носителе и машинограмме, создаваемым средствами вычислительной техники, а также стандарты по информационному обмену управленческими данными.

- ГОСТ 7 - Система стандартов по информации, библиотечному и издательскому делу. В эту группу стандартов, в частности, включены стандарты обмена библиографической и архивной информацией.
- ГОСТ 2 (ЕСКД) – Единая система конструкторской документации, определяющая, в частности, правила выполнения схем и чертежей в области аппаратного обеспечения ИТ.

Стандарты для информационных технологий:

- ГОСТ 19 – Единая система программной документации.
- ГОСТ 34 – Стандарты области информационных технологий. Содержит основную массу стандартов ИТ, в первую очередь стандарты, относящиеся к взаимодействию открытых систем и к автоматизированным системам.
- ГОСТ 24 - единая система стандартов автоматизированных систем управления, распространяющихся на АСУ, АСУП, АСУ ТП и другие организационно-экономические системы
- комплекс стандартов (система 23501); распространяющихся на системы автоматизированного проектирования;
- четвертая группа 14-й системы стандартов, распространяющаяся на автоматизированные системы технологической подготовки производства.

Практика применения стандартов на АСУ, САПР, АСУ ТП, АСТПП показала, что в них применяется одинаковый понятийный аппарат, имеется много общих объектов стандартизации, однако требования стандартов не согласованы между собой, имеются различия по составу и содержанию работ, различия по обозначению, составу, содержанию и оформлению документов и пр.. Ввиду отсутствия единой технической политики в области создания АС многообразие стандартов не обеспечивало широкой совместимости АС при их взаимодействии, не позволяло тиражировать системы, тормозило развитие перспективных направлений использования средств вычислительной техники.

Единый комплекс стандартов и руководящих документов, относящихся к 34 системе, должен распространяться на автоматизированные системы различного назначения: АСНИ, САПР, ОАСУ, АСУП, АСУТП, АСУГПС, АСК, АСТПП, включая их интеграцию. Однако, часть стандартов 24, 14 и 23 системы продолжают действовать просто в силу того, что соответствующих им стандартов в серии 34 не существует,

Реально действуют:

- стандарты на документирование программных продуктов, в первую очередь входящие в ЕСПД, что связано в основном с требованиями госорганов, для которых разрабатывается та или иная система;
- стандарты на алгоритмы шифрования и ЭЦП, что связано с обязательным лицензированием деятельности в этой области.

3.1.3. Использование международных стандартов

В соответствии со статьей 12 закона «О техническом регулировании» предусматривается принцип применения международного стандарта как основы разработки национального стандарта, за исключением случаев, если такое применение признано невозможным.

К настоящему времени на международном уровне сформировалась мощная кооперация организаций, разрабатывающих стандарты в области информационных технологий, среди которых, в первую очередь, следует назвать Международную организацию по стандартизации - ИСО (International Organization for Standardization - ISO), Международную электротехническую комиссию - МЭК (International Electrotechnical Commission - IEC) и Международный союз электросвязи - МСЭ (International Telecommunication Union - ITU); его сектор по телекоммуникациям МСЭ-Т является с 1993 года правопреемником МККТТ.

В 1987 году ИСО и МЭК объединили свою деятельность по стандартизации в области информатизации, создав Совместный технический комитет №1 - СТК1 ИСО/МЭК "Информационные технологии" (Joint Technical Committee N1 - ISO/IEC JTC1 "Information Technology"), основной задачей которого является разработка базовых стандартов информационных технологий вне зависимости от их конкретных приложений, членом которого (наряду с 54-мя другими государствами) является Российская Федерация. В структуре СТК1 функционирует свыше 20 подкомитетов и рабочих групп, охватывающих в своей деятельности практически весь спектр стандартизации в области информационных технологий и осуществляющих разработку стандартов по следующим основным направлениям:

- Наборы символов и кодирование информации;
- Телекоммуникация и обмен информацией;
- Программная инженерия;
- Языки программирования;
- Машинная графика и обработка изображений;
- Взаимосвязь оборудования информационных технологий;
- Методы защиты информации;

- Конторское оборудование;
- Кодирование аудио, видео, мультимедиа и гипермедиа-информации;
- Методы автоматической идентификации, кодирования и фиксации данных;
- Управление и обмен данными;
- Языки описания и обработки документов;
- Интерфейсы пользователя;
- Методы обучения.

Помимо ИСО, МЭК и МСЭ разработкой стандартов в области информационных технологий и, в частности, - в области открытых систем, занимается ряд авторитетных международных, региональных, национальных и специализированных организаций, консорциумов и групп, например такие, как:

- Общество Интернет (Internet Society);
- Европейский комитет стандартизации - СЕН (Comite European de Normalization - CEN) и Европейский комитет стандартизации в области электротехники - СЕНЭЛЕК (Comite European de Normalization Electrotechnique - CENELEC);
- Европейская ассоциация производителей компьютеров - ЕКМА (European Computer Manufacturers Association - ECMA);
- Европейские рабочие группы по открытым системам - ЕВОС (European Workshops on Open Systems - EWOS);
- Европейский институт стандартизации телекоммуникаций (European Telecommunications Standards Institute);
- Институт инженеров по электротехнике и электронике (Institute of Electrical and Electronic Engineers - IEEE),
- X/Open, организованная поставщиками компьютерной техники;
- Фонд открытого программного обеспечения (Open Software Foundation - OSF),;
- Группа объектного управления (Object Management Group - OMG);
- Форум управления сетями (Network Management Forum - NMF) и др.

В настоящее время действующий в России комплекс стандартов в области информационных технологий обеспечивает прямое введение только приблизиельно 25% от общего числа стандартов СТК1.

Прогрессирующее отставание России по числу действующих стандартов ИТ связано с недостаточным финансированием разработки соответствующих стандартов, а также с несовершенством организационно-методических форм принятия международных стандартов в качестве государственных.

Характерной особенностью современных информационных технологий является чрезвычайно высокие темпы их развития, в то время, как принятие официальных стандартов области информационных технологий значительно отстает от реально сложившейся системы стандартов.

Значительная часть реально применяемых в ИТ протоколов и спецификаций, является продуктом деятельности различных сообществ - добровольного объединения граждан различных стран. Даже основополагающие стандарты Интернета, публикуемые консорциумом W3C, издаются в формате RFC (request for comments), т.е. «приглашений к обсуждению».

Неофициальный характер таких стандартов, с одной стороны, обеспечивает их высокую гибкость и способность противостоять моральному старению, с другой стороны, осложняет их принятие на государственном уровне.

3.1.4. Вопросы, связанные с применением стандартов ИТ

3.1.4.1. Терминология

Одной из важнейших проблем при подготовке договорной, проектной и рабочей документации на информационные системы является отсутствие стандартизированной терминологии по современным технологиям. В силу сложившейся практики ИТ-специалисты используют главным образом английские термины, зачастую не имеющие русскоязычных аналогов, неоднозначно транскрибируемые или неудачно ложащиеся на морфологию русского языка. Все это привносит в документацию нежелательный сленговый окрас, затрудняет ее восприятие. Кроме того, толкование многих применяемых терминов неоднозначно и, напротив, одно и то же понятие часто обозначается разными терминами даже в пределах одного документа.

В связи с этим имеется насущная необходимость в разработке ряда глоссариев и словарей, которые, с одной стороны, дали бы адекватные российские синонимы для применяемых в международной практике терминов, с другой стороны, предоставили бы их единое толкование для юридических и технических целей. Среди необходимых словарей можно выделить:

- терминологию области современных локальных сетей и сетевых архитектур;
- терминологию области глобальных сетей, в первую очередь интернета;
- терминологию области жизненного цикла программных средств и информационных систем;
- терминологию области технологии знаний;
- терминологию области управленческих систем и систем

поддержки принятия решений;
- терминологию графического интерфейса пользователя.

3.1.5. Типология информационных систем

Действующая система стандартов в области ИТ ориентирована на автоматизированные системы, главным образом производственного класса (САПР, СУТП и т.п.). Развитие информационной технологий приводит к проникновению информационно-компьютерных технологий в самые различные области человеческой деятельности, включая быт, досуг, масс-медиа и т.п. непромышленные сферы. Это вынуждает разработчиков и пользователей прибегать к расширительному толкованию понятия «автоматизированные системы», относя его к системам управления производственной и коммерческой деятельности, и к телекоммуникационным и информационно-справочным системам, к интернет-сайтам и порталам и т.д. В то же время принципы функционирования всех не могут быть сведены к простой автоматизации некоторых процессов, более того, многие из перечисленных систем не являются «автоматизирующими» в строгом смысле этого слова.

В связи с этим возникает потребность расширения понятийной сферы системы стандартов информационных технологий. Несмотря на отсутствие стандартов данной группы, введение четкой классификации и определение минимального набора функциональных задач и показателей назначения различных типов и классов информационных систем, в первую очередь систем обеспечения управленческой деятельности, таких, как ERP, CRM и т.п. с

3.1.6. Процедура разработки

На настоящий момент в России принят стандарт ИСО 12207, регламентирующий жизненный цикл программных продуктов, однако многими специалистами отмечается сложность его применения на практике в силу чрезмерной универсальности. Отсутствует разработка ряда стандартов или руководящих документов, увязанных со стандартами ИСО/МЭК 12207 и 9000, и регламентирующий процедуры разработки, контроля и документирования для различных типов информационных систем:

- анализ информационных систем, построение моделей бизнес-процессов;
- разработка пилотных версий и прототипов;
- разработка комплексных решений на базе готовых программных продуктов;
- адаптация ПО, включая локализацию и интернационализацию;
- альфа- и бета-тестирование;
- выпуск обновлений, совершенствование версий (апгрейд);
- устранение ошибок и сбоев путем выпуска «заплаток» (патчей) и

т.п.

3.1.7. Нефункциональные показатели информационных систем

В настоящее время ни на уровне официальных стандартов, ни на уровне практического применения в отрасли не существует методик объективной оценки таких важнейших нефункциональных показателей информационных систем, как:

- производительность;
- быстродействие;
- объемы обрабатываемой информации;
- надежность;
- регламент функционирования;
- интенсивность обслуживания;
- эргономические показатели и удобство использования.

При формулировке требований к системам эти параметры либо не регламентируются, либо, что равнозначно, устанавливаются эмпирически, «на уровне стандартных представлений о возможностях систем подобного рода».

3.1.8. Документирование информационных систем

Состояние стандартов, относящихся к документированию информационных систем:

- серия стандартов ЕСПД (ГОСТ 19) заметно устарела, и лояльность разработчиков к ней связана исключительно с заложенной в идеологию стандарта свободой его применения. Так, разработчик документации имеет право на свое усмотрение вводить и исключать разделы документации, самостоятельно определять состав необходимых документов. При этом предложенная в стандарте структура и состав документов уже не соответствует реалиям современных программных продуктов, будучи рассчитана на пакетные процедуры обработки данных. В результате использование стандартов ЕСПД сводится в основном к оформлению рамок и надписей на титульных листах.
- стандарты по документированию серии 34 значительно более информативны, однако избегаются разработчиками в силу отмечавшегося выше перекоса в сторону производственной автоматизации, обилия разделов и документов, связанных с аппаратным обеспечением, и жестко фиксированной процедурой разработки систем, плохо ложащейся на современные технологии интенсивного проектирования.

- нормы ИСО 12207 вызывают затруднения в силу их универсализма и размытости, а также сложности толкования.

3.1.9. Документирование исходного кода

В связи с общим движением мирового сообщество к осознанию необходимости раскрытия исходных кодов программных продуктов возникает задача обеспечения его читаемости.

В Российской Федерации отсутствуют рекомендации, регламентирующие следующие аспекты создания текстов программ:

- использование единых принципов именования программных объектов;
- минимальный уровень комментирования текста;
- порядок описания программных интерфейсов приложений;
- обеспечение автоматического документирования программ.

3.1.10. Форматы информационного обмена

В современной системе ГОСТ/ГОСТ Р формально действует около сотни стандартов на форматы и протоколы информационного обмена физического и канального уровня, таких, как «толстый Ethernet» или X.25. На их разработку и внедрение были затрачены большие усилия, в то время как сами технологии морально устарели.

Сообщество разработчиков ИТ в Российской федерации успешно справляется с саморегулированием в области использования сетевых протоколов и форматов обмена универсального назначения (Например: самоорганизация по применению протоколов обмена данными в Интернет, проектирование и развертывание Структурированных Кабельных Сетей (СКС) и др.).

Несмотря на то, что в системе Госстандарта имеется ряд документов, регламентирующих форматы информационного обмена, большинство из них морально устарело, относится к представлению данных на специфических носителях (перфокарты, штрих-коды, магнитная лента) или в редуцированном, неудобном для восприятия и применения формате, порожденном ограниченными возможностями ЭВМ прошлых лет.

Отсутствуют стандарты документарных форматов:

- обмена организационно-распорядительной и финансовой информацией;
- представления отчетной и статистической информацией с государственными органами;
- обмена метаданными – справочной, библиографической, архивной и иной информацией, необходимой для эффективного поиска, сортировки, автоматизации обработки данных.

Некоторые результаты, все же достигнуты. Например, принят стандарт ИСО 2709 «Система стандартов по информации, библиотечному и издательскому делу. Формат для обмена информацией. Структура записи», однако он носит ярко выраженный библиографический характер и не покрывает всех задач представления информации о различных типах объектов.

Международный стандарта на структуру метаданных содержит ряд рекомендаций международных сообществ, таких, как:

- Dublin Core Metadata Initiative, http://www.dublincore.org
- PRISM (Publishing Requirements for Industry Standard Metadata, http://www.prismstandard.org/)
- RSS (RDF Site Summary, http://web.resource.org/rss/1.0/)
- рекомендаций RFC 2798 группы IETF

К методологическим проблемам можно отнести вынужденные попытки применениям общепринятых протоколов и спецификаций при разработке частных стандартов (например, стандартов предприятий по разработке или по применению форматов информационного обмена). Многие из этих протоколов и спецификаций являются продуктом деятельности независимых организаций и сообществ. В тексте нормативных документов не вполне корректно использовать ссылки эти организации в качестве первоисточника, как с точки зрения неопределенности их юридического статуса, так и из соображений нежелательности зависимости системы поддержки тех или иных стандартов неформальным сообществом.

В качестве примера, отсутствуют нормативные положения такими технологиями, как: HTTP, HTML, XML, XHTML, XLS CSS, SSL и ряд других.

3.1.11. Кодировки символов

Предложенная кодировка в действующем ГОСТ 19768-93 практически никем не используется. На сайте Госстандарта, например, в соответствии с требованиями ГОСТ 19768-93, информация должна быть представлена в кодировке WIN-1251.

Данная проблема не создает сложностей для разработчиков и пользователей, но затрудняет использование стандартов при разработке ПО.

В связи с этим в качестве промежуточной меры предлагается выбрать и признать одну из реально используемых кодировок, не являющихся в то же время собственностью определенного разработчика.

Дальнейшую деятельность в области регламентации кодировок предлагается сосредоточить на применении различных версий формата UNICODE.

3.2. Обеспеченность организаций и предприятий РФ средствами и ресурсами информационных технологий (укрупненные показатели)

3.2.1. Используемые программные платформы.

Самыми распространенными аппаратными платформами являются платформы семейства Windows:

- Windows 2000,
- Windows XP.

Они используются практически в каждой организации. На втором месте по популярности применения являются платформы Linux и FreeBCD. Распространенность морально устаревшей платформы MS DOS, учитывать.

3.2.2. Используемые аппаратные платформы.

Здесь безусловное лидерство удерживает программная платформа x86 (Intel и т.д.), которая используется во всех организациях исследования.

Далее следует платформа Sun. Платформы Mac, Alfa и Power PC, встречаются редко.

3.2.3. Ограничения на использование других платформ.

Ограничений на использование других платформ не существует. Однако если они и есть, то, как правило, связаны с соображениями экономии бюджетных средств и технологическими особенностями организаций. При этом на выбор платформы не влияет ни один нормативный документ (федеральные или местные законы, подзаконные акты и т.д.).

3.2.4. Основные типовые программные конфигурации используемого пользовательского оборудования.

Наиболее часто в качестве операционной системы используется операционная система Windows, при этом самой распространенной конфигурацией выступает Windows XP/2000. Все большую популярность приобретает Windows Vista.

Самой популярной офисной программой выступает Microsoft Office, причем выделить основную используемую типовую конфигурацию данного вида ПО достаточно сложно. Обычно используется Microsoft Office 2003.

Что касается баз данных, то здесь стоит отметить, что наблюдается значительный разброс в предпочтениях: в основном используются программы Access, Firebird, Lotus Notes, FoxPro, MS SQL, Oracle, Interbase, а некоторые организации предпочитают разрабатывать собственные базы данных.

Среди справочных и, в частности, правовых систем, наибольшей популярностью пользуются следующие программные конфигурации – Консультант+, Гарант, Кодекс.

Самыми распространенными бухгалтерскими программами являются программные конфигурации Парус и 1С.

Работа в сети Интернет осуществляется с помощью программы Internet Explorer. При этом в качестве почтовых программ выступают Outlook Express, Bat или программы Lotus Notes.

Ситуация в области применяемых систем электронного оборота (Work-flow) аналогична ситуации относительно используемых баз данных. Наиболее часто используемыми являются программы на базе Lotus Notes, KIS 2000 и Дело. Иногда организации создают систему документооборота собственными силами, однако зачастую такой программы просто нет.

Основными инструментами борьбы с вирусами являются Kaspersky Anti-Virus, DrWeb или Norton Anti-Virus

3.2.5. Используемое ПО.

На первом месте по использованию находится серийное (тиражируемое, коробочное) ПО. Затем следует ПО, разработанное собственными силами. Наконец, в последнюю очередь используется заказное ПО.

3.2.6. Сервисы, обеспечиваемые серверами.

Большая часть организаций использует серверы в качестве файловых хранилищ и систем баз данных, а также задействованы в системе документооборота и связаны с различными приложениями. Сетевые серверы, обеспечивают сетевые сервисы и сервисы Интернета.

3.2.7. Учет числа клиентов и пользователей.

В большинстве организаций, ведется учет числа клиентов и пользователей.

3.2.8. Примерная структура используемого оборудования по сроку использования.

Сроки использования оборудования в большей части организаций варьируются от 2-3 лет до 5 лет. Если выразить структуру в процентном соотношении, то общая ситуация будет следующей: 50% оборудования используется не более 2 лет, 30% - от 2 до 4 лет, 20% - не более 4 лет. Оборудование модернизируется каждые 2-3 года. Другим словами, идет процесс плавного обновления оборудования. Тем не менее, встречаются и такие организации, часть оборудования которых была куплена 7-10 лет назад.

3.3. Характеристики управления процесса внедрения и использования ИТ

3.3.1. Обеспеченность организаций оборудованием, а также сотрудниками, занимающимися обслуживанием ИТ-сервисов.

Обеспеченность сотрудников организаций рабочими станциями значительно отличается в разных регионах и муниципальных образованиях. Показатели составляют от 50% до 100% (полной обеспеченности).

Усредненные показатели таковы:

- Обеспеченность рабочих станций серверами демонстрирует разброс от 15 – 20 до 60 станций на 1 сервер. При этом средние показатели это 30 – 35 станций.
- Кадровая обеспеченность подразделений организаций, ответственных за обслуживание ИТ-сервисов, в расчете на количество рабочих станций также различается существенным образом (от 15 до 60 рабочих станций на 1 специалиста). Средний показатель составляет 1 сотрудник на 30 – 35 рабочих станций.

3.3.2. Электронный архив.

Практически все организации за некоторым исключением стараются использовать возможности электронной архивации данных. Наиболее распространены два способа доступа к электронным документам: доступ осуществляется либо напрямую (то есть он является свободным), либо с помощью системы электронного доступа, в основе которой лежит авторизация через пароль и логин. Иногда права доступа к документам и информации, хранящейся в базах данных и архивах, регламентируются внутренними распорядительными документами и инструкциями.

3.3.3. Форматы хранения документов.

В основном для хранения документов используются 5 форматов: *.doc,*.txt,*.rtf, *.html,*.xls. Наиболее популярными являются форматы *.doc и *.txt.

3.3.4. Предполагаемое время хранения документов.

В большинстве случаев планируется, что информация храниться в электронном архиве более 10 лет и даже бессрочно. Некоторые организации предполагают сохранять документы в течение 5 лет. В организациях не существует четко регламентированной процедуры, определяющей период хранения документов в электронном архиве.

3.3.5. Уровень грамотности сотрудников и уровень грамотности пользователей).

Оценки уровня грамотности сотрудников и пользователей сильно различаются. В целом же, примерно 90% сотрудников их организаций умеют работать на компьютере, 80% имеют навыки работы в Интернете, однако лишь 60% способны адаптироваться к смене аппаратной или программной платформы.

3.4. Нынешняя ситуация в области использования различных схем лицензирования ПО и оценки перспектив использования ПО со свободной лицензией.

3.4.1. Используемые схемы лицензирования.

Как правило, лицензирование программного обеспечения осуществляется путем закупки отдельных экземпляров легального программного обеспечения с последующей установкой на большое число компьютеров. Кроме того, лицензия на программное обеспечение может быть получена вместе с приобретением компьютера (предустановленное программное обеспечение). Другой же способ – массовая закупка легального программного обеспечения на все компьютеры (1 пользователь – 1 лицензия) – используется достаточно редко.

3.4.2. Используемое ПО со свободной лицензией.

Большинство организаций не использует программное обеспечение со свободной лицензией. Если же такие организации и находятся, то основными используемыми программами со свободной лицензией являются Linux, Appache и FreeBSD. Доля пиратского ПО составляет 80% - 90%.Информированность сотрудников о свободном ПО. Сотрудники организации, в том числе топ-менеджеры осведомлены о существовании лицензионного и свободно-лицензированного программного обеспечения, о механизмах использования и о преимуществах и недостатках.

К свободно распространяемому программному обеспечению обычно относят операционные системы, WEB-серверы, работу в сети Интернет и офисное ПО.

К препятствиям по внедрению свободного ПО обычно относят три группы сложностей, технологические, психологические и финансовые.

- К технологическим сложностям относят такие проблемы, как слабая техническая поддержка свободного ПО и необходимость создания собственной сети поддержки, конвертация документов, длительность внедрения свободного ПО.
- Психологические сложности – это, прежде всего, переориентация всех сотрудников на использование нового ПО, изменение привычек, низкая готовность сотрудников.

- К финансовым трудностям относятся затраты на повышение квалификации, на обучение и информирование сотрудников.

3.4.3. Затраты на полную легализацию используемого ПО.

Полная легализация ПО может составлять суммы, 20 тыс. долларов, 50 -70 тыс. долларов и 140 тыс. долларов. для каждой из организаций.

3.5. ГОСТ Р ИСО/МЭК 12207-99. Описание стандарта

Стандарт называется «Информационная технология. Процессы жизненного цикла программных средств» (Information technology. Software life cycle processes).

Стандарт ISO 12207 принят в России, как ГОСТ Р ИСО/МЭК 12207-99 [18]. Он разработан Всероссийским научно-исследовательским институтом стандартизации (ВНИИстандарт) Госстандарта России; принят и введен в действие постановлением Госстандарта России от 23 декабря 1999 г. № 675-ст. Стандарт устанавливает, используя четко определенную терминологию, общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, работы и задачи, которые используются: при приобретении системы, содержащей программные средства, или отдельно поставляемого программного продукта; при оказании программной услуги, а также при поставке, разработке, эксплуатации и сопровождении программных продуктов. Определения процессов, работ и задач носят описательный характер, указывается, что должно делаться, но не дается рекомендаций как это делать. Детальное решение вопросов реализации стандарта возлагается на организацию, использующую стандарт. На практике стандарт используется следующим образом: из общего перечня процессов, работ и задач выбираются те, которые адекватно соответствуют специфике конкретной организации. Выбранные элементы уточняются до уровня документированных процедур, и таким образом, регламентируются процессы, установленные в организации.

Данный стандарт посвящен практически всем вопросам и понятиям, связанным с разработкой программной продукции и его применение, хотя и не носит обязательного характера, но неформально является руководящим документом.

Изучение стандарта ГОСТ Р ИСО/МЭК 12207-99 в рамках данного курса является обязательным. Работа по изучению ГОСТ Р ИСО/МЭК 12207-99 выполняется самостоятельно.

Публикуемые аннотации к стандарту ГОСТ Р ИСО/МЭК 12207-99, содержат справку по библиографии[12]:

Обозначение	ГОСТ Р ИСО/МЭК 12207-99
Заглавие на русском языке	Информационная технология. Процессы жизненного цикла программных средств
Заглавие на английском языке	Information technology. Software life cycle processes
Дата введения в	01.07.2000

[12] http://www.standards.ru/document/4159467.aspx

действие	
Дата огр. срока действия	
ОКС	35.080
Код ОКП	
Код КГС	П85
Код ОКСТУ	5001
Индекс рубрикатора ГРНТИ	
Аннотация (область применения)	Настоящий стандарт применяется при приобретении систем, программных продуктов и оказании соответствующих услуг; а также при поставке, разработке, эксплуатации и сопровождении программных продуктов и программных компонентов программно-аппаратных средств, как в самой организации, так и вне ее. Стандарт содержит также те аспекты описания системы, которые необходимы для обеспечения понимания сути программных продуктов и услуг
Ключевые слова	обработка данных;оборудование обработки данных;вычислительные машины;программные средства вычислительных машин;жизненный цикл
Термины и определения	Раздел стандарта
Наличие терминов РОСТЕРМ	
Вид стандарта	Основополагающие стандарты
Вид требований	
Дескрипторы (английский язык)	
Обозначение заменяемого(ых)	
Обозначение заменяющего	
Обозначение заменяемого в части	
Обозначение заменяющего в части	
Примечание	
Гармонизирован	

с:	
Аутентичный текст с ISO	ISO/IEC 12207:1995
Аутентичный текст с IEC	
Аутентичный текст с ГОСТ	
Аутентичный текст с прочими	
Содержит требования: ISO	
Содержит требования: IEC	
Содержит требования: СЭВ	
Содержит требования: ГОСТ	
Содержит требования: прочими	
Нормативные ссылки на: ISO	ISO/IEC 2382-1:1993;ISO/IEC 2382-20:1990;ISO 8402:1994
Нормативные ссылки на: IEC	
Нормативные ссылки на: ГОСТ	ГОСТ Р ИСО 9001-2001;ГОСТ Р ИСО/МЭК 9126-93
Нормативные ссылки на: Прочие	
EN аутентичен ИСО/МЭК	
EN содержит требования ИСО/МЭК	
Документ внесен организацией СНГ	
Документ принят организацией СНГ	
Номер протокола	
Дата принятия	
Присоединившиеся страны	

Управление Ростехрегулирования	530 - Отдел стандартизации и сертификации информационных технологий, продукции электротехники и приборостроения
Технический комитет России	22 - Информационные технологии
Разработчик МНД	
Межгосударственный ТК	
Дата последнего издания (Дата изменения)	01.07.2003
Номер(а) изменении(й)	переиздание
Входит в сборник	
Количество страниц (оригинала)	46
Организация - Разработчик	ВНИИстандарт Госстандарта России
Статус	Действует
Стандарт ГОСТ Р ИСО/МЭК 12207-99 входит в рубрики классификатора:	КГС \ Измерительные приборы. Средства автоматизации и вычислительной техники \ Средства вычислительной техники и автоматизированные системы управления \ Виды представления информации и математическое обеспечение машин \ ОКС \ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. МАШИНЫ КОНТОРСКИЕ \ Документация на разработку программного обеспечения и системная документация \

Стандарт состоит из следующих разделов:

аттестация (validation)	Подтверждение экспертизой и представлением объективных доказательств того, что конкретные требования к конкретным объектам полностью реализованы. Примечания: В процессе проектирования и разработки аттестация связана с экспертизой продукта в целях определения его соответствия потребностям пользователя. Аттестацию обычно проводят для конечного продукта в установленных условиях эксплуатации. При необходимости аттестация может проводиться на более ранних стадиях. Термин «аттестован» используется для обозначения соответствующих состояний объекта. Может быть проведен ряд аттестаций, если они преследуют различные цели. (См. 2.18 ИСО 8402).
аудит (audit)	Проверка, выполняемая компетентным органом (лицом) с целью обеспечения независимой оценки степени соответствия программных продуктов или процессов установленным требованиям.
базовая линия (baseline)	Официально принятая версия элемента конфигурации, независимая от среды, формально обозначенная и зафиксированная в конкретный момент времени жизненного цикла элемента конфигурации.
верификация (verification)	Подтверждение экспертизой и представлением объективных доказательств того, что конкретные требования полностью реализованы. Примечания: В процессе проектирования и разработки верификация связана с экспертизой результатов данной работы в целях определения их соответствия установленным требованиям. Термин «верифицирован» используется для обозначения соответствующих состояний проверенного объекта. (См. 2.17 ИСО 8402).
версия (version)	Определенный экземпляр объекта. Примечание: В результате модификации версии программного продукта появляется новая версия, подвергающаяся управлению конфигурацией.

выпуск (release)	Конкретная версия элемента конфигурации, которая доступна для реализации конкретной цели (например, тестируемый выпуск).
готовый продукт (off-the-shelf product)	Ранее разработанный и доступный для приобретения продукт, пригодный для использования в поставляемом или модифицированном виде. **оператор** (operator)
договор (contract)	Обязательное соглашение между двумя сторонами, подкрепленное законодательно, или аналогичное соглашение внутри данной организации по предоставлению программной услуги; на поставку, разработку, производство, эксплуатацию или сопровождение программного продукта.
заказ (acquisition)	Процесс приобретения системы, программного продукта или программной услуги.
заказчик (acquirer)	Организация, которая приобретает или получает систему, программный продукт или программную услугу от поставщика. Примечание: Заказчиком может быть оптовый или розничный покупатель, клиент, владелец, пользователь.
защита (security)	Сохранение информации и данных так, чтобы недопущенные к ним лица или системы не могли их читать или изменять, а допущенные лица или системы не ограничивались в доступе к ним.
заявка на подряд (request for proposal [tender])	Документ, используемый заказчиком в качестве средства для объявления о своих намерениях выступить в качестве потенциального покупателя конкретной системы, программного продукта или программной услуги.
квалификационное испытание (qualification testing)	Испытание (тестирование), проводимое разработчиком, при необходимости санкционированное заказчиком, для демонстрации того, что программный продукт удовлетворяет установленным требованиям и готов к использованию в заданных условиях эксплуатации.

квалификационное требование (qualification requirement)	Набор критериев или условий, которые должны быть удовлетворены для того, чтобы квалифицировать программный продукт на соответствие установленным требованиям и готовность к использованию в заданных условиях эксплуатации.
квалификация (qualification)	Процесс демонстрации возможности объекта выполнять установленные требования (См. 2.13 ИСО 8402).
модель жизненного цикла (life cycle model)	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.
надзор (monitoring)	Проверка заказчиком или третьей стороной состояния работ, выполняемых поставщиком, и их результатов.
непоставляемое изделие (non-deliverable item)	Техническое или программное средство, которое не поставляется по условиям договора, но может быть применено при создании программного продукта.

обеспечение качества (quality assurance)	Все запланированные и систематически выполняемые в рамках системы качества работы; при необходимости объективные доказательства, обеспечивающие уверенность в том, что объект будет полностью соответствовать установленным требованиям качества. Примечания: Существуют как внешние, так и внутренние цели обеспечения качества внутреннее обеспечение качества — внутри организации обеспечение качества создает уверенность у руководства; внешнее обеспечение качества — в договорных или других ситуациях обеспечение качества создает уверенность у потребителя или других лиц. Некоторые виды работ по управлению качеством и обеспечению качества взаимосвязаны. Если требования к качеству не полностью отражают потребности пользователя, то обеспечение качества может не создать достаточной уверенности. (См. 3.5 ИСО 8402).
оценка (evaluation)	Систематическое определение степени соответствия объекта установленным критериям.
персонал сопровождения (maintainer)	Организация, которая выполняет работы по сопровождению.
пользователь (user)	Лицо или организация, которое использует действующую систему для выполнения конкретной функции. Примечание: Пользователь может также выполнять и другие роли, например, заказчика, разработчика или сопровождающего персонала.
поставщик (supplier)	Организация, которая заключает договор с заказчиком на поставку системы, программного продукта или программной услуги на условиях, оговоренных в договоре. Примечания: Синонимами термина «поставщик» являются термины «подрядчик», «производитель», «оптовик» или «продавец». Заказчик может определить в качестве

	поставщика подразделение собственной организации.
программная услуга (software service)	Выполнение работ, заданий или обязанностей, связанных с программным продуктом, таких как разработка, сопровождение или эксплуатация.
программно-аппаратное средство (firmware)	Сочетание технических устройств и машинных команд или используемых вычислительной машиной данных, постоянно хранящихся на техническом устройстве в виде постоянного программного средства. Данное программное средство не может изменяться только средствами программирования.
программный модуль (software unit)	Отдельно компилируемая часть программного кода (программы).
программный продукт (software product)	Набор машинных программ, процедур и, возможно, связанных с ними документации и данных.

процесс (process)	Набор взаимосвязанных работ, которые преобразуют исходные данные в выходные результаты. Примечание: Термин «работы» подразумевает использование ресурсов (см. 1.2 ИСО 8402).
разработчик (developer)	Организация, выполняющая работы по разработке (включая анализ требований, проектирование, приемочные испытания) в процессе жизненного цикла программных средств.
система (system)	Комплекс, состоящий из процессов, технических и программных средств, устройств и персонала, обладающий возможностью удовлетворять установленным потребностям или целям.
снятие с эксплуатации (retirement)	Прекращение активной поддержки действующей системы со стороны эксплуатирующей или сопровождающей организации, частичная или полная замена ее новой системой или ввод в действие модернизированной системы.

соглашение (agreement)	Определение границ и условий, при которых будут осуществляться рабочие взаимоотношения.
тестируемость (testability)	Степень, до которой могут быть запланированы объективность и реализуемость тестирования, проверяющего соответствие требованию.
тестовое покрытие (test coverage)	Степень, до которой с помощью контрольных примеров проверяют требования к системе или программному продукту.
техническое задание (statement of work)	Документ, используемый заказчиком в качестве средства для описания и определения задач, выполняемых при реализации договора.
элемент конфигурации (configuration item)	Объект внутри конфигурации, который удовлетворяет функции конечного использования и может быть однозначно определен в данной эталонной точке.

4 ПРИКЛАДНОЕ ПРИМЕНЕНИЕ НАСТОЯЩЕГО СТАНДАРТА

4.1 Построение стандарта

4.1.1 Процессы жизненного цикла

4.1.1.1 Основные процессы жизненного цикла

4.1.1.2 Вспомогательные процессы жизненного цикла

К вспомогательным процессам, ГОСТ Р ИСО/МЭК 12207-99 относит:

- **Аттестация**. Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.
- **Аудит**. 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).
- **Верификация**. Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.
- **Документирование**. Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.
- **Обеспечение 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.
- **Решения проблемы**. 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.
- **Совместный анализ**. 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.
- **Управление конфигурацией**. Определяет работы по управлению конфигурацией.

4.1.1.3 Организационные процессы жизненного цикла

ГОСТ Р ИСО/МЭК 12207-99 относит вспомогательные процессы к организационным процессам жизненного цикла.

4.1.1.4 Организационные процессы жизненного цикла

ГОСТ Р ИСО/МЭК 12207-99 относит процессы:

- **Управление** (основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла);

- **Создание инфраструктуры** (основные работы по созданию основной структуры процесса жизненного цикла);
- **Усовершенствование** (основные работы, которые организация выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла), а также
- **Обучение** (работы по соответствующему обучению персонала) к организационным процессам жизненного цикла.

4.1.2 Процесс адаптации

4.1.3 Взаимосвязи между процессами и организациями

5 ОСНОВНЫЕ ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА

ГОСТ Р ИСО/МЭК 12207-99 определяет процессы жизненного цикла, как основные:

Заказ;

Поставка;

Разработка;

Эксплуатация;

Сопровождение.

5.1 Процесс заказа

5.1.1 Подготовка

5.1.2 Подготовка заявки на подряд

5.1.3 Подготовка и корректировка договора

5.1.4 Надзор за поставщиком

5.1.5 Приемка и закрытие договора

5.2 Процесс поставки

5.2.1 Подготовка

5.2.2 Подготовка ответа

5.2.3 Подготовка договора

5.2.4 Планирование

5.2.5 Выполнение и контроль

5.2.6 Проверка и оценка

5.2.7 Поставка и закрытие договора

5.3 Процесс разработки

5.3.1 Подготовка процесса

5.3.2 Анализ требований к системе

5.3.3 Проектирование системной архитектуры

5.3.4 Анализ требований к программным средствам

5.3.5 Проектирование программной архитектуры

5.3.6 Техническое проектирование программных средств

5.3.7 Программирование и тестирование программных средств

5.3.8 Сборка программных средств

5.3.9 Квалификационные испытания программных средств

5.3.10 Сборка системы

5.3.11 Квалификационные испытания системы

3.6. Выводы

Изучая раздел “Управление качеством процесса разработки”, мы пришли к выводам о том, что:

- Стандарт ISO 12207, называемый «Информационная технология. Процессы жизненного цикла программных средств» (Information technology. Software life cycle processes) принят в России, как ГОСТ Р ИСО/МЭК 12207-99.
- Стандарт описывает жизненный цикл программного обеспечения целиком.
- Стандарт определяет основные процессы жизненного цикла программного обеспечения, такие, как: «Заказ, Поставка, Разработка, Эксплуатация и Сопровождение».
- Стандарт определяет вспомогательные процессы жизненного цикла программного обеспечения, такие, как: «Аттестация, Аудит, Верификация, Документирование, Обеспечение качества, Решения проблемы, Совместный анализ, Управление конфигурацией».
- Стандарт определяет организационные процессы жизненного цикла программного обеспечения, такие, как: «Управление, Создание, Усовершенствование и Обучение». Кроме того, вспомогательные процессы стандарт также относит к организационным процессам.
- В стандарте приняты следующие определения: «Аттестация, Аудит, Базовая линия, Верификация, Версия, Выпуск, Готовый продукт, Договор, Заказ, Заказчик, Защита, Заявка на подряд, Квалификационное испытание, Квалификационное требование, Квалификация, модель жизненного цикла, Надзор, Непоставляемое изделие, Обеспечение качества, Оценка, Персонал сопровождения, Пользователь, Поставщик, Программная услуга, Программно-аппаратное средство, Программный модуль, Программный продукт, Процесс, Разработчик, Система, Снятие с эксплуатации, Соглашение, Тестируемость, Тестовое покрытие, Техническое задание, Элемент конфигурации».

3.7. Вопросы и задания для самостоятельной работы студента по теме «Управление качеством процесса разработки»

1) Дайте определение понятию «Атрибуты качества» программного продукта.
2) Что такое жизненный цикл ПО?
3) Дайте определения любому из терминов: «аттестация (validation), аудит (audit), базовая линия (baseline),

верификация (verification), версия (version), выпуск (release), готовый продукт (off-the-shelf product), договор (contract), заказ (acquisition), заказчик (acquirer), защита (security), заявка на подряд (request for proposal [tender]), квалификационное испытание (qualification testing), квалификационное требование (qualification requirement), квалификация (qualification), модель жизненного цикла (life cycle model), надзор (monitoring), непоставляемое изделие (non-deliverable item), обеспечение качества (quality assurance), оператор (operator), оценка (evaluation), персонал сопровождения (maintainer), пользователь (user), поставщик (supplier), программная услуга (software servise), программно-аппаратное средство (firmware), программный модуль (software unit), программный продукт (software product), процесс (process), разработчик (developer), система (system), снятие с эксплуатации (retirement), соглашение (agreement), тестируемость (testability), тестовое покрытие (test coverage), техническое задание (statement of work), элемент конфигурации (configuration item)».

4) Дайте определения основным и вспомогательным и организационным процессам: «аттестации, аудита, верификации, документирования, заказа, обеспечения качества, обучения, поставки, разработки, решения проблем, совместного анализа, создания инфраструктуры, сопровождения, управления, управления конфигурацией, усовершенствования, эксплуатации».

Литература по теме «Управление качеством процесса разработки»

[1] ГОСТ Р ИСО 9000-2001.

[2] Брауде Э. Технология разработки программного обеспечения. СПб, Питер, 2004.

[3] ГОСТ Р ИСО/МЭК 12207-99

Тема 4. Хорошие практики управления качеством процесса разработки ПО. ITIL и ITSM

4.1. БИЗНЕС и ИТ

"...По-моему, появление слаженной, сработавшейся команды в любом проекте можно считать успехом! И об успехе проекта нужно судить точно так же: не просто оценивать качество программного продукта, а учитывать, появилась ли в результате работы сплоченная команда разработчиков, которая готова немедленно продолжать совместную работу и приступить к следующему проекту..."
Великий Вождь Народов[13]

Фактическим заказчиком ИТ-услуг является бизнес. В связи с очевидностью влияния качества оказываемых ИТ-сервисов на бизнес, применение ИТ-технологий как вспомогательных сервисов приводит к прямой зависимости бизнеса от ИТ-организаций, подразделений или служб (ИТ-организации). Стремление бизнеса к сокращению затрат на ИТ-технологии приводит к снижению качества ИТ-услуг и, в конечном счёте, к увеличению приведённой стоимости таких услуг из-за неизбежных затрат на их поддержание и модернизацию.

Фактическим исполнителем ИТ-сервисов являются ИТ-организация. Для существования ИТ-организации необходимы заказы от бизнеса для разработки, поддержания и модернизации оказываемых ИТ-услуг. Вспомогательная роль ИТ-организации не позволяет ей участвовать в увеличении качества продукции заказчика, или её участие реализуется весьма опосредовано. ИТ-организация при этом полностью зависит от бизнеса.

Такой цикл взаимоотношений бизнеса и ИТ-организации не предназначен для непрерывного улучшения системы менеджмента качества бизнеса, и такой подход является устаревшим.

[13] Том Де Марко "Deadline. Роман об управлении проектами". М, Вершина, 2006, ISBN 5-9626-0132-7, глава 9. "Марков, генерал в отставке"

Для эффективного изменения схемы отношений ИТ и бизнеса необходимо усовершенствовать управление в ИТ сфере предоставления услуг.

Улучшение процессов в структуре ИТ позволяет бизнесу:

- Совершенствовать использование ресурсов;
- Быть более конкурентоспособным;
- Сокращать долю переделок;
- Устранять избыточные работы;
- Улучшать время и качество результатов проектов;
- Увеличивать доступность, надёжность и безопасность ИТ услуг, критичных для бизнеса;
- Обосновывать стоимость качества предоставляемых услуг;
- Предоставлять обслуживание, удовлетворяющее требованиям бизнеса, заказчиков и пользователей бизнес-подразделений;
- Интегрировать централизованные процессы;
- Документировать и распределять роли и обязанности внутри сервисных отделов;
- Учиться на предыдущем опыте;
- Предоставлять очевидные показатели производительности.

Целью улучшения процессов в ИТ-организации является предоставление именно тех услуг, которые требуются в данный момент, без потерь времени и без снижения производительности "по техническим причинам".

Для улучшения процессов в структуре ИТ-организации необходимы мероприятия по внедрению системы менеджмента качества (СМК) по разработке программного обеспечения, по автоматизации и управлению сетевым оборудованием, серверами и корпоративными приложениями, по хранению и безопасности данных и по управлению рабочими местами пользователей. Руководство ИТ-организации получает возможность быстро и адекватно оценивать качество работы своих отделов, определять текущие затраты и ресурсы и планировать необходимые изменения. Критерием оценки эффективности работы такой ИТ-организации становится снижение эксплуатационных расходов в формировании стратегии бизнеса, а ИТ-организация становится провайдером ИТ-услуг.

Мировым стандартом и руководством по развертыванию и сопровождению ИТ-сервисов сегодня фактически является ITIL (IT Infrastructure Library).

Концепция ITIL представлена в виде библиотеки их семи книг, в том числе:

- Поддержка услуг (Service Support);
- Предоставление/Доставка услуг (Service Delivery);
- Планирование внедрения Управления услугами (Planning to Implement Service Management);
- Управление приложениями (Application Management);
- Управление инфраструктурой ИКТ (ICT Infrastructure Management);
- Управление безопасностью (Security Management);
- Бизнес-перспектива (The Business Perspective).

Кроме того, в комплект документации ITIL включена “дополнительная” (complementary) книга — Управление конфигурациями ПО (Software Asset Management).

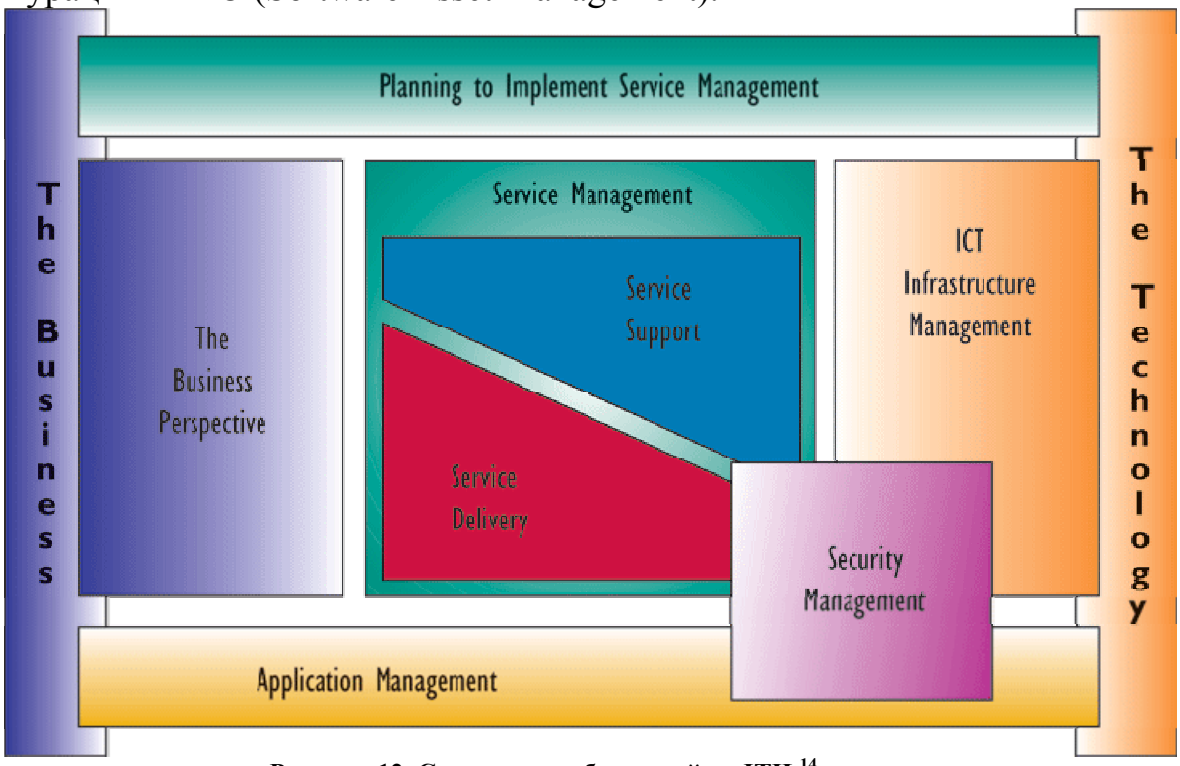

Рисунок 12. Структура публикаций по ITIL[14]

Библиотека ITIL разработана Центральным агентством по вычислительной технике и телекоммуникациям (Central Computer and Telecommunications Agency — CCTA) правительства Великобритании.

[14] Ввиду отсутствия официальных российских документов по ITIL, эта и следующие иллюстрации приводятся в виде “как есть”. Источник: OGC.

После объединения ССТА с Государственной торговой палатой (Office of Government Commerce — OGC) в 2001 г., владельцем библиотеки ITIL является OGC.

Форум по ИТ Сервис-менеджменту (Information Technology Service Management Forum — itSMF) является единственной имеющей международное признание независимой группой пользователей, занимающейся вопросами ИТ Сервис-менеджмента. Отделения форума itSMF есть в таких странах, как ЮАР, Бельгия, Германия, Австрия, Швейцария, Канада, США и Австралия и все они сотрудничают в рамках форума itSMF International. Региональные отделения itSMF способствуют обмену информацией и опытом, что позволяет ИТ-организациям повысить качество поставляемых услуг. Кроме того, региональные отделения издают информационные бюллетени и используют Web-сайты для обмена информацией, способствуя тем самым дальнейшему развитию библиотеки ITIL.

“Exameninstituut voor Informatica” (EXIN) из Голландии и британская “Information Systems Examination Board” (ISEB) в тесном сотрудничестве с OGC и itSMF, разработали профессиональную систему сертификации в рамках библиотеки ITIL. Квалификационные сертификаты ITIL состоят из трех уровней:

- базовый сертификат по ИТ Сервис-менеджменту — Foundation Certificate in IT Service Management;
- сертификат специалиста по ИТ Сервис-менеджменту — Practitioner Certificate in IT Service Management]
- сертификат менеджера по ИТ Сервис-менеджменту — Manager Certificate in IT Service Management.

Система сертификации ITIL основана на требованиях по эффективному выполнению соответствующей роли в ИТ-организации.

Базовый сертификат предназначен для всего персонала, который должен быть в курсе главных задач ИТ-организации и их взаимосвязей. Экзамен на получение базового сертификата включает оценку знаний Службы *Service Desk* и процессов *Управления Инцидентами*, *Проблемами, Изменениями*, *Конфигурациями*, *Релизами, Уровнем Услуг*, *Доступностью*, *Мощностями*, *Непрерывностью ИТ-услуг* и *Финансами ИТ*.

После получения базового сертификата разрешается сдавать экзамены на получение сертификата специалиста и менеджера. Специалисты получают навыки практического осуществления отдельных ITIL-процессов и задач в рамках таких процессов, как *Управление Инцидентами, Изменениями* и *Уровнем Услуг*. Менеджеры получают теоретические знания о том, как управлять всеми процессами, перечисленными в базовом сертификате, как консультировать по структуре и оптимизации процессов и как осуществлять внедрение этих процессов.

Библиотека ITIL представляет собой гораздо большее, чем серию полезных книг по ИТ Сервис-менеджменту. В практический опыт ITSM входит вся индустрия, включающая организации, инструментарии, тренинговые и консалтинговые услуги, соответствующие методологическое обеспечение и публикации. Библиотека ITIL это не только структурированная основа, она является подходом и философией, разделяемой теми, кто использует в своей работе передовой опыт ITIL.

Положения, изложенные в библиотеке ITIL, являются ИТ-стандартами управления и внедрения ИТ-сервисов.

С 2005 началось внедрение стандарта ISO 20000[15] основанного на британском стандарте BS 15000 и на практических рекомендациях из материалов библиотеки ITIL. В Российской федерации ожидается выход национального стандарта[16] основанного на рекомендациях библиотеки ITIL. Национальный стандарт позволит определять требования к управлению услугами ИТ и по содержанию будет близок к ГОСТ Р ИСО 9000—2001.

По мнению первого заместителя председателя партнёрства itSMF России, Михаила Потоцкого "…только с принятием стандарта ISO 20000 появилась возможность оценить, сколько процессов сертифицировано и в каком объеме. До этого преобладала субъективная оценка…" [19].

Рисунок 13. Михаил Потоцкий. Первый заместитель председателя партнерства itSMF России

В соответствии с рекомендациями ITIL, ИТ-организация, подразделение или служба является основным производственным подразделением и отношения между ИТ и другими подразделениями следует строить как со сторонней сервисной организацией.

По рекомендациям ITIL деятельность по обеспечению сервисов представляется в виде процессов. Процесс - это последовательность шагов, направленная на достижение определенной цели или результата. Каждый процесс должен иметь собственника (владельца) этого процесса. Владелец процесса - это физическое лицо, отвечающее за сквозное выполнение процесса. Управляемо только то, что измеряемо. Поэтому с каждым процессом должен быть связан некоторый набор метрик.

[15] ISO/IEC 20000-1:2005 “Information technology — Service Management. Part 1: Specification”. ISO/IEC 20000-2:2005 “Information technology — Service Management. Part 2: Code of Practice”.

[16] “Основные положения и словарь” (аналог ISO/IES 20000-1:2005) и “Практическое руководство” (аналог ISO/IES 20000-2:2005).

В отличие от “технологического” подхода, при котором ИТ-организация предоставляет системы, программы, модули, технологии и многое другое, взаимодействие ИТ-организации и бизнес-подразделений основывается на описании предоставляемых услуг в терминах и определениях бизнес-процессов с указанием уровня, на котором предоставляются услуги. Эффективность такого взаимодействия определяются критерием оценки качества работы ИТ, при котором развернутая ИТ-инфраструктура соответствует требованиям и ожиданиям бизнеса. Предоставление ИТ-услуги организуется в виде процесса, а управление ИТ-инфраструктурой описывается как комплекс процессов, затрагивающих различные структурные подразделения и направленных на достижение определенных целей. Для каждого процесса определяются роли, процедуры, входящая и исходящая информация. Деятельность по процессу предполагает эффективное ролевое взаимодействие, направленное на достижение поставленной цели независимо от места участника процесса в организационной структуре ИТ-организации.

Процессная модель ITIL включает следующие группы процессов управления ИТ:

- уровень инфраструктуры (ICT Infrastructure Management): дизайн и планирование, распространение, сопровождение и техническая поддержка (Управление инфраструктурой состоит из управления системами, сетями, базами данных, ПК и распространением ПО);
- уровень поддержки услуг (Service Support): управление инцидентами, управление проблемами, управление конфигурациями, управление изменениями, управление релизами;
- уровень предоставления услуг (Service Delivery): управление уровнем услуг, финансовое управление ИТ-услугами, управление доступностью, управление непрерывностью услуг, управление мощностями.

Взаимодействие бизнеса и ИТ-организацией представлены в процессной модели ITIL тремя уровнями:

- операционный (взаимодействие с пользователями);
- тактический (взаимодействие с заказчиками);
- и стратегический (согласование целей бизнеса и ИТ).

Центральное место в библиотеке ITIL занимают процессы тактического и операционного уровней. Они относятся к разделу Сервис Менеджмента (управления услугами).

Каждая из книг библиотеки ITIL содержит описание того, что необходимо для организации ИТ Сервис-менеджмента, и предлагает структурированную основу для планирования процессов, ролей и видов деятельности, и определяет связи между ними и необходимые виды коммуникации.

Библиотека ITIL частично основана на таких системах качества, как стандарты серии ISO-9000, и общие схемы обеспечения качества (Total Quality frameworks), предлагаемые Европейской организацией Управления Качеством (European Foundation of Quality Management — EFQM). Библиотека ITIL поддерживает эти системы за счёт четкого описания процессов и передового опыта ИТ Сервис-менеджмента.

Книги “Поддержка услуг” (Service Support) и “Предоставление услуг” (Service Delivery) занимают центральное место в библиотеке ITIL.

Подробное и пока единственное описание библиотеки ITIL на русском языке представлено в книге форума itSMF "Введение в ИТ Сервис-менеджмент" [20], подготовленной под общей редакцией Михаила Юрьевича Потоцкого и при поддержки со стороны российской компании IT-Expert.

4.2. Поддержка услуг (Service Support)

Операционный уровень ITIL представлен книгой “Поддержка услуг” (Service Support).

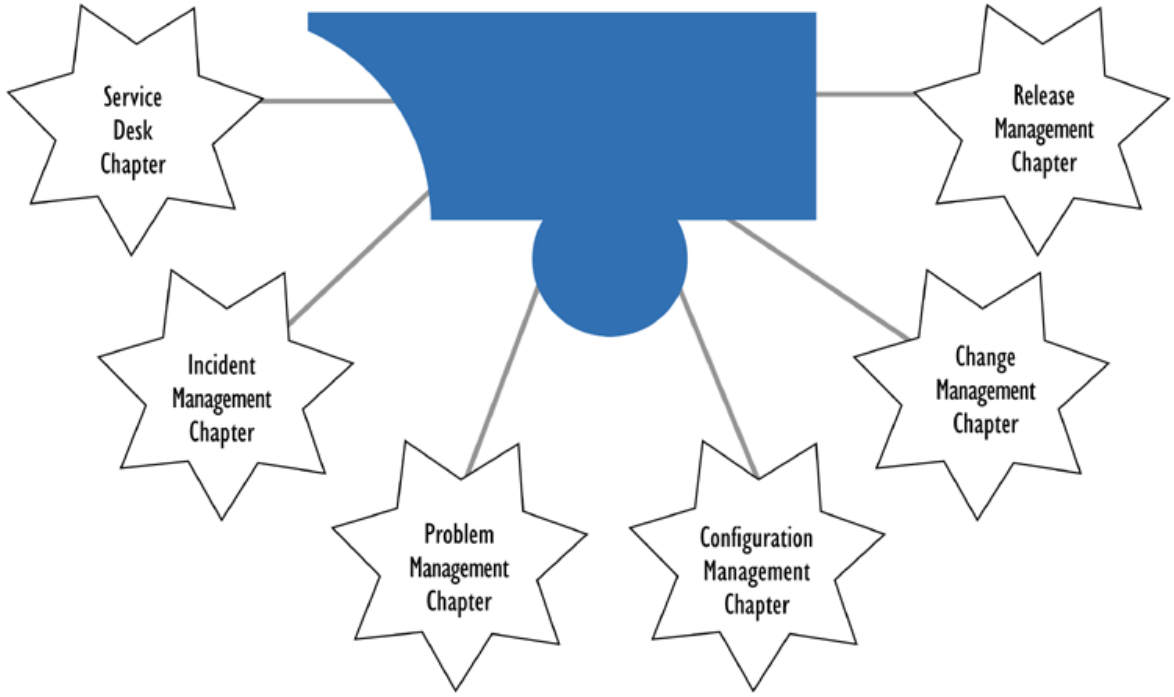

Рисунок 14. Состав книги Service Support

В книге “Поддержка услуг” (Service Support) описывается, каким образом заказчик может получить доступ к соответствующим услугам для поддержки своего бизнеса:

- Служба Service Desk;
- Управление Инцидентами (Incident Management);
- Управление Проблемами (Problems Management);
- Управление Конфигурациями (Configuration Management);
- Управление Изменениями (Change Management);
- Управление Релизами (Release Management).

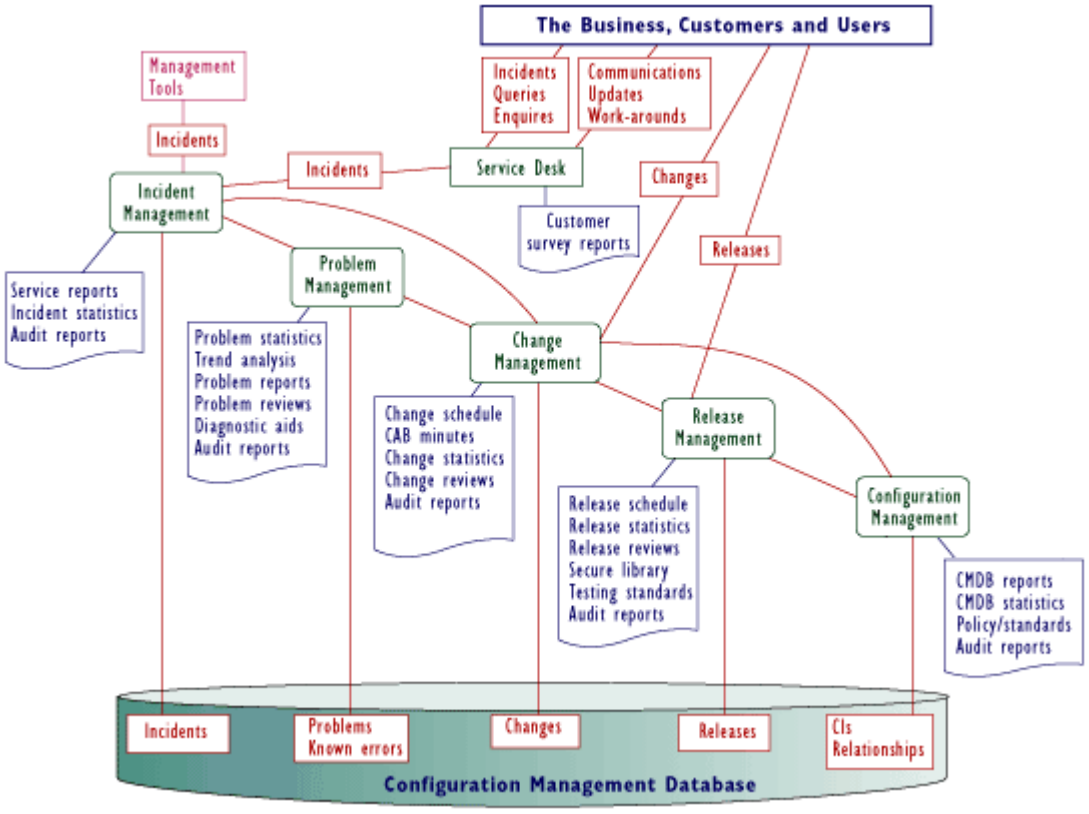

Рисунок 15. Модель процесса Service Support

Служба *Service Desk*

Служба *Service Desk* является точкой контакта пользователя с ИТ-организацией.

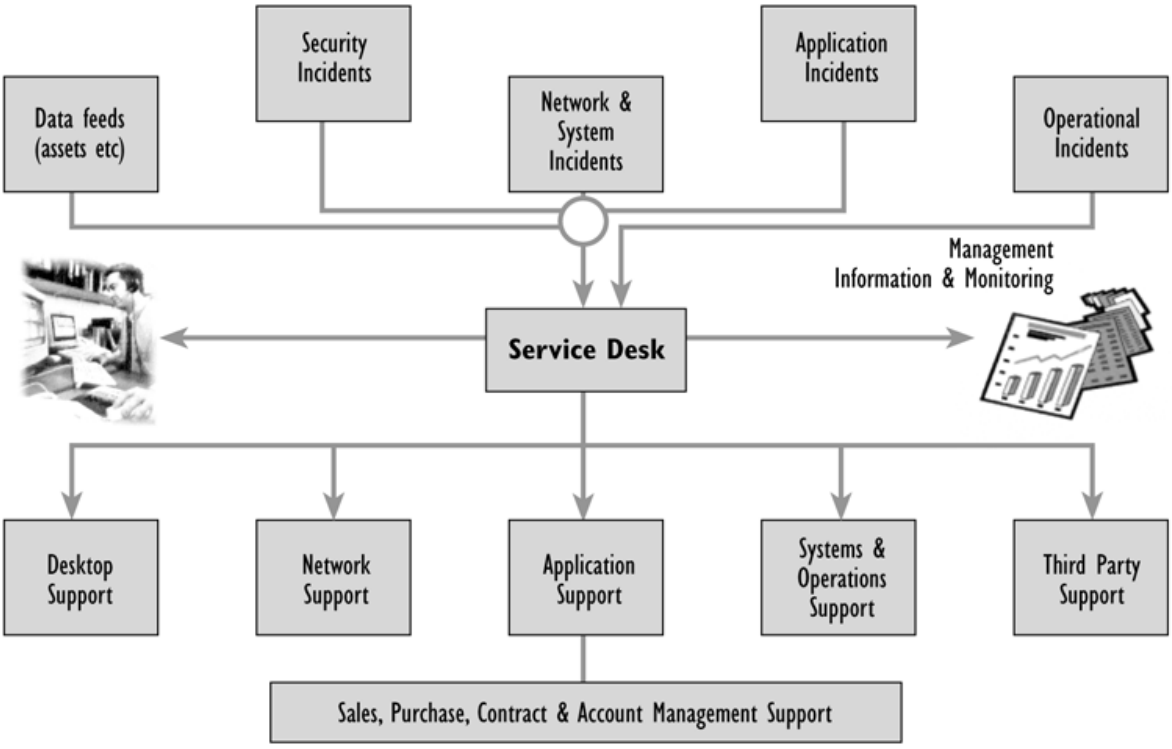

Рисунок 16. Служба Service Desk и инфраструктура инцидентов

Служба *Service Desk* имеет следующие задачи: регистрация, решение и отслеживание инцидентов[17], получение и отслеживание запросов на изменения.

Управление Инцидентами (Incident Management)

Главным достоинством процесса *Управления Инцидентами* является установление различия между быстрым восстановлением услуги и установлением причины инцидента и ее устранением. Процесс *Управления Инцидентами* ориентирован на устранение инцидента и на быстрое возобновление предоставления услуг. Инциденты регистрируются, качество регистрационной информации определяет эффективность ряда других процессов.

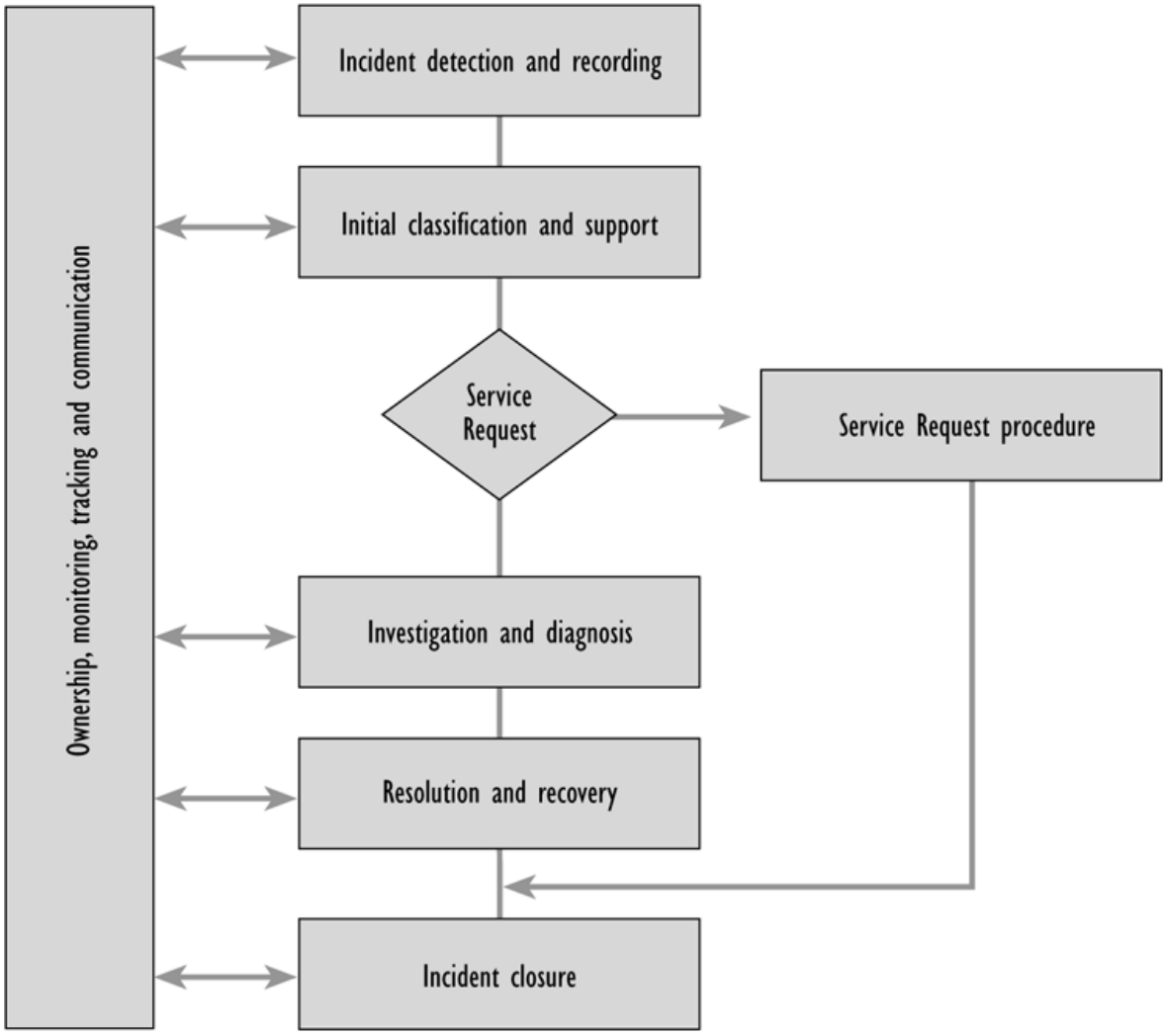

Рисунок 17. Диаграмма процесса жизненного цикла инцидента

Про процесс *Управления инцидентами* говорят[18], что он носит реактивный характер. Его часто называют службой технической помощи. Основная задача процесса — свести к минимуму случаи прерывания обслуживания. Процесс *Управления инцидентами*

[17] Инцидент — это любое событие, не являющееся частью стандартных операций по предоставлению услуги, которое привело или может привести к нарушению или снижению качества этой услуги.

[18] В. Притуленко. Управление IT службами на основе ITIL. Корпоративные Системы. 5, 2004

сочетает обработку обращений и эффективную поддержку первого, второго и третьего уровней. С процессом *Управления инцидентами* связаны процессы *Управления изменениями* и *Управления конфигурациями*.

Управление Проблемами (Problems Management)

Процесс *Управления Проблемами* предназначен для установления корневой причины проблемы. Проблема может возникнуть из-за наличия выявленных инцидентов. Конечной целью процесса *Управления Проблемами* является предотвращение сбоев везде, где это возможно.

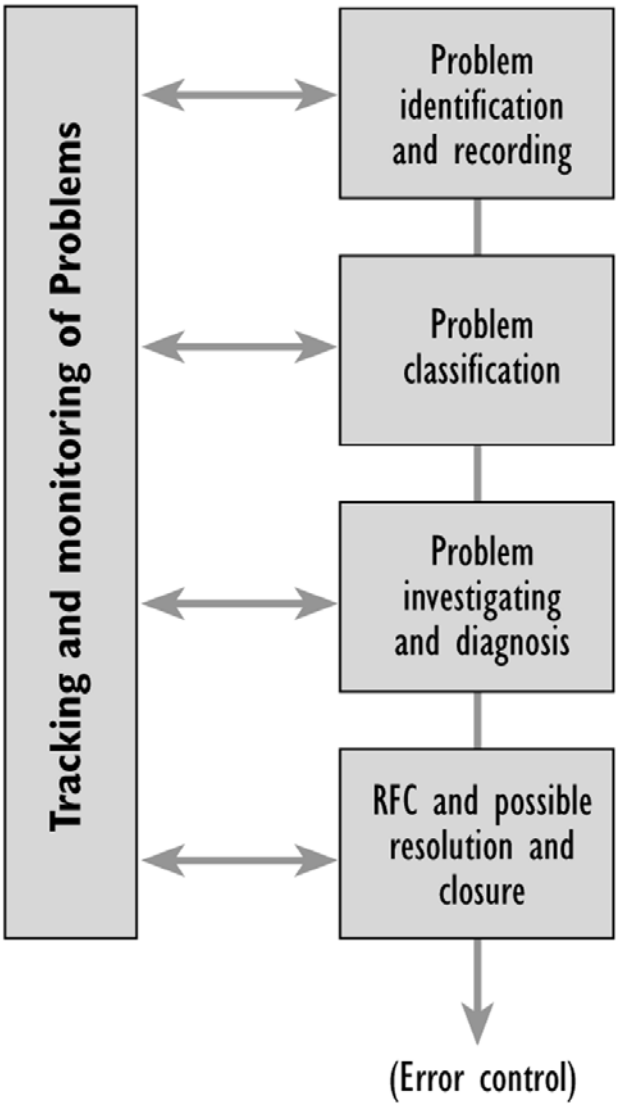

Рисунок 18. Процесс контроля проблем

При обнаружении причин возникшей проблемы (например, определены ошибки), принимается бизнес-решение о необходимости делать улучшения в инфраструктуре для предотвращения возникновения новых инцидентов. Процесс *Управления проблемами* носит превентивный характер и направлен на снижение числа производственных неполадок. Процесс реализуется путем изучения источников их возникновения на основе информации о прошлых инцидентах и включает анализ тенденций и контроль известных ошибок

с расчетом на устранение их источников в долговременной перспективе. Процесс *Управления Проблемами* связан с процессом *Управления инцидентами*.

Управление Конфигурациями (Configuration Management)

Задачами процесса *Управления Конфигурациями* являются: контроль изменяющейся ИТ-инфраструктуры - стандартизация и мониторинг статуса, сбор и управление документацией по ИТ-инфраструктуре, идентификация *Конфигурационных Единиц (Configuration item - CI)* - инвентаризация, верификация и регистрация и предоставление информации об ИТ-инфраструктуре для всех других процессов.

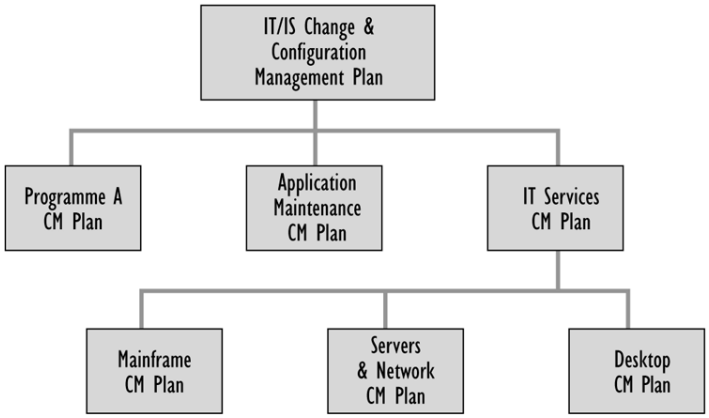

Рисунок 19. Пример управлением конфигурациями

Управление Изменениями (Change Management)

Процесс *Управление Изменениями* направлен на контроль проведения изменений в ИТ-инфраструктуре с целью определения необходимых изменений и способов их проведения с минимальным негативным воздействием на ИТ-услуги, при одновременном обеспечении контроля (отслеживании) изменений посредством консультаций и координации действий со всей организацией.

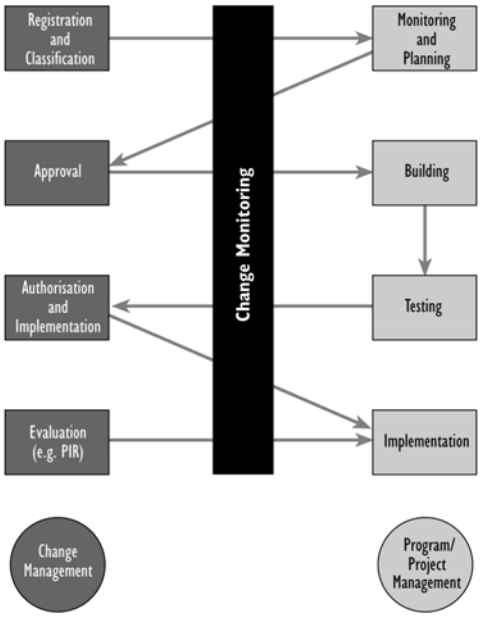

Рисунок 20. Пример процесса управления изменениями для большого проекта программы

Собственно изменения производятся из процесса *Управления Проблемами* и из некоторых других процессов. Процесс *Управление Изменениями* связан с процессом *Управления Конфигурациями*. Внесение изменений производится в соответствии с разработанной в ИТ-организации схемой, которая включает определение, планирование, создание и испытание, принятие окончательного решения о проведении, внедрение и оценку.

Управление Релизами (Release Management)

Главной задачей процесса *Управления Релизами* является обеспечение успешного развертывания релизов, включая интеграцию, проведение тестирования и хранение.

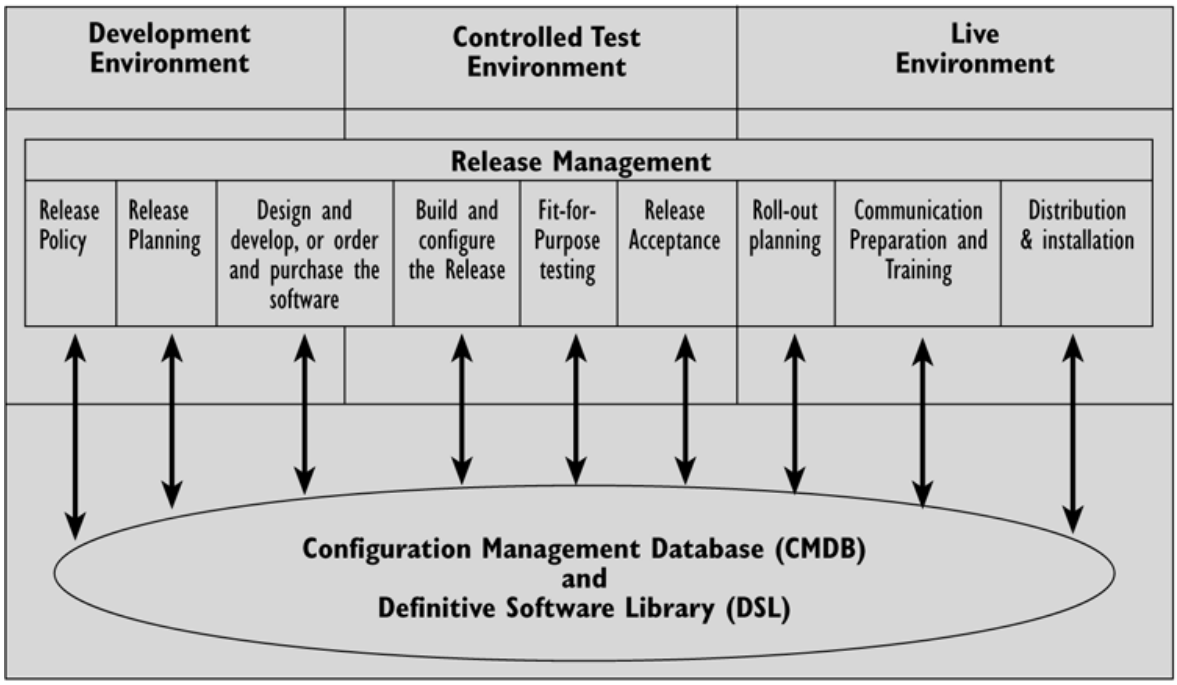

Рисунок 21. Диаграмма деятельности процесса Управления релизами

Процесс *Управление Релизами* предназначен для гарантирования того, что в использовании находятся только корректные и оттестированные версии авторизованного программного и аппаратного обеспечения. Процесс *Управление Релизами* связан с процессами *Управления Конфигурациями* и *Управлению Изменениями*. Релизом ITIL называется набор *Конфигурационных Единиц (Cl)* , которые совместно оттестированы и введены в активную рабочую среду.

4.3. Предоставление услуг (Service Delivery)

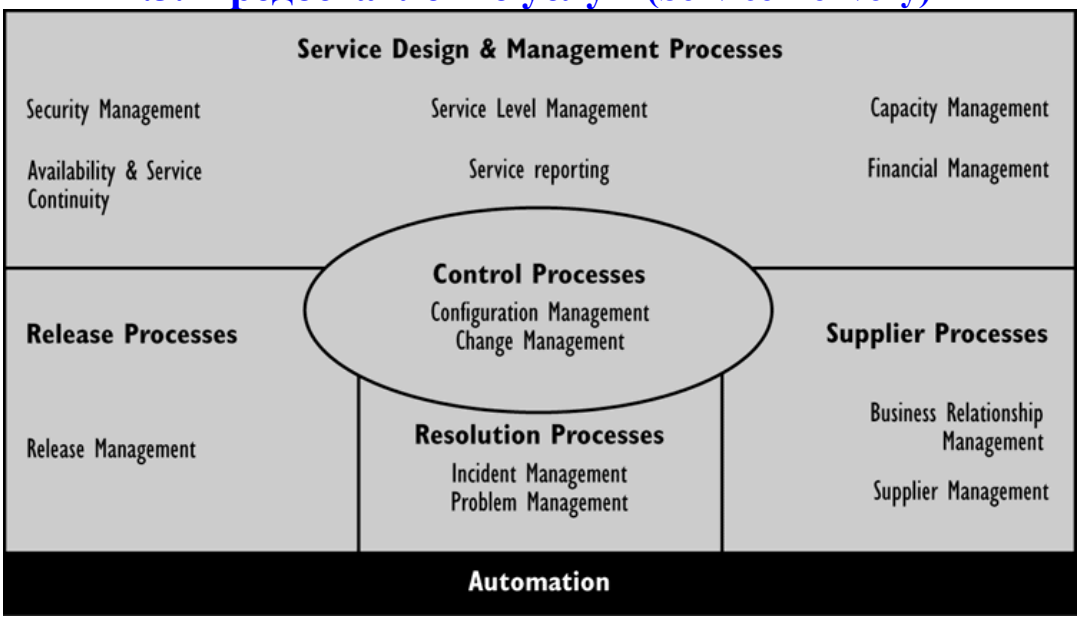

Рисунок 22. Процессы управления ИТ-сервисами по стандарту BS 15000

Тактический уровень ITIL “Предоставление услуг” (Service Delivery) основывается на стандарте BS 15000, который разработан британским институтом стандартов.

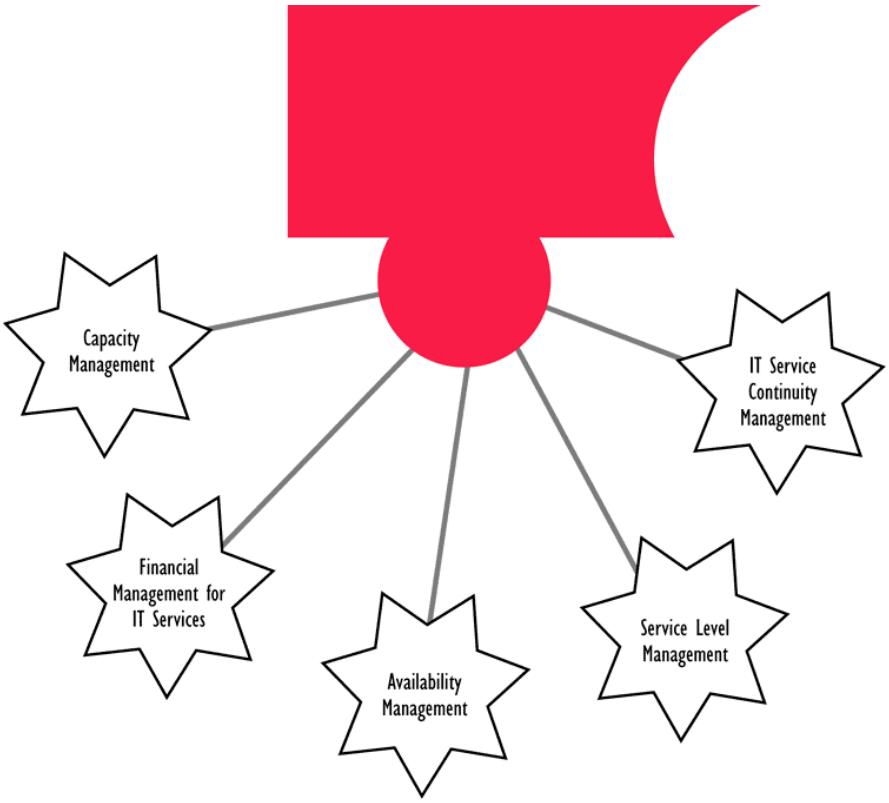

Рисунок 23. Состав книги Service Delivery

В книге “Предоставление услуг” (Service Delivery) описываются требования, необходимые для оказания услуг. В ней представлены следующие вопросы:

- Управление Уровнем Услуг (Service Level Management);
- Управление Финансами ИТ (Financial Management for IT Services);
- Управление Мощностями (Capacity Management);
- Управление Непрерывностью ИТ-услуг (IT Service Continuity Management);
- Управление Доступностью (Availability Management);
- Управление Информационной Безопасностью (Security Management).

Управление Уровнем Услуг (Service Level Management)

Процесс *Управление Уровнем Услуг* направлен на достижение ясных соглашений с заказчиком об ИТ-услугах и на реализацию этих соглашений (Service Level Agreement, SLA).

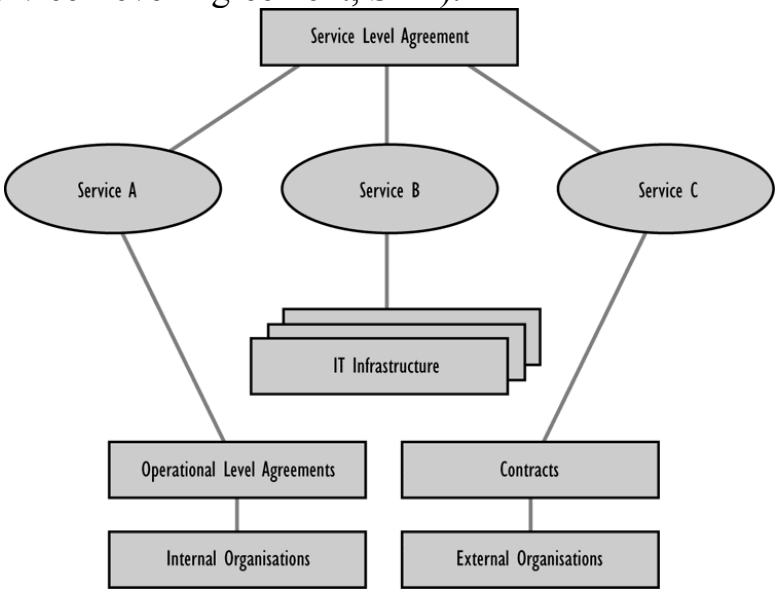

Рисунок 24. Структура поддержки SLA

Для *Управления Уровнем Услуг* необходима достоверная информация о потребностях заказчика, о предоставляемых ИТ-организацией технических средствах и о финансовых ресурсах. Процесс *Управление Уровнем Услуг* рассматривает услуги, предоставляемые заказчику с акцентом на услугах, которые могут поставляться, и корректирует их спектр в соответствии со спецификой бизнеса компании. IT-подразделение позиционируется как провайдер услуг для бизнес-подразделений.

Введение SLA между подразделениями позволяет повысить взаимную ответственность по предоставлению услуг и обеспечить достижение конечного результата. В SLA содержатся характеристики сервиса, что позволяет избежать ненужных дискуссий и одновременно помогает специалистам и пользователям говорить на общем языке.

Одной из таких характеристик является доступность: отношение фактически обеспеченного времени сервиса к времени, предусмотренному в SLA.

Управление Финансами ИТ (Financial Management for IT Services)

Процесс *Управление Финансами ИТ* посвящён экономическим вопросам предоставляемых ИТ-услуг. К ним, например, относятся расходы, возникшие при предоставлении услуг, что необходимо для учета соотнесения расходов и доходов, цены и результата. Определение, отнесение расходов, их прогноз и отслеживание в ITIL называется ”бюджетированием и бухгалтерским учетом”.

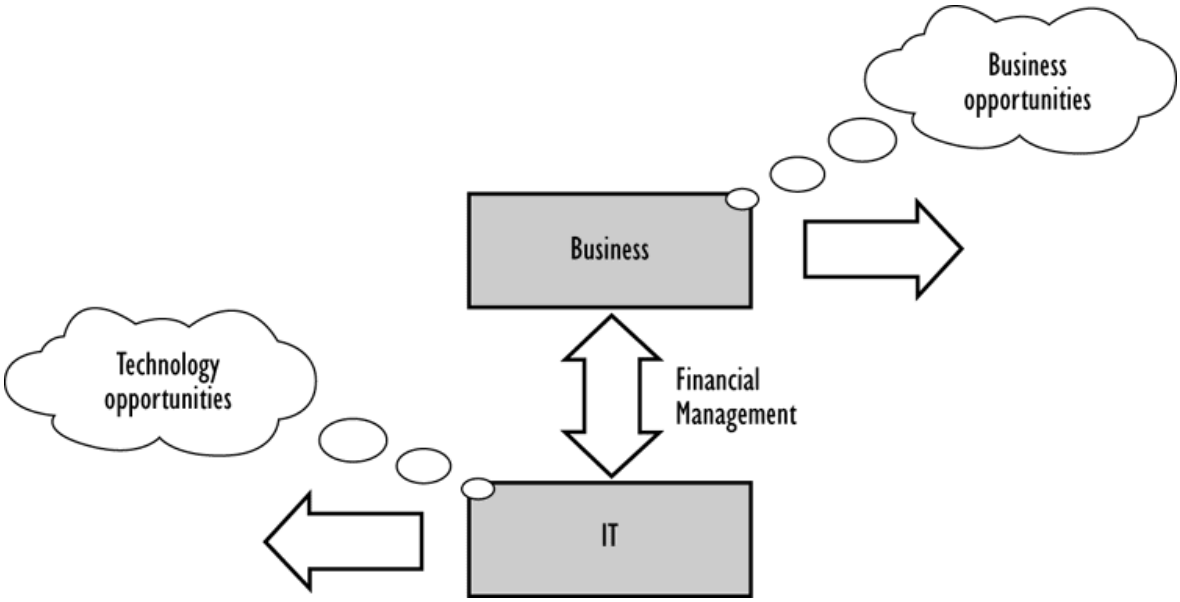

Рисунок 25. Роль процесса управления финансами

Эта деятельность повышает информированность о расходах и может использоваться при составлении бюджета. Процесс *Управление Финансами ИТ* описывает различные методы выставления счетов, включая определение цели выставления счетов за ИТ-услуги, а также определение ценообразования и аспекты бюджетирования.

Управление Мощностями (Capacity Management)

Процесс *Управление Мощностями* предназначен для оптимизации расходов, времени приобретения и размещения ИТ-ресурсов с целью обеспечения выполнения договоренностей с заказчиком.

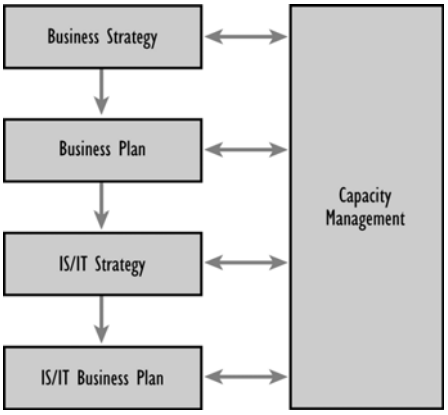

Рисунок 26. Процессы Управления Мощностями и Бизнеса

Процесс *Управления Мощностями* связан с процессами *Управления Ресурсами*, *Управления Производительностью*, *Управления Спросом на ИТ*, а также с процессами *Моделирование* и *Планирование мощностей*, и с управлением нагрузкой и с определением необходимого объема технических средств для работы приложений.

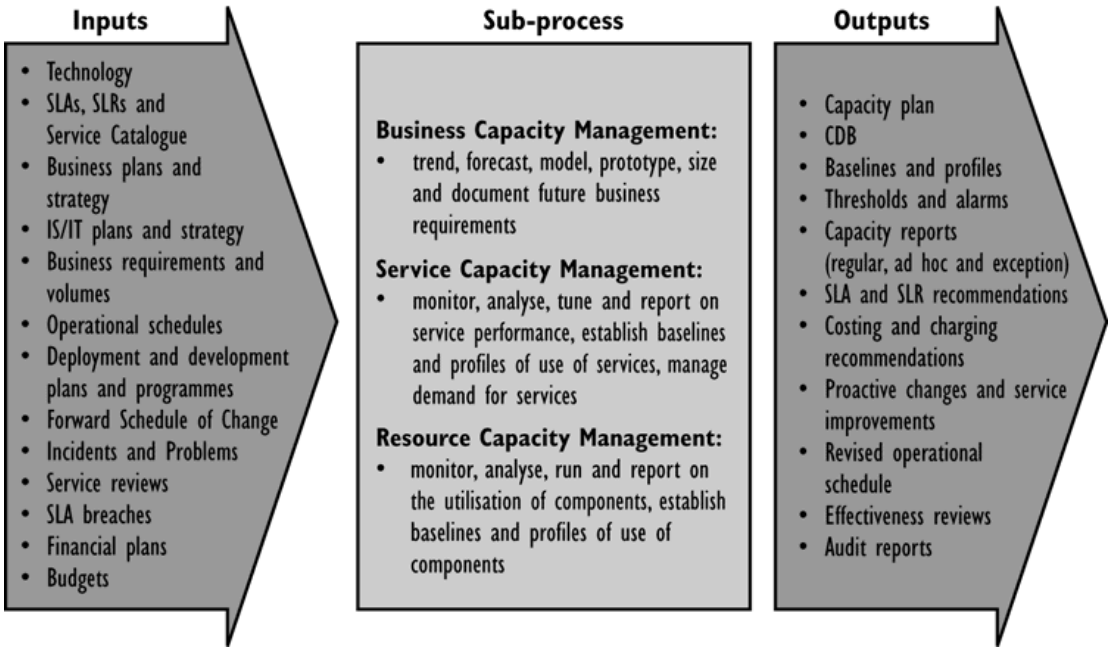

Рисунок 27. Процесс Управления Мощностями

Процесс *Управление Мощностями* позволяет выполнять планирование для обеспечения *Уровня Услуг*, как сейчас, так и в будущем.

Управление Доступностью (Availability Management)

Процесс *Управление Доступностью* обеспечивает размещение ресурсов, методов и технологий для поддержки согласованных с заказчиком *Уровня Доступности ИТ-услуг*.

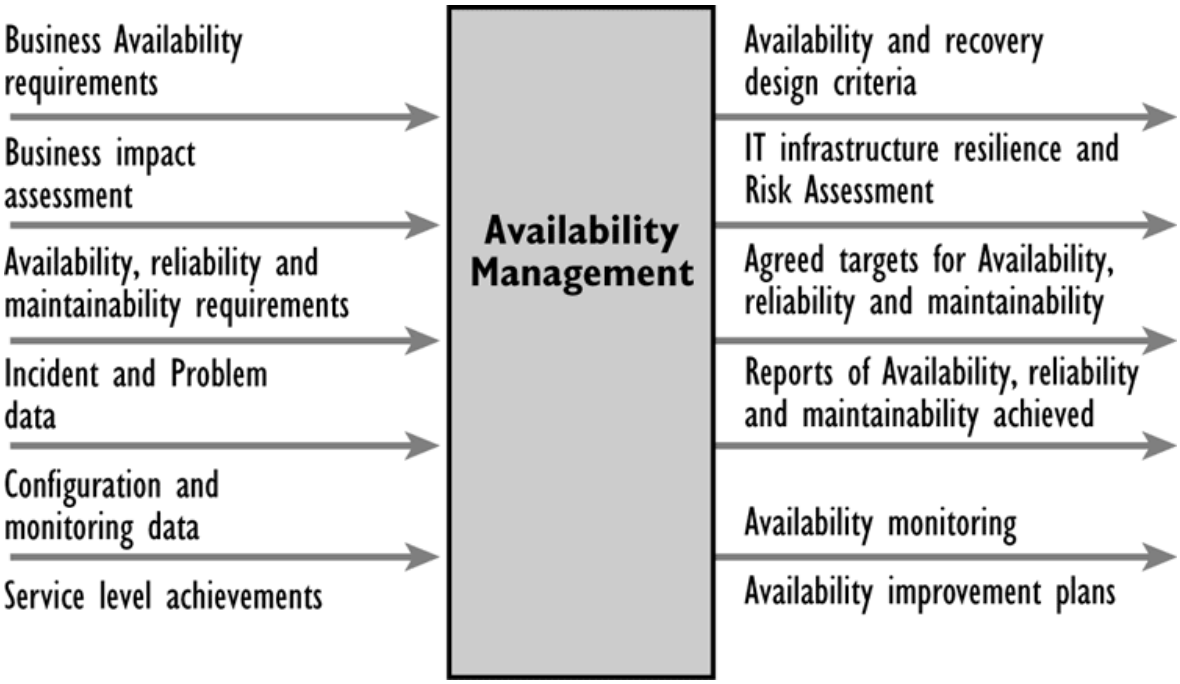

Рисунок 28. Процесс Управления Доступностью

Процесс *Управление Доступностью* предназначен для оптимизации обслуживания и позволяет разрабатывать способы минимизации числа инцидентов.

Управление Непрерывностью ИТ-услуг (IT Service Continuity Management)

Процесс *Управление непрерывностью ИТ-услуг* предназначен для подготовки и планирования способов устранения чрезвычайных ситуаций с ИТ-услугами в случае остановки бизнеса. В процессе *Управление непрерывностью ИТ-услуг* основное внимание уделяется связям между всеми компонентами, необходимым для защиты непрерывности деятельности компании при катастрофах для процесса *Управление Непрерывностью Бизнеса* - Business Continuity Management (BCM), а также средствам предотвращения таких катастроф.

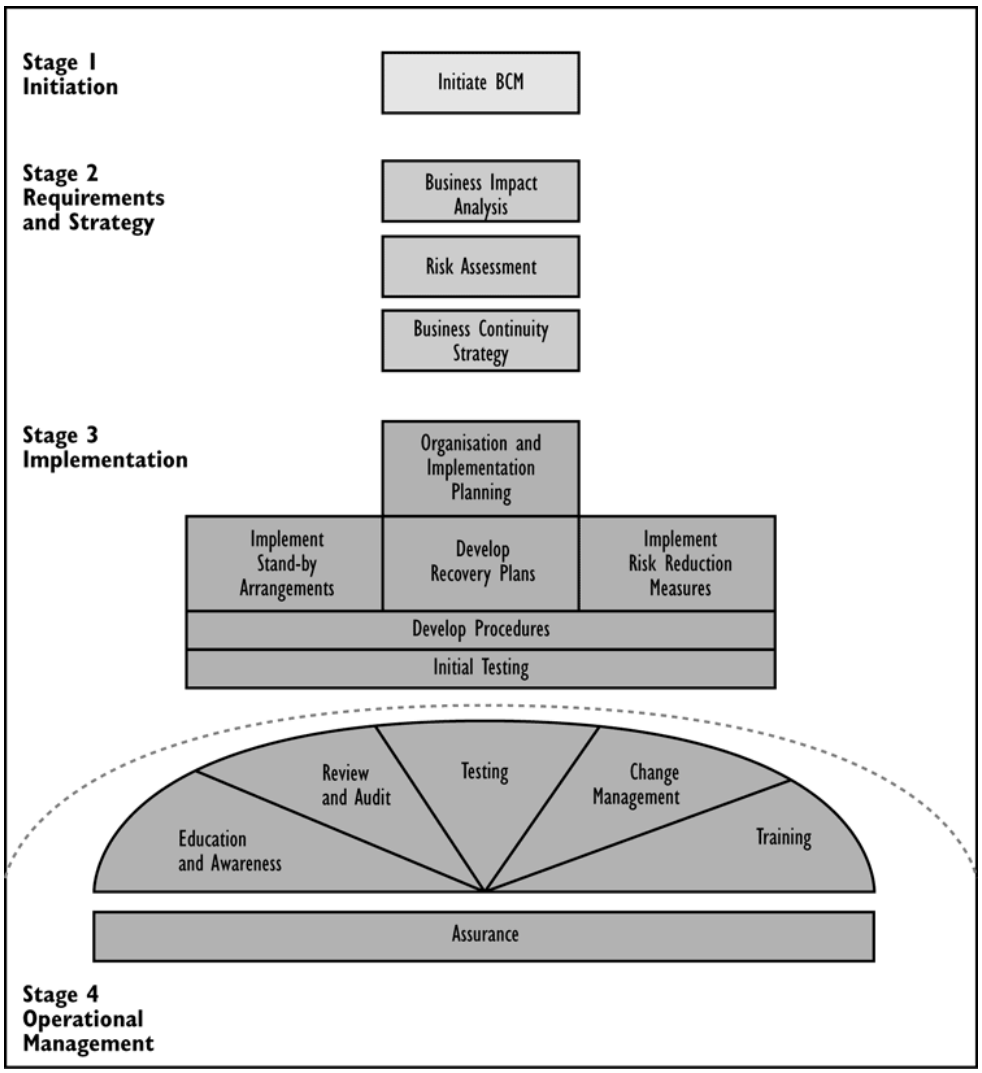

Рисунок 29. Процесс Управление Непрерывностью Бизнеса

Процесс *Управление непрерывностью ИТ-услуг* является процессом планирования и координации технических, финансовых и управленческих ресурсов, необходимых для обеспечения непрерывности услуг после катастроф, в соответствии с договоренностью с заказчиком. Несмотря на то, что чрезвычайные ситуации стали часто встречающимся явлением, многие руководители

экономят на внедрении процесса *Управления Непрерывностью ИТ-сервисов (IT Service Continuity Management — ITSCM).* Чрезвычайная ситуация намного серьезнее инцидента — почти всегда это приостановка бизнеса. Такие примеры, как нападение на Всемирный торговый центр в Нью-Йорке уже не выглядят гипотетическими. И если раньше традиционный процесс планирования непрерывности работы и восстановления функционирования в основном носил реактивный характер, то теперь процесс *Управления Непрерывностью ИТ-сервисов* выполняет превентивную роль, т. е. работает над предотвращением катастроф.

Управление Информационной Безопасностью (Security Management)

Процесс *Управления Информационной Безопасностью* входит в сферу компетенции общей информационной безопасности, задачей которой является обеспечение сохранности информации от известных рисков и уклонение от известных рисков. Целью процесса *Управления Информационной Безопасностью* является защита ценной информации. Ценность информации влияет на необходимый *Уровень Конфиденциальности, целостности и доступности*.

Процесс *Управления Информационной Безопасностью* имеет жесткие связи с другими процессами. Некоторые виды деятельности по обеспечению безопасности осуществляются другими процессами библиотеки ITIL, под контролем процесса *Управления Информационной Безопасностью*. Процесс Управления Информационной Безопасностью имеет две цели:

- выполнение требований безопасности, закрепленных в SLA и других требованиях внешних договоров, законодательных актов и установленных правил;
- обеспечение базового Уровня Безопасности, независимого от внешних требований.

Процесс Управления Информационной Безопасностью необходим для поддержки непрерывного функционирования ИТ-организации.

4.4. Автоматизация управления инфраструктурой IT

Появление на рынке программного обеспечения автоматизированных моделей управления инфраструктурой ИТ-организации – вполне естественное явление, направленное на развитие оптимизации как внутренних процессов ИТ-оранизаций, так и для развития бизнеса в целом. Рассмотрим некоторые из них.

В компании Hewlett-Packard разработала модель реализации управления ИТ-сервисами - HP ITSM Reference Model. В настоящий момент этот продукт называется OpenView Service Desk.

OpenView Service Desk включает набор функций: по поддержке службы обработки пользовательских обращений (Help Desk Manager), по управлениию инцидентами, управления проблемами, реализами, изменениями и средства управления уровнем обслуживания. Модуль Service Level Manager позволяет контролировать соблюдение соглашений об уровне обслуживания (Service Level Agreement, SLA), включая как внешние соглашения, так и внутренние регламенты OLA (Operating Level Agreement), задающие метрики обеспечения сервиса со стороны ИТ-инфраструктуры.

Разработанные и свободно распространяемые справочные руководства Microsoft Operations Framework (MOF) [21] и Microsoft Solutions Framework (MSF) также базируются на основе ITIL.

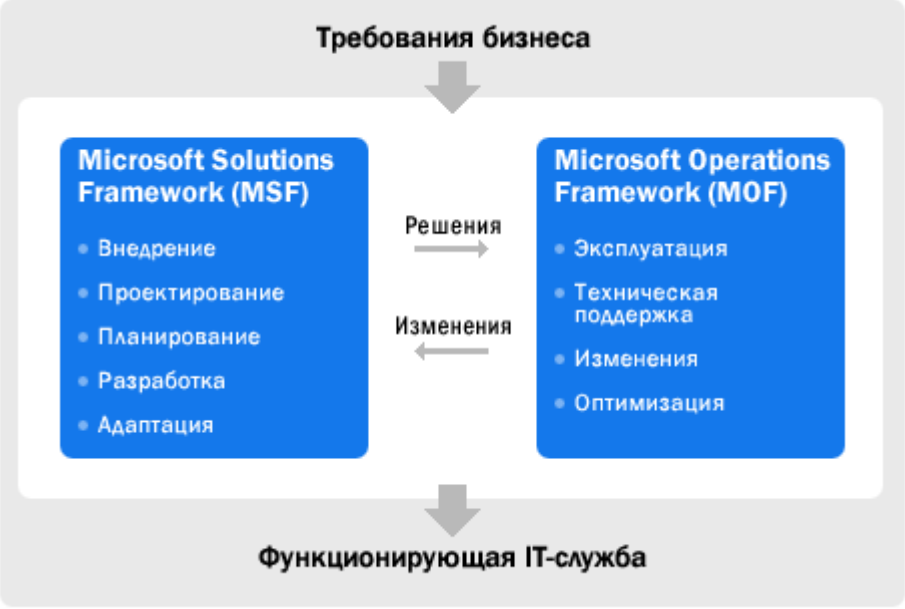

Рисунок 30. Схема взаимодействия MSF и MOF

MOF (Microsoft Operations Framework) состоит из технических руководств, помогающих компаниям достигнуть требуемых от информационной системы уровней надежности, доступности, простоты в технической поддержке и управляемости при построении решений на базе продуктов и технологий Microsoft. Рекомендации MOF касаются

вопросов персонала, процессов, технологий и стратегии управления в сложных, распределенных, гетерогенных IT-средах (www.microsoft.com/mof (EN)).

MSF (Microsoft Solutions Framework), включает в себя принципы, модели и знания для управления людьми, процессами, технологическими элементами. В MSF содержатся сведения о том, как эти ресурсы могут взаимно дополнять друг друга в рамках типичных проектов, а также примеры из практики планирования, дизайна системной архитектуры, разработки и внедрения IT-решений (www.microsoft.com/msf (EN)).

Для успешной работы служб, использующих IT-технологии, персонал технического отдела должен справлять с двумя ключевыми задачами:

1. Ясно понимать текущие потребности в сервисах и услугах и создавать адекватное решение.
2. Использовать решение для организации сервисов и оказания услуг.

Для решения первой задачи (анализ потребностей и создание удовлетворяющего их решения) предназначен MSF, в то время как MOF относится к второй задаче (использование решения в повседневной деятельности предприятия).

Обе методологические структуры дополняют друг друга, сокращая период материализации (промежуток времени от формулировки потребности до начала оказания услуги) и используют общую терминологию и набор концепций, что способствует созданию с их помощи высококачественных решений.

Создание и эксплуатация нового решения, согласно MOF и MSF, состоят из четырех базовых этапов:

1. Определить новые потребности бизнеса.
2. Построить и внедрить решение с помощью MSF и MOF.
3. Использовать решение, руководствуясь MOF.
4. Повторить цикл усовершенствования.

Необходимость во внесении изменений в уже внедренное решение может исходить из практической необходимости, вновь появившихся нужд бизнеса или же из требований административного порядка.

Модель жизненного цикла IT-решений, применяемая в компании Microsoft, комбинирует отдельные операции, осуществляемые IT-подразделением организации, таким образом, чтобы оказываемые им услуги были бесперебойными, а издержки — минимальными.

Структуры материалов руководств MOF и MSF нацелены на различные, но связанные между собой фазы жизненного цикла IT-решений. Каждая из этих структур содержит полезную и детальную информацию о кадрах, процессах и технологиях, необходимых для успешной деятельность в соответствующей области.

Высокая интеграция между собой продуктов компании Microsoft и применение системы Microsoft System Center (система работы с операционными базами Microsoft Operations Manager и Systems Management Server), позволяют разработчикам из ИТ-организаций создавать решения с готовой автоматизацией получения аналитической информации в едином формате по управлению поставляемых систем, и формировать отчеты и прогнозы для администраторов ИТ- и бизнес-подразделений. Продукт Microsoft Operations Manager позволяет получать информацию о событиях в ИТ-инфраструктуре. Configuration Manager предназначен для управления конфигурациями. Office 2007 используется для организации совместной работы и формирования шаблонов процессов, а среда разработки Visual Studio 2007 - для добавления новых конфигурационных единиц и описания потоков работ. Корпорация Microsoft продолжает Service Desk разработки не только процессов управления инцидентами и проблемами, но и управления изменениями и ресурсами.

Разработка компании IBM - Rational Unified Process (RUP) [22] представляет собой методологию коллективной разработки, которая предназначена для поддержки, как малых, так и крупных ИТ-организаций. RUP содержит описание процесса разработки, технологии и правил создания артефактов, полностью (по ролям и задачам) описывает виды деятельности каждого участника проекта на каждом этапе. Кроме того, RUP дает рекомендации по применению инструментальных средств поддержки всех процессов жизненного цикла программ, предлагая настраиваемые шаблоны, позволяющие использовать информацию проекта для создания отчетных документов.

В IBM Rational эту методологию снабдили набором средств, которые позволяют организации формировать свой вариант RUP путем адаптации. RUP предлагает не жесткие правила, регламентирующие выполнение всех действий в ходе разработки, а набор гибких методов и подходов, из которых можно выбирать то, что соответствует особенностям конкретного проекта.

4.5. Выводы

Изучая раздел “Хорошие практики управления качеством процесса разработки ПО. ITIL и ITSM”, мы пришли к выводам о том, что:

- Заказчиком IT услуг является бизнес и бизнес не может обойтись без этих услуг.
- Бизнес старается экономить на IT услугах или отводит IT сервисам второстепенные роли при решении бизнес-задач.
- Поставщиком IT-услуг являются IT-организации. IT-организации зависят от наличия заказов от бизнеса.
- IT-организации не участвуют в бизнесе заказчика.
- Вспомогательная роль ИТ-организации не позволяет ей участвовать в увеличении качества продукции заказчика, или её участие реализуется весьма опосредовано.
- Цикл взаимоотношений бизнеса и ИТ-организации не предназначен для непрерывного улучшения системы менеджмента качества бизнеса, и такой подход является устаревшим.
- Целью улучшения процессов в ИТ-организации является предоставление именно тех услуг, которые требуются в данный момент бизнесу, без потерь времени и без снижения производительности "по техническим причинам".
- Улучшение процессов в структуре ИТ позволяет бизнесу:
  - Совершенствовать использование ресурсов;
  - Быть более конкурентоспособным;
  - Сокращать долю переделок;
  - Устранять избыточные работы;
  - Улучшать время и качество результатов проектов;
  - Увеличивать доступность, надёжность и безопасность ИТ услуг, критичных для бизнеса;
  - Обосновывать стоимость качества предоставляемых услуг;
  - Предоставлять обслуживание, удовлетворяющее требованиям бизнеса, заказчиков и пользователей бизнес-подразделений;
  - Интегрировать централизованные процессы;
  - Документировать и распределять роли и обязанности внутри сервисных отделов;
  - Учиться на предыдущем опыте;
  - Предоставлять очевидные показатели производительности.
- Для разрешения противоречий во взаимоотношениях IT и бизнеса разработан мировой стандарт и руководство по развертыванию и сопровождению ИТ-сервисов - ITIL (IT Infrastructure Library).

4.6. Вопросы и задания для самостоятельной работы студента по теме «Хорошие практики управления качеством процесса разработки ПО. ITIL и ITSM»

1) Дайте определение стандарта ITIL
2) Сколько книг описывает стандарт ITIL?
3) Дать определение процессам Управления: «Доступности (Availability Management), Изменения, (Change Management), Информационная Безопасность (Security Management), Инцидентами (Incident Management), Конфигурациями (Configuration Management), Мощностями (Capacity Management), Непрерывностью ИТ-услуг (IT Service Continuity Management), Проблемами (Problems Management), Релизами (Release Management), Уровнем Услуг (Service Level Management), Финансами ИТ (Management for IT Services), Service Desk».
4) Указать их отношение к сервисам: “Поддержка услуг” (Service Support) или “Предоставлению услуг” (Service Delivery).
5) Какие модели автоматизации управления инфраструктурой IT-организаций Вам известны?

Литература по теме «Хорошие практики управления качеством процесса разработки ПО. ITIL и ITSM»

[1] Седов Олег. "Управление ITIL", Computerworld Россия, 07.2007 г.
[2] Книга Форума itSMF. "Введение в ИТ Сервис-менеджмент", под редакцией Потоцкого М.Ю. (перевод на рус. язык). Открытые Системы.
[3]http://www.microsoft.com/technet/solutionaccelerators/cits/mo/mof/default.mspx
[4] http://www-306.ibm.com/software/awdtools/rup/support/

Тема 5. Мероприятия, направленные на контроль и обеспечение качества артефактов проекта

Вырабатывайте в себе привычку все делать набело, ничего не откладывая на потом, и ваша профессиональная судьба сложится удачно.

Федор Александрович Новиков[19]

Рисунок 31. Э́дсгер Ви́бе Де́йкстра (Edsger Wybe Dijkstra) 11 мая 1930, Роттердам (Нидерланды) — 6 августа 2002) Выдающийся нидерландский учёный, идеи которого оказали огромное влияние на развитие компьютерной индустрии. Активно участвовал в разработке языка программирования Алгол и написал первый компилятор Алгол-60. Будучи одним из авторов концепции структурного программирования, он проповедовал отказ от использования инструкции GOTO. Ему принадлежит идея применения «семафоров» для синхронизации процессов в многозадачных системах и алгоритм нахождения кратчайшего пути на ориентированном графе с неотрицательными весами рёбер, известный как Алгоритм Дейкстры. В 1972 году Дейкстра стал лауреатом премии Тьюринга.

Программирование — это краеугольный камень процесса разработки программного продукта. Дейкстра делил все программы на три класса: плохие, хорошие и работающие. Если программа не прошла должного тестирования, она не является работающей. Таким образом, программирование и тестирование — это две фундаментальные основы, на которых зиждется все здание технологии создания программного обеспечения.

5.1. Правила хорошего стиля программирования

В настоящее время разработка программного обеспечения является интенсивно развивающейся отраслью промышленности. Время, когда программирование было уделом отдельных талантливых одиночек, закончилось уже лет 30 назад. Сегодня крупные программные проекты разрабатываются большими коллективами, причем развитие и сопровождение проекта часто длятся годами. За это время успевает обновиться коллектив разработчиков, нередки случаи, когда инициаторы проекта к моменту его завершения могут отсутствовать не только в фирме, но и в стране.

В этих условиях естественным является требование унификации стиля программирования. Код, написанный

[19] Федор Александрович Новиков - автор ряда книг по программному обеспечению. Автор курса “Анализ и проектирование на UML”, подготовленного при поддержке Sun Microsystems.

различными программистами, должен читаться как единое целое и быть понятен не только его авторам. Отсюда следует, что программа должна быть рационально написана и тексты программы должны содержать достаточно комментариев для понимания сути алгоритмов другими программистами. Отметим, ничто так не раздражает программиста–профессионала, как небрежно написанный и плохо документированный (включая комментарии) код. Так же как невозможно представить инженерный чертеж, оформленный без учета требований стандартов, так и программа должна иметь некий унифицированный вид. Целью данной главы является набор рекомендаций, следуя которым можно представить свой код как документ, имеющий общую форму и доступный для изучения, а также для дальнейшего сопровождения и развития. Приведенные здесь правила и рекомендации не являются обязательным стандартом, вместе с тем, их применение желательно.

Существует несколько стилей написания и оформления программ, одгим изкоторых является стиль Microsoft (стиль MFC). Поскольку большинство профессиональных программистов, пишущих программы для MS Windows, разрабатывает свой код средствами Microsoft Visual Studio, рекомендуется применять этот стиль в качестве образца и придерживаться его при выполнении программных проектов.

Часто приходится сталкиваться с ситуацией, когда программа сначала пишется и отлаживается и только потом документируется. Причем, всем хорошо известно, что такая практика неверна, и вот почему:

- прежде чем начинать кодировать, необходимо продумать структуру программы и зафиксировать ее в документации;
- на завершающем этапе разработки времени всегда не хватает, в результате комментированием программы либо вообще пренебрегают, либо делают это небрежно, лишь бы “отвязаться”.

Комментирование программы должно происходить одновременно с написанием кода. Технически процесс вставки «шапок» перед программами и классами может быть реализован с помощью макросов. В этом случае комментирование выполняется буквально «в два щелчка» мыши.

Приведенные в данной главе рекомендации являются общепринятыми. В некоторых организациях документы аналогичного содержания доводятся до сведения каждого сотрудника и являются своего рода внутренним стандартом фирмы. Однако, даже если в организации отсутствует такой документ, фиксирующий требования к стилю программирования, все равно такие требования имеют место, хотя и носят более размытый, интуитивный характер. В любом случае плохой стиль сразу заметен и корректируется административными мерами.

5.2. Стандартизация при написании кода программного обеспечения

Стандартизация становится неизбежной, когда процесс разработки программного обеспечения принимает промышленный характер. То есть:

- когда в работе над проектом участвует достаточно многочисленный коллектив разработчиков;
- когда проект состоит из нескольких частей проекта, и отдельные исполнители не знают детально, чем, собственно, они занимаются. Каждый делает свою часть проекта. Проект целиком представляет себе руководитель проекта и несколько ведущих специалистов.;
- когда в процессе работы происходит ротация персонала, кто-то увольняется, появляются новые сотрудники, которых нужно быстро ввести в курс дела;
- олжны быть некие общие правила поведения, без которых невозможна слаженная работа коллектива разработчиков и их взаимодействие с заказчиками.

Одним словом, если вы участвуете в разработке достаточно большого и длительного проекта, вам придется следовать правилам работы того коллектива, в котором вы работаете. Чем более «оригинально» вы оформите свой код, тем меньше окажется желающих в нем разобраться.

Наконец, если вы пишете программу «для себя», вам неизбежно придется неоднократно возвращаться к ней для устранения ошибок или развития программы. Нередки случаи, когда, заглянув в свой проект через пару недель, программист уже не помнит, с какой целью он написал тот или иной фрагмент кода. Такого не случится, если программа будет прокомментирована.

5.3. Имена

Имя – результат размышлений, в основе которых лежит понимание работы системы в целом. Имя должно соответствовать тому свойству или процессу, который оно обозначает.

- Имена должны быть осмысленными и понятными;
- Имена должны быть построены на основе английских слов;
- Избегайте использования имен переменных и функций, составленных целиком из больших букв;
- Имена констант следует задавать только большими буквами.

5.3.1. Имена классов

- Имя класса должно соответствовать тому объекту, который описывает данный класс.

  Пример:

```
class CInterpolator           // интерполятор
class CSummer                 // сумматор
class CConicInterpolator      // конический интерполятор
class ClinearInterpolator // линейный интерполятор
```

- Имя класса должно начинаться с буквы “C”, что означает “class”. Это традиция фирмы Microsoft, другие фирмы могут использовать другую литеру. Например, фирма Borland использует букву “T”.
- Если имя класса состоит из нескольких слов, каждое слово должно начинаться с заглавной буквы.

  Пример:

```
class CConnectionPointForMyOcx;
```

- Имена, объединяющие в себе более трех слов, использовать не рекомендуется. Длинные идентификаторы затрудняют чтение программы.

  Пример:

```
class CSemaphorForMyNewPort;     //слишком длинно и сложно
class CNewPortSemaphor;          //  уже лучше
```

- Имена производных классов не должны содержать в себе имен родительских классов, каждый класс должен иметь собственное имя, кем бы он ни порождался.
- Имя класса должно начинаться с заглавной буквы. Если имя класса состоит из нескольких слов, каждое слово должно начинаться с заглавной буквы, которая служит разделителем слов.

5.3.2. Имена библиотек классов

- Чтобы избежать конфликта имен с другими библиотеками классов, используйте для своей библиотеки некоторый уникальный префикс.
- Длина префикса не должна превышать трех символов.
- Классы, производные от стандартных C++ библиотек, должны иметь тот же префикс, который используется этими библиотеками. Например, префикс “C” используется библиотекой MFC (`class CWnd`), префикс “T” используется библиотекой OWL (`class TWindow`).
- Для классов и функций, разрабатываемых в данной фирме, в качестве префикса можно использовать сокращенное имя фирмы.

  Пример:

```
class ArcNewPortSemaphor    // Arc – от "ARCADIA"
class NitOldPortSemaphor         // Nit – от "NITA"
```

5.3.3. Имена методов класса

- Используйте те же правила, что и для имен классов.
- Имя не должно напоминать ругательство. Длинное имя обычно лучше, чем короткая аббревиатура, однако не следует злоупотреблять.
- Обычно каждый метод и функция выполняет некоторое действие, поэтому имя должно описывать это действие:

```
CheckForErrors()  вместоErrorCheck(),
DumpDataToFile()  вместо     DataFile().
```

Следующие суффиксы иногда могут быть полезны:

- *Max* - означает наибольшее значение чего либо.
- *Cnt* - означает текущее количество чего либо.
- *Key* – ключевое значение.

Пример: *RetryMax* – означает максимальное число попыток.
RetryCnt – означает номер текущей попытки.

Следующие префиксы иногда могут быть полезны:

- *Is* – используется, чтобы задать некоторый вопрос:
  IsEnable(), IsWindow(), IsActive();
- *Get* – используется для считывания некоторого значения:
  GetData(), GetWndRect();
- *Set* – используется для задания нового значения:
  SetData(), SetWindowText();
- *On* – используется при задании обработчика событий:
  OnCreate(), OnClose().

Пример:

```
class CNameOneTwo
{
   public:
     int   DoIt();
     void  HandleError();
     BOOL  IsItPrintable();
     long  OnWmCreate(WPARAM wParam1, LPARAM lParam2);
     const char* GetFirstName() const;
     void  SetSecondName(const char* pszSecName);
};
```

5.3.4. Имена функций

- Используйте те же правила, что и для имен методов классов.

Пример:

```
BOOL   AddChildWindow(HWND hChildWnd);
socket GetListenSocket(unsigned short uListenPort);
BOOL   IsTransactionValid(TRANS_ID trid);
```

5.3.5. Имена аргументов функций и методов класса

- Используйте те же правила, что и для имен классов.
- Используйте упрощенную Венгерскую нотацию.

Пример:

```
class CTest
{
   public:
         int  RunTest(int nParam1, long lParam2);
};
```

5.3.6. Имена переменных

- Используйте упрощенную Венгерскую нотацию.

Prefix	Type	Description	Example
Basic Types			
ch, c	**Char**	8-bit character	*ChGrade*
sz, str	**char[]**	8-bit string	*szBuffer[]*
s	**Short**	16-bit signed integer	*sIndex*
n, l	**Int**	Integer (size dependent on operating system)	*nLength*
u	**unsigned**	Unsigned integer (size dependent on operating system)	*uSize*
l	**Long**	32-bit signed integer	*lSum*
ul	**usigned long**	32-bit unsigned integer	*ulFreeSpace*
flt	**Float**	Floating point	*fltResult*
dbl	**Double**		*DblResult*
ldbl	**long double**		*ldblResult*
Aliased Types			
by	**BYTE**	Byte (unsigned char)	*byLower*
b	**BOOL**	Boolean value	*bEnabled*
n, u	**UINT**	Unsigned value (size dependent on operating system)	*nLength*
w	**WORD**	16-bit unsigned value	*wPos*
l	**LONG**	32-bit signed integer	*lOffset*
dw	**DWORD**	32-bit unsigned integer	*dwRange*
h	**Handle**	Handle to Windows object	*hWnd*
Pointers			
p	**void***	Ambient memory model pointer	*pDoc*
pp	**void****	Pointer to pointer	*ppDoc*
lp	**FAR***	Far pointer	*lpDoc*
np	**NEAR***	Near pointer	*npDoc*
lpsz	**LPSTR**	32-bit pointer to character string	*lpszName*
lpfn	**callback**	Far pointer to CALLBACK function	*lpfnAbort*
p(type-prefix)	**basic_type***	Pointer to basic type variable	
Derived Types			
pt	**POINT**	Points	*ptCurPos*

rc	**RECT**	Rectangle	*rcViewPort*
Obj	**OBJECT**	Object	*objOld*
Arr	**ARRAY**	Array	*arrItems*

5.3.7. Имена свойств класса

- Используйте упрощенную Венгерскую нотацию.
- Имена свойств должны иметь префикс `'m_'`.
- После `'m_'` следует имя свойства, имеющее ту же структуру, что и имя переменных.
- `'m_'` всегда должно предшествовать любым другим модификаторам, например, 'p' для указателей.

Пример:

```
class CNameOneTwo
{
  public:
    int   VarAbc();
    int   ErrorNumber();

  private:
    int   m_nVarAbc;
    int   m_nErrorNumber;
    char* m_pszName;
};
```

5.3.8. Имена указателей

- Имени указателя должен предшествовать модификатор ‘p’.
- Символ ‘*’ следует помещать сразу после типа указателя, а не перед именем указателя.

Пример:

```
class CTest
{
  public:
    void   PrintString(char* pszString);

  private:
    char*  m_pszClassName;
};
```

5.3.9. Имена ссылок

Ссылке может предшествовать символ “r”, это позволит различать модифицируемые и немодифицируемые объекты.

Пример:

```
class CTest
{
  public:
```

```
    void   DoSomething(StatusInfo& rStatus);
    const StatusInfo&  GetStatus() const;

  private:
    StatusInfo& m_rStatus;
};
```

5.3.10. Имена глобальных переменных

- Используйте упрощенную Венгерскую нотацию.
- Глобальной переменной должен предшествовать префикс “g_”.

Пример:

```
long    g_lCounter;
char*   g_pszAppName;
```

5.3.11. Имена статических переменных

- Используйте упрощенную Венгерскую нотацию.
- Статической переменной должен предшествовать префикс “s_”.

Пример:

```
class CTest
{
  private:
        static StatusInfo ms_Status;
};

static int    s_nReadBytes;
static char*  s_pClassName;
```

5.3.12. Имена констант

- Константы в C++ можно задать тремя операторами: `const`, `#define`, `enum`.
- Константы должны состоять из заглавных букв и сепараторов “_”.
- Имена констант не должны совпадать, независимо от того, каким способом вы задали константы.

Пример:

```
#define GLOBAL_CONSTANT     5
const int nLocalConst     = 4;
```

5.3.13. Имена макроопределений #define

- Константы, определяемые оператором `#define`, должны состоять из заглавных букв и сепараторов “_”.
- Рекомендуется использовать префиксы, чтобы сгруппировать константы в категории.

  Пример:

```
#define MAX(a,b)          blah
#define IS_ERR(err)       blah

#define ERRMSG_NO_RAM     0
#define ERRMSG_WRITE      1
#define ERRMSG_READ       2
```

5.3.14. Имена констант const

- Имена констант в перечислении задаются или только заглавными буквами, или только строчными буквами (с использованием сепаратора ‘_’). Смешение стилей не допускается. Предпочтительным является использование строчных букв, чтобы не смешивать константы с макросами.

  Пример:

```
const int default_lval=0;
```

5.3.15. Имена перечислений enum

- Некоторые программисты используют `enum` вне класса для задания константы, в этом случае следует убедиться, что имя данной константы не совпадает с другой константой, заданной операторами `const` или `#define`.

Пример:

```
enum PinStateType
{
  pin_off=0,
  pin_on
};
```

- Имена констант в перечислении задаются или только заглавными буквами, или только строчными буквами (с использованием сепаратора ‘_’). Смешение стилей не допускается. Предпочтительным является использование строчных букв для того, чтобы не смешивать константы с макросами.
- Если `enum` задается при определении класса, то для чтения константы необходимо указать имя класса, в котором задана `enum`:

```
  Aclass::pin_off.
```

- `enum` можно использовать для задания неинициализированных или ошибочных состояний.

  Пример:

```
Enum {state_err, state_open, state_running, state_dying};
```

5.3.16. Имена типов данных

Имена типов данных, созданных при помощи `typedef`, следует писать только строчными буквами (с использованием сепаратора “_”) для отличия от макросов, созданных директивой `#define`. Это отличие весьма важно, поскольку, например, объявление

```
typedef void *(ptr_to_func) ( int );
```

позволяет писать как

```
(ptr_to_func) (p);        // convert p into ptr_to_func
```

так и

```
ptr_to_func f( long );    // f returns a pointer to a
                          //   function
```

В то же время

```
#define PTR_TO_FUNCTION (void ( * ) ( int ))
```

позволяет привести тип

```
(PTR_TO_FUNCTION)(p);
```

но не определить функцию.

Пример:

```
typedef unsigned int   module_id;
typedef unsigned long  system_type;
```

5.3.17. Имена меток

- Использования оператора `goto Label;` следует избегать, когда это возможно.
- Имена меток следует задавать строчными или прописными буквами с использованием сепаратора “_”. Смешивание стилей недопустимо.

Пример:

```
FIRST_JUMP:
goto restart_point;
```

5.3.18. Имена файлов

- Файлы программ для языков C/C++ должны иметь расширения “`.c`”/”`.cpp`”, соответственно. Заголовочные файлы для языков C/C++ должны иметь расширения “`.h`”. Файлы, содержащие `inline`-определения, должны иметь расширения “`.inl`”. Файлы ресурсов для Windows - проектов должны иметь расширения “`.rc`”. `Include` – файлы, содержащие идентификаторы для ресурсов, могут иметь расширения “`.h`”, “`.rh`”, “`.hm`”.
- Используйте для имен и расширений файлов соглашение 8.3 (8 символов – имя, и 3 символа - расширение), когда это возможно. Некоторые операционные системы и архиваторы все еще не поддерживают длинные имена; используя соглашения 8.3, вы избежите ненужных конфликтов.

- Если Вы используете Wizard для создания своего проекта, не изменяйте без крайней необходимости имена файлов, которые он предложит.
- Всегда старайтесь выбрать уникальное имя для своих файлов, насколько это возможно.
- Старайтесь включать имя класса в имя файла, в котором он определяется.

5.4. Форматирование

5.4.1. Общие требования

- На одной строке должно находиться не более одного оператора C/C++.
- Если вызов функции, операция инициализации, список и т.п. занимают более одной строки, перенос на следующую строку должен следовать сразу после запятой.
- Если выражение занимает несколько строк, переход на следующую строку должен выполняться сразу после бинарной операции.

5.4.2. Максимальная длина строки

- Максимальная длина строки не должна превышать 70 символов. Хотя большие мониторы могут отображать и более длинные строки, печатающие устройства более ограничены в своих возможностях.
- Чем больше размеры окна, тем меньше окон можно одновременно наблюдать на экране дисплея. Принимая во внимания требование удобства отладки, мы утверждаем, что возможность одновременно иметь несколько открытых окон важнее, чем возможность иметь одно, но большое окно. Из этого соображения также следует ограничение на длину строки.

5.4.3. Использование для организации отступов табуляции и пробелов

- Используйте для организации отступов табуляцию вместо пробелов.
- На размер отступа не накладывается ограничений, но злоупотреблять отступами не стоит. Если размер отступа более 4 или 5 пробелов, ваш код может не уместиться по ширине страницы. Большинство редакторов позволяет задавать размер отступа табуляции.
- Комментарии должны находиться на уровне того оператора,

которому они соответствуют.

- Комментарии справа от операторов должны быть выровнены с помощью пробелов, а не табуляции.

Пример:

```
int Func(int nParam1, long lParam2)
{
   int    nVariable;          // in-line comment
   char   sBuffer[10];        // in-line comment

   // comment
   if (something_bad)
   {
        if (another_thing bad)
        {
             while (more_input)
             {
             }
        }
   }

   // comment
   for (int i = 0; i < 10; i++)
        nVariable += i;

   return nVariable;
}
```

5.4.4. Пунктуация

- Точке с запятой ';' не должны предшествовать пробелы или табуляция.
- После запятой должен следовать пробел.
- Правильное совместное использование операторов и пробелов проиллюстрировано в следующей таблице.

Operator	L	R	Comments	Example
!	.	-	Not	`if (!TRUE)`
!=	+	+	Arithmetic inequality	`if (a != b)`
#	.	-	Preprocessor directive	`#define YES 1`
##	+	+	Token-paste	`#define cat(x, y) x ## y`
%	+	+	Modulus	`a = b % c;`
%=	+	+	Modulus and assign	`a %= b;`
&	+	+	Bitwise AND	`if (mask & attrib)`
&	.	-	Address of an operand	`a = &b;`
&&	+	+	Logical AND	`while (a && b)`
&=	+	+	Bitwise AND assign	`a &= mask;`
()	.	-	Cast operators	`a = (INT)b;`
*****	+	+	Multiply	`a = b * c;`
*****	.	-	Indirection operator	`a = *b;`
***=**	+	+	Multiply and assign result	`a *= b;`
+	+	+	Add	`a = b + c;`
++	.	-	Increment prefix	`++a;`
++	-	.	Increment postfix	`a++;`
+=	+	+	Add assign result	`a += b;`
,	-	+	Evaluate an rvalue	`a = b, b = c, c = a;`
-	+	+	Subtract	`a = b - c;`
--	.	-	Decrement prefix	`--a;`
--	-	.	Decrement prefix	`a--;`
-=	+	+	Subtract and assign	`a -= b;`
->	-	-	Struct/union pointer offset	`a = b->c;`
.	-	-	Select struct/union member	`a.b = 2;`
/	+	+	Divide	`a = b / c;`
/=	+	+	Divide and assign	`a /= b;`
<	+	+	Less then	`if (a < b)`
<<	+	+	Left shift	`a = b << 2;`
<<=	+	+	Left shift and assign	`a <<= 4;`
<=	+	+	Less than or equal	`if (a <= b)`
=	+	+	Assignment operator	`a = b;`
==	+	+	Equality	`if (a == b)`
>	+	+	Greater than	`if (a > b)`
>=	+	+	Greater than or equal	`if (a >= b)`
>>	+	+	Right shift	`a = b >> 2;`
>>=	+	+	Right shift and assign	`a >>= 4;`
?	+	+	if/else (cf. :)	`a = (b > c) ? b : c;`
:	+	+	if/else (cf. ?)	`a = (b > c) ? b : c;`
[	-	-	Array subscript (cf.])	`a = s[i];`
]	-	.	Array subscript (cf. [)	`a = s[i];`
^	+	+	Exclusive OR	`a = b ^ c;`
^=	+	+	Exclusive OR and assign	`a ^= b;`
sizeof	.	-	Size in bytes	`a = sizeof(INT);`

|	+	+	Bitwise OR	`a = b | c;`
|=	+	+	Bitwise OR and assign	`a |= b;`
||	+	+	Logical OR	`if (a || b)`
~	.	-	One's complement	`a = ~b;`

Здесь:

- L: левый пробел;
- R: правый пробел;
- -: пробел недопустим;
- +: пробел обязателен;
- (.) означает, что пробел не обязателен, но допустим.

Замечание: Оператор ‘##’ использовать не рекомендуется, поскольку не все компиляторы поддерживают этот оператор.

5.4.5. Выравнивание блока объявлений

- Блок объявлений должен быть выровнен по типу, имени, начальному значению и комментарию.
- Для выравнивания используйте пробелы, а не табуляции.
- Символы ‘&’ и ‘*’ должны следовать без пробелов за объявлением типа, а не имени.
- Целью выравнивания является улучшение ясности и читаемости кода.

Пример:

```
void Func()
{
   DWORD       dwLength;       // in-line comment
   DWORD*      pdwLength;      // in-line comment
   char        szName[12];     // in-line comment
   char*       pszName;        // in-line comment

   dwLength  = 0;
   pdwLength = NULL;
   szName[0] = '\0';
   pszName   = NULL;

        ...
}
```

5.4.6. Расстановка фигурных скобок

Существует несколько традиций расстановки скобок, наиболее употребительными являются следующие:

- традиция фирмы Microsoft:

```
If(condition)      While(condition)
{                  {
     ...                ...
}                  }
```

- традиция Unix:

```
if(condition){     While(condition){
     ...                ...
}                  }
```

Традиция Microsoft более предпочтительна, поскольку иногда бывает полезно точно знать, где находится граница блока.

Пример:

```
if (very_long_condition && second_very_long_condition)
{
     ...
}
else if (...)
{
     ...
}
```

5.4.7. Расстановка круглых скобок

- Ключевое слово, заключенное в круглые скобки, должно начинаться и заканчиваться пробелом. Исключением является оператор `sizeof()`.
- Недопустимо ставить пробелы сразу после имени функции.
- Не применяйте скобки в операторе `return` без необходимости.
- Не используйте пробелы перед текстом, заключенным в кавычки, а также после него.
- Каждое выражение, за исключением арифметических операций, должно использовать скобки, чтобы явно задать порядок выполнения операций. В арифметических выражениях также лучше всегда использовать скобки, поскольку разные компиляторы по разному выполняют разбор арифметических выражений. Такого не должно быть, но это имеет место - это экспериментальный факт.

  Пример:

```
int Func(char* pszParam1, char* pszParam2)
{
   int nAddrSize = sizeof(char*);
   if (pszParam1 == NULL) return 0;
   if (pszParam2 == NULL) return 0;
   strcpy(pszParam1, pszParam2);
   return 1;
}

int GetMaxValue(int nValue1, int nValue2)
{
   return ((nValue1 >= nValue2) ? nValue1 : nValue2);
}
```

5.4.8. Форматирование операторов if-else

- При сравнении с константой ее лучше размещать справа от оператора “==” или “!=”.

```
if (nErrorNum == 6)
     ...
```

- Выравнивать `if/else` следует так, как это показано в приведенном ниже примере.

Пример:

```
if (condition)                  // in-line comment
{
   ...
}
else if (condition)             // in-line comment
{
   ...
}
else                            // in-line comment
{
   ...
}
```

5.4.9. Форматирование операторов switch-case

- Если оператор `break` не заканчивает блок команд, следующих после оператора `case`, отметьте в комментарии, что это не случайный факт, а ваше сознательное решение.
- Если на вход оператора `switch` должны попадать только данные, перечисленные в `case`-ловушках, то оператор `default` должен обязательно присутствовать и генерировать сообщение об ошибке.

Пример:

```
switch (...)
{
```

```
    case 1:                    // in-line comment
         ...
    // fall-through comment must be here
    case 2:                   // in-line comment
         ...
         break;

    case 3:                   // in-line comment
    {
         int v;
              ...
    }
    break;

    default:
         break;
}
```

5.4.10. Комментарии

- Строка комментария должна иметь такой же отступ, как и операция, которую она комментирует.
- In-line комментарии должны размещаться справа от комментируемого кода.
- Операторы комментария “/*” и “//” должны отделяться от текста комментария по крайней мере одним пробелом.
- In-line комментарии должны быть выровнены использованием пробелов, а не табуляций.

Пример:

```
POINT ptCurrentMenuArea;   // in-line comment

// check if designated point is in the menu area
if (!IsMenuArea(ptCurrentMenuArea))
{
   // out of the menu area
   return NO_MENU_ID;
}
```

5.4.11. Методы и функции

- Формат задания формальных параметров функции зависит от количества этих параметров. Если список параметров помещается на одной строке, то функция записывается в одну строку, в противном случае каждый параметр следует писать на новой строке, при этом полезно применять in-line комментирование.

Пример: Функция с коротким списком формальных параметров:

```
int Func(int nParam1, long lParam2, char* pszParam3)
{
```

```
    ...
}
```

Функция с длинным списком формальных параметров:

```
int Func
(
    int   nParam1, // Integer parameter
    long  lParam2, // Long parameter
    char* pszParam3// String parameter
)
{
    ...
}
```

5.5. Документирование исходного кода

В этом разделе описываются меры, способствующие повышению читабельности кода и облегчающие его сопровождение в течение всего процесса разработки проекта.

5.5.1. Общие требования

- Документация к исходному коду, содержащаяся в комментариях, должна быть достаточной для полного понимания кода и его дальнейшего сопровождения как для разработчиков (проектировщиков) системы, так и для программистов. Следует иметь в виду, что недокументированный код *не имеет смысла*, а следовательно, и права на существование.
- Каждый файл исходного кода программы должен иметь вводную часть, содержащую в форме комментария информацию об авторе программы, имени файла и его содержании.
- Исходный код должен включать в себя заявление о защите авторских прав. Если программа разрабатывается в течение нескольких лет, указывается каждый год.
- Все комментарии рекомендуется писать на английском языке. Программистов, не владеющих английским, в природе не существует, а вот компиляторы, не поддерживающие кириллицу, к сожалению, встречаются часто.
- Исходный код необходимо документировать. Следует правильно выбирать имена переменных, функций и классов. Необходимо структурировать код, такой код требует меньше комментирования.
- Имейте в виду, что комментарии в заголовочных h-файлах предназначены для пользователей классов, тогда как комментарии в CPP-файлах реализации предназначены для разработчиков этих классов.
- Комментарии подразделяются на стратегические и тактические. В стратегических комментариях описывается общее назначение

функций или фрагментов кода, и они вставляются в текст программы блоком из нескольких комментарных строк. Тактический комментарий задается одной строкой и описывает операцию на следующей строке. Его также можно размещать справа от комментируемой операции. Слишком много тактических комментариев делает код программы нечитабельным, по этой причине рекомендуется применять стратегическое комментирование. Общее представление важнее деталей реализации кода.

5.5.2. Исходные файлы (CPP-files)

- Каждый исходный файл должен содержать реализацию только одного класса или группы функций, близких по своему назначению.
- Каждый файл должен иметь заголовок, а информация должна быть единообразным образом структурирована.
- Каждый `.cpp`-файл должен включать в себя соответствующие заголовочные файлы (`.h`-files), которые должны содержать:
  - объявление типов и функций, которые используются функциями или методами класса, реализуемого в данном `.cpp`-файле;
  - объявление типов, переменных и методов классов, которые реализуются в данном `.cpp`-файле.

5.5.3. Заголовок исходного файла

Заголовок исходного `CPP`-файла представляет собой закомментированный текст, помещенный в начало файла. Этот текст имеет следующую структуру.

```
/************************************************************\
  FILE…….....: filename.ext
  AUTHOR…….....: Name(s) of the programmer(s)
  DESCRIPTION…: The description of the file.
  CLASSES……...: List of classes implemented in this module.
  FUNCTIONS……..: List of functions implemented in this module.
  SWITCHES……...: Preprocessor switches and their meaning.
  NOTES…….....: Other information supposed to be useful, e.g.,
                       compilers,
                             portability,
                                   updates
                                         and so on.
  COPYRIGHT……..: Copyright information.
  HISTORY…….....: DATE     COMMENT
                   -------- ---------------------------------------
                   mm-dd-yy Comments – Author.
\************************************************************/
```

Поля `FILE`, `AUTHOR`, `DESCRIPTION`, `COPYRIGHT` и `HISTORY` являются обязательными для заполнения. Заполнение остальных полей остается на усмотрение программиста.

5.5.4. Структура исходного файла

- Каждый исходный файл в проекте должен иметь следующую структуру.

```
/*************************************************************\
  FILE…….....: filename.ext
  AUTHOR…….....: Author’s name.
  COPYRIGHT…...: Copyright information.
  DESCRIPTION…: Description of the source file.
  HISTORY……..: DATE     COMMENT
                -------- -----------------------------------
                mm-dd-yy Comments – Author.
\*************************************************************/

/*============================[ SPECIAL ]==================*/
/*============================[ IMPORT DECLARATIONS ]=======*/
/*============================[ PUBLIC DECLARATIONS ]=======*/
/*============================[ PRIVATE DECLARATIONS ]======*/
/*============================[ ..PRIVATE CONSTANTS ]=======*/
/*============================[ ..PRIVATE TYPES ]===========*/
/*============================[ ..PRIVATE VARIABLES ]=======*/
/*============================[ ..PRIVATE FUNCTIONS ]=======*/
/*============================[ ..PRIVATE PSEUDO FUNCTIONS ]*/
/*============================[ PUBLIC  VARIABLES ]=========*/
/*============================[ PRIVATE VARIABLES ]=========*/
/*============================[ CLASS LIFECYCLE ]===========*/
/*----------------------------[ CLASS_NAME1 ]---------------*/

/*----------------------------[ CLASS_NAME2 ]---------------*/

/*============================[ CLASS OPERATORS ]===========*/

/*----------------------------[ CLASS_NAME1 ]---------------*/

/*----------------------------[ CLASS_NAME2 ]---------------*/

/*============================[ PUBLIC CLASS METHODS ]======*/

/*----------------------------[ CLASS_NAME1 ]---------------*/

/*----------------------------[ CLASS_NAME2 ]---------------*/

/*============================[ PROTECTED CLASS METHODS ]===*/

/*----------------------------[ CLASS_NAME1 ]---------------*/

/*----------------------------[ CLASS_NAME2 ]---------------*/

/*============================[ PRIVATE CLASS METHODS ]=====*/
```

```
/*---------------------------[ CLASS_NAME1 ]---------------*/

/*---------------------------[ CLASS_NAME2 ]---------------*/

/*===========================[ PUBLIC  FUNCTIONS ]=========*/

/*===========================[ PRIVATE FUNCTIONS ]=========*/

/** (END OF FILE  : filename.ext) **************************/
```

Приведем ниже описание каждого раздела.

Section	Description
SPECIAL	В этом разделе должны находиться блоки условной компиляции (#ifndef … и т.п.).
IMPORT DECLARATIONS	Все системные, библиотечные и заголовочные файлы (h-файлы) должны находиться в данном разделе.
PUBLIC DECLARATIONS	Эта строка должна предшествовать объявлением *public* методов и данных класса.
PRIVATE DECLARATIONS	Эта строка должна предшествовать объявлением *private* методов и данных класса.
..PRIVATE CONSTANTS	Все частные #defines и constants должны быть объявлены в данном разделе.
..PRIVATE TYPES	Все частные типы, которые используются в данном исходном файле, должны быть объявлены в данном разделе.
..PRIVATE VARIABLES	В этом разделе объявляются все частные переменные.
..PRIVATE FUNCTIONS	Все частные функции, используемые в данном исходном файле, должны быть объявлены в этом разделе.
..PRIVATE PSEUDO FUNCTIONS	Все макросы должны быть объявлены в этом разделе.
PUBLIC VARIABLES	Все глобальные переменные должны быть заданы в этом разделе.
PRIVATE VARIABLES	Этот раздел должен содержать все частные переменные данного исходного файла.
CLASS LIFECYCLE	Раздел жизненного цикла предназначен для размещения объектов, которые управляют жизненным циклом объекта. Обычно это конструкторы, деструкторы и методы, изменяющие состояние объекта.
CLASS_NAME	Если исходный файл содержит реализацию нескольких классов, то их методы должны быть разделены данной секцией с именем класса в квадратных скобках.
CLASS OPERATORS	Все оператора класса должны находится в этом разделе.
PUBLIC CLASS METHODS	Все *public* методы класса должны находится в этом разделе.
PROTECTED CLASS METHODS	Все *protected* методы класса должны находится в этом разделе.
PRIVATE CLASS METHODS	Все *private* методы класса должны находится в этом разделе.
PUBLIC FUNCTIONS	Раздел для реализации *public* функций.
PRIVATE FUNCTIONS	Раздел для реализации *private* функций.

- Если некоторый раздел не содержит информации, его можно опустить, но порядок разделов менять не разрешается.
- Наличие комментария, обозначающего начало нового раздела и содержащего имя этого раздела, не обязательно, но весьма желательно. Дело в том, что имеется целый ряд утилит для автоматической генерации документации по откомментированному в соответствии с формальными правилами тексту программы (например, Cweb, описанная в книге Donald. E. Knuth, S. Levy. The Cweb system of structured documentation. Reading: Addison-Wesley, 1994). Такие утилиты очень помогают, в особенности, тем, кто, не являясь разработчиком, должен модифицировать программу.
- В любом случае обязательными являются строки комментария, обозначающими конец и начало файла. Их отсутствие должно сигнализировать о том, что «что-то не в порядке» (например, в результате ошибки записи часть файла оказалась потеряна).

5.5.5. Заголовочные файлы (H-files)

- Каждый класс должен быть задан в своем .h-файле.
- Следующие объекты должны задаваться в своих отдельных .h-файлах:
  - базовые классы;
  - объекты (классы), которые являются данными других классов;
  - классы, которые передаются в качестве формальных параметров, функций или методов класса или возвращаются оператором `return`;
  - прототипы функций или методы классов, реализованные, как `inline`.
- Описания классов, доступных только через указатели (`*`) или ссылки (`&`), не должны включаться из `.h`-файлов. Используйте опережающие ссылки.
- Чтобы избежать многократного включения заголовочных файлов, содержимое каждого `.h`-файла должно находиться между следующими операторами условной компиляции:

```
#ifdef   FILENAME_H
#define  FILENAME_H

/* contents of FILENAME.H */

#endif /*FILENAME_H */
```

- Никогда не используйте абсолютные имена путей, например:

```
"#include c:\project\sources\include\project.h".
```

Используйте относительные пути или задавайте путь в опциях компилятора (в последнем случае необходимость задания путей в опциях должна быть задокументирована, в том числе и в заголовке программы).

- Никогда не включайте при помощи `#include` файлы исходного кода (`.cpp`). Включать надо только заголовочные файлы (`.h`).
- Включите в начало `h`-файла следующий фрагмент текста.

```
/*************************************************************\
  FILE…….....: filename.ext
  AUTHOR…..…..: Name(s) of the programmer(s)
  DESCRIPTION…: The description of the file.
  CLASSES…....: List of classes declared in this module.
  FUNCTIONS……: List of functions declared in this module.
  SWITCHES……..: Preprocessor switches and their meaning.
  NOTES…….....: Other information supposed to be useful, e.g.,
                      compilers, portability, updates and so on.
  COPYRIGHT……: Copyright information.
  HISTORY…..…..: DATE     COMMENT
                 -------- -----------------------------------
                 mm-dd-yy Comments – Author.
\*************************************************************/
```

Поля `FILE, AUTHOR, DESCRIPTION, COPYRIGHT, HISTORY` подлежат обязательному заполнению, остальные – на ваше усмотрение.

Каждый h-файл должен иметь следующую структуру.

```
/*************************************************************\
  FILE…….....: filename.ext
  AUTHOR…..…..: Author’s name.
  COPYRIGHT……: Copyright information.
  DESCRIPTION…: Description of the source file.
  HISTORY…..…..: DATE     COMMENT
                 -------- ---------------------------------------
                  mm-dd-yy Comments – Author.
\*************************************************************/
/*============================[ REDEFINITION DEFENCE ]=======*/
     #ifndef FILENAME_EXT
     #define FILENAME_EXT
/*============================[ SPECIAL ]====================*/
/*============================[ PUBLIC CONSTANTS ]===========*/
/*============================[ PUBLIC TYPES ]===============*/
/*============================[ FORWARD REFERENCES ]=========*/
/*============================[ PUBLIC VARIABLES ]===========*/
/*============================[ PUBLIC FUNCTIONS ]===========*/
/*============================[ PSEUDO/INLINE FUNCTIONS ]====*/
/*============================[ END REDEFINITION DEFENCE ]===*/
#endif /* FILENAME_EXT */

/** (END OF FILE  : filename.ext) ***************************/
```

Описание каждого раздела приведено ниже.

Section	Description
REDEFINITION DEFENCE	Гарантирует однократное включение h-файла
SPECIAL	Блок операторов условной компиляции
PUBLIC CONSTANTS	Публичные константы
PUBLIC TYPES	Публичные типы
FORWARD REFERENCES	Ссылки вперед
PUBLIC VARIABLES	Публичные переменные
PUBLIC FUNCTIONS	Публичные функции
PSEUDO/INLINE FUNCTIONS	Inline функции (методы класса)
END REDEFINITION DEFENCE	Окончание однократно включаемого h-файла

Если некоторый раздел не содержит информации, его можно опустить, но порядок разделов менять не разрешается.

5.5.6. Классы

Заголовок класса представляет собой закомментированный текст, предшествующий определению класса. Он имеет следующий вид.

```
/*************************************************************\
  CLASS…….: Name of the class.
  DESCRIPTION…: The description of the class and its purpose.
\*************************************************************/
```

Пример:

```
/*************************************************************\
  CLASS…….: CdiskMngr
  DESCRIPTION…: Provides low-level file I/O operations.
\*************************************************************/
class CdiskMngr : public CcheckSumObject
{
      …
};
```

5.5.7. Методы

Заголовок метода представляет собой закомментированный текст, предшествующий определению метода. Он имеет следующий вид:

```
/*************************************************************\
  METHOD…….....: Name of the method.
  DESCRIPTION…: Purpose of the method.
  ATTRIBUTES….: Method's attributes
  ARGUMENTS……: Arguments of the method, their type and meaning.
  RETURNS……...: Possible return values and their meaning
  NOTES……......: For instance, explanation of the usage and/or an
                        example, limitations, pseudocode, etc.
\*************************************************************/
```

Поля `METHOD, DESCRIPTION, ARGUMENTS, RETURNS` являются обязательными, остальные могут отсутствовать.

Каждому аргументу метода может предшествовать один из следующих префиксов:

- ***IN*** – означает, что данный параметр – только для чтения и изменению не подлежит;
- ***OUT*** – означает, что данный параметр будет изменен при выполнении метода, однако его начальное значение методом не используется;
- ***INOUT -*** означает, что данный параметр будет изменен при выполнении метода, а его начальное значение методом используется.

Однако не все компиляторы их поддерживают (например, MS Visual C++ их не поддерживает). Поэтому предпочтительно использовать данные слова в `in-line` комментариях.

Пример:

```
/*************************************************************\
  METHOD........: Read
  DESCRIPTION...: Reads nCount bytes to pBuffer beginning from
                      the given file position.
  ATTRIBUTES....: Public
  ARGUMENTS.....: fpStart - the file position to be read from,
                   pBuffer - pointer to a buffer that is to receive
                                 read data from the file,
                   nCount  - maximum bytes to be read.
  RETURNS.......: Returns the number of bytes read. If the method
                        tries to read at end of file, it returns 0.
                        If the reading failed, the method return -1.
                        To get extended error information, call
                        GetLastError() method.
  NOTES.........: The following error codes GetLastError() can
                        return after
                        the method failed:
                        ERR_MEMORY    - not enough memory,
                        ERR_FILELOCK - the file is locked to read
\*************************************************************/

int CDiskMngr::Read
(
  /* IN  */    FilePtr fpStart,  // Начало
  /* OUT */    void*   pBuffer,  // Адрес буфера
  /* IN  */    int     nCount    // Счетчик
)
{
     ...
}
```

5.5.8. Функции

- Заголовок функции представляет собой закомментированный текст, предшествующий определению функции. Он имеет следующий вид:

```
/*************************************************************\
  FUNCTION......: Name of the function.
```

```
  DESCRIPTION...: Purpose of the function.
  ARGUMENTS.....: Arguments of the function, their type and
                       meaning.
  RETURNS.......: Possible return values and their meaning.
  EXTERNS.......: Global data used/affected in the function.
  NOTES.........: For instance, explanation of the usage and/or an
                           example, limitations, pseudocode, etc.
\************************************************************/
```

Поля FUNCTION, DESCRIPTION, ARGUMENTS, RETURNS являются обязательными, остальные могут отсутствовать.

- Каждому аргументу метода может предшествовать один из следующих префиксов:
  - ***IN*** – означает, что данный параметр – только для чтения и изменению не подлежит.
  - ***OUT*** – означает, что данный параметр будет изменен при выполнении метода, однако его начальное значение методом не используется.
  - ***INOUT -*** означает, что данный параметр будет изменен при выполнении метода, а его начальное значение методом используется.

Однако не все компиляторы их поддерживают (например, MS Visual C++ их не поддерживает). Поэтому предпочтительно использовать данные слова в in-line комментариях.

Пример:

```
/************************************************************\
  FUNCTION......: AbbreviateFileName
  DESCRIPTION...: Makes an abbreviate file name from the given
                        Full path name.
  ARGUMENTS.....: lpszCanon    - pointer to a full path name to be
                                 abbreviated,
                  ichMax       - max number of characters of the
                                 resultant file name,
                  bAtLeastName - specify whether the file name
                                 should be left at least after
                                 abbreviation.
  RETURNS.......: None.
  NOTES.........: Below is the example how the function works.
                   lpszCanon = "C:\MYAPP\DEBUGS\C\TESWIN.C";
                   ichMax bAtLeastName  Result (lpszCanon)
                   ------- ------------ --------------------------
                       1-7   FALSE        <empty>
                       1-7   TRUE         TESWIN.C
                       8-14    x          TESWIN.C
                       15-16   x          C:\...\TESWIN.C
                       17-23   x          C:\...\C\TESWIN.C
                       24-25   x          C:\...\DEBUGS\C\TESWIN.C
                       26+     x          C:\MYAPP\DEBUGS\C\TESWIN.C
\************************************************************/

void PASCAL AbbreviateFileName
(
  /* INOUT */    LPTSTR lpszCanon,
```

```
  /* IN    */    int    ichMax,
  /* IN    */    BOOL   bAtLeastName
)
{
      ...
}
```

5.5.9. Документирование каталогов

Каждый каталог должен содержать `README` файл, в котором описано:

- Назначение каталога и его содержимое.
- Одна строка комментария на каждый файл. Комментарий обычно извлекается из поля `NAME` оглавления файла.
- Описание процедуры инсталляции.
- Перечень ресурсов и ссылки на них:
  - каталоги исходных файлов;
  - оперативная документация (справочная система и др. доступные в электронной форме руководства);
  - перечень бумажных документов (т.е. документов, представленных не в электронной форме).

5.5.10. Требования к комментариям

- Комментарии должны формулироваться таким образом, чтобы, если их извлечь из кода программы, они составили осмысленное связное изложение – описание функционирования программы.
- Комментарии должны документировать принимаемые вами решения (что вы делаете в данном месте программы и почему).
- Комментарии должны быть лаконичными и понятными. Встроенные ключевые слова (*gotchas*) можно использовать, чтобы отметить места, чреватые проблемами.
- Gotcha Keywords – ключевые слова в комментарии.

  :TODO: topic. Означает: ЗДЕСЬ НАДО ДОРАБОТАТЬ. НЕ ЗАБУДЬ.

  :BUG: [bugid] topic. Означает: ЗДЕСЬ НАХОДИТСЯ ИЗВЕСТНЫЙ BUG. Объясните его причину и присвойте ему идентификатор (bug ID)
- **:KLUDGE:** Если вы сделали что-то неудачное, объясните, как это произошло, и что вы намерены делать впредь, чтобы избежать этого в будущем.
- **:TRICKY:** Предупреждение, что следующая часть программы очень сложная, ее не следует изменять без тщательного обдумывания.
- **:WARNING:** Предупреждение о чем-либо.
- **:COMPILER:** Иногда требуется решить проблему с компилятором.
  - Задокументируйте ее. Проблема может быть решена позднее.

- **:ATTRIBUTE: value**. Общий вид атрибута, вставленного в комментарий. Вы можете задавать собственные атрибуты, которые затем могут быть извлечены из комментария.

В дополнение вы можете использовать возможности препроцессора, но имейте в виду, что не все компиляторы их поддерживают.

```
#define chSTR(x)     #x
#define chSTR2(x)         chSTR(x)
#define TODO(desc)   message(":TODO:("__FILE__",    \
                              "chSTR2(__LINE__) ") " #desc)
#define BUG(desc)    message(":BUG:("__FILE__",     \
                             "chSTR2(__LINE__) ") " #desc)
```

Когда вы вставляете эти строки в ваш `.h/.cpp` – файл, вы можете использовать их следующим образом:

```
main()
{
  if (condition)              // in-line comment
  {
     #pragma TODO("Add error handling here")
     ...
  }
  else if (condition)         // in-line comment
  {
     ...
  }
  else                        // in-line comment
  {
     #pragma TODO("There is no error handling here")
     ...
  }
}
```

Это позволит видеть все потенциальные проблемы программы во время ее компиляции.

- Форматирование кода программы при использовании ключевых слов Gotcha.
- Ключевое слово Gotcha должно быть первым словом комментария.
- Комментарий может состоять из нескольких слов, но первое слово должно содержать смысл комментария, а последующие слова служат лишь для уточнения.
- Имя автора и дата внесения замечания должны быть частью комментария.

Часто Gotcha остаются в программе дольше, чем это необходимо. Данные о дате и авторе изменений позволят принять другим

программистам решение. Зная имя автора, вы сможете связаться с ним и получить консультацию.

Пример:

```
// :TODO: tmh 960810: possible performance problem
// We should really use a hash table here but for now we'll
// use a linear search.

// :KLUDGE: tmh 960810: possible unsafe type cast
// We need a cast here to recover the derived type. It should
// probably use a virtual method or template.
```

5.6. Операторы

5.6.1. Оператор if…else

- Если `if` завершается оператором `return`, то не используйте `else`. Код получается более понятным.

  Так, вместо

```
if (condition)  return 0;
else
{
   do_something();
}
```

  лучше написать

```
if (condition) return 0;
do_something();
```

  В этом случае последним `return` может быть выдача кода ошибки (это позволит отловить «случайное» попадание в ненужную ветвь алгоритма).
- Необходимо стараться проверять все альтернативные возможности (в том числе все “предельные случаи”).

5.7. Циклы

- По возможности следует избегать циклов `do…while`. Они опасны тем, что тело цикла исполняется всегда хотя бы один раз. Кроме того, при чтении программы очень трудно отыскать проверяемое условие, т.к. `while` в такой конструкции располагается «внизу», т.е. в конце тела цикла. По этим причинам никогда не следует употреблять `do…while` для организации бесконечного цикла – для этой цели намного лучше использовать `while(1)`.
- Всегда используйте оператор `for`, если присутствуют любые два из инициализирующего, условного или инкрементного выражений. Это сокращает код и делает его более понятным.

5.8. Препроцессор

- Обязательно заключайте в скобки тела сложных макросов и их аргументы. Например, следует писать
  - `#define TWO_K (1024+1024) //correct!`
    - а не
  - `#define TWO_K 1024+1024 //incorrect!`
  - В последнем случае `TWO_K*10` будет вычислено на этапе компиляции как `1024+1024*10=11264` вместо ожидаемого `20480`.
- Из приведенного примера видно, что макросы может быть достаточно трудно сопровождать. Поэтому:
- для задания констант, используемых в программе (кроме системо-зависимых параметров) по возможности следует использовать модификатор `const` или перечисление `enum` и избегать использования `#define`;
- вместо параметризованных макрооопределений `#define` лучше использовать `inline`-функции.

5.9. Константы и перечисления

Следует помнить, что `const int` на самом деле не задает константу, а резервирует память под `int` (хотя и запрещает менять содержащееся в ней значение). Под перечисление `enum` память никогда не выделяется, поэтому использование `enum` может быть предпочтительнее.

5.10. Мобильность

Мобильность программ – это свойство, позволяющее выполнять программы на разных компьютерах, под управлением разных операционных систем, с минимальными изменениями кода. Естественно, мобильность предусматривает и возможность трансляции программы разными компиляторами. Думать о мобильности нужно даже в том случае, если предполагается, что программа пишется, скажем, только «под» Windows NT и только с использованием MS Visual C++ (т.е. никогда не планируется использовать другой компилятор). Дело в том, что аппаратное и программное обеспечение не стоит на месте, и вполне возможно, что через несколько лет написанную вами программу придется полностью переделывать под новую версию «старой» операционной системы и, соответственно, под новый компилятор.

5.10.1. Зависимость от компилятора

Некоторые детали в C/C++ не стандартизованы. Например:

- не определен порядок вычисления операндов большинства бинарных

операций, таких как сложение и умножение. Следовательно, не определен порядок появления возможных побочных эффектов;

- некоторые директивы и ключевые слова (в основном это директивы препроцессора) поддерживаются только определенными компиляторами.

Не следует явно или неявно пользоваться нестандартными возможностями, предоставляемыми конкретным компилятором.

5.10.2. Зависимость от компьютера

- Не следует полагаться на определенный размер машинного слова, он может быть разным у разных компьютеров.
- Размеры данных базовых типов (`int`, `float`, `double`, `char` и т.п.) в байтах и в машинных словах могут быть разными у разных компьютеров и в разных операционных системах. Гарантировано только, что размер `short` не меньше размера `int`, размер `float` не меньше размера `double` и т.п. Размеры данных могут сказаться на обработке двоичных масок. Например, в фрагменте

```
#define MASK 0177770  // incorrect!

int x;
  …
x &= MASK;
```

три правых бита целого `x` будут обнуляться только в случае, если данные типа `int` занимают 16 бит. Но если размер данных типа `int` больше 16 бит, то обнуляться будут левые биты `x`. Поэтому предпочтительнее

```
#define MASK (~07) // correct!
int x;
  …
x &= MASK;
```

- По возможности следует вообще избегать использования двоичных масок. Это рудимент “ассемблерного” стиля программирования. Лучше использовать битовые поля.
- Используйте `sizeof()` для определения размера объектов, в том числе и переменных базовых типов.
- Следует тщательно проверять операции двоичного сдвига. Максимальное число бит, которые могут быть сдвинуты вправо или влево, различается на разных компьютерах и в разных операционных системах.
- Максимальный размер битового поля зависит от размера машинного слова и, следовательно, не является стандартным.
- При работе с объектами разных типов данных (классов) используйте преобразование типа. Особенно важно использовать преобразование типов для указателей.

- Используйте макроопределения `#define` для задания системозависимых именованных констант.
- Не полагайтесь на внутреннюю кодировку целых и вещественных типов данных. Например, не следует полагаться на использование дополнительного или обратного кода для представления целых типов.
- Не являются стандартными порядок и число байт в машинном слове, число бит в байте (данная константа определена в файле `<values.h>`).
- Не следует полагаться на конкретные значения основных констант, таких как `NULL` и `EOF`. Например, `NULL`, как правило, является константой `0`, но это вовсе не обязательно. Поэтому

```
if ( lval == NULL ) {  … }
```

  лучше, чем

```
if ( !lval )  { … }
```

- Символы `(char)` могут иметь знак. Для повышения мобильности можно использовать явное описание `unsigned char` или преобразовывать символы перед обработкой к типу `unsigned char`.
- Нельзя полагаться на конкретную кодировку символов, в том числе на определенную последовательность символов в кодировке. Например,

```
if ( islower(ch) ) { … }
```

  лучше, чем

```
if ( ch >= 'a' && ch <= 'z' ) { … }
```

  так как использует стандартную библиотечную функцию `islower()`, а не полагается на предположение о последовательности строчных букв в кодировке.
- Для выделения/освобождения памяти лучше использовать `new/delete`, а не `malloc()/free()`.

5.11. Правильная организация программ

Программа организована правильно, если ее легко читать, модифицировать, эксплуатировать и, следовательно, переносить на другие компьютеры и в другое операционное окружение. Для правильной организации программ имеет смысл придерживаться следующих рекомендаций.

- Все определения, связанные с конкретным компьютером и/или операционной средой, следует оформлять при помощи директивы препроцессора `#define` и помещать в отдельный заголовочный файл. Туда же следует помещать системозависимые определения типов данных (для этой цели стоит использовать `typedef`). Стандартные системные определения типов содержатся в `<sys/types.h>`.

- Для локализации системозависимых программных фрагментов следует использовать директивы условной компиляции и директиву `#define`. Для локализации системозависимых определений типов данных следует использовать `typedef`.
- Следует особенно тщательно проверять число и тип аргументов, передаваемых функциям (методам классов). Для мобильного определения функций с переменным числом аргументом используйте средства, описанные в файле `<varargs.h>`. Тип передаваемых в функцию (метод) аргументов должен соответствовать типу формальных параметров; добиться этого необходимо при помощи приведения типов.
- Стоит активно пользоваться модификатором `const` применительно к передаваемым в функцию (метод) параметрам для указания того, что эти параметры не могут быть изменены. Например, описание
- `void func (const SomeClass &);`
- запрещает модификацию переданного по ссылке объекта функцией `func`. В идеале все ссылочные аргументы функций (методов) должны быть описаны как `const`.
- Следует тщательно следить за инициализацией указателей и переполнением их значений. Нельзя полагаться на инициализацию переменных (в т.ч. указателей) по умолчанию.
- Имеет смысл пользоваться классом String для работы со строками. Использование `char*` в качестве синонима `String` – это потенциально опасный рудимент «чистого» C.
- Не следует описывать тело метода внутри определения класса. Исключения могут составлять очень короткие методы и короткие перегружаемые операторы. Методы, тело которых включено в определение класса, как правило, реализуются компилятором как `inline`. При необходимости задать inline-метод (функцию) лучше сделать это путем явного объявления.
- Следует стремиться к тому, чтобы в используемых классах все данные были `private`. Желательно избегать дружественных `(friend)` функций и классов.

5.12. Основы тестирования

Тестирование — важнейший процесс разработки. В общем случае тестирование — это проверка соответствия требованиям. В этом смысле тестированию подлежит не только программный код (это само собой разумеется) но и все артефакты процесса разработки.

5.12.1. Процесс контроля качества

Ответственность за качество артефакта в первую очередь несет человек, создающий этот артефакт. Но, как бы то ни было, «один в поле не воин». Всем требуется сторонний взгляд на выполняемую работу. Взгляд со стороны необходим, чтобы избежать недальновидности, нереалистичной самооценки и застоя. Процесс контроля качества является также общественной обязанностью. Каждая часть работы, выполненная инженером, должна быть детально проверена по крайней мере одним человеком, желательно независимым от автора работы.

В дополнение к ответственности индивидуальных разработчиков, многие организации определили процесс раздельной систематической и полной проверки всех артефактов— *контроль качества* (КК). В функции контроля качества входят проверки, инспектирование и тестирование. Контроль качества должен начинаться вместе с запуском каждого проекта (рис. 33). Лучше всего привлекать контроль качества и для проверки правильности используемого процесса и актуальности документации.

Рисунок 32. Осуществление контроля качества.

5.12.2. Методы «белого ящика» и «черного ящика»

В случае контроля качества методом *«черного ящика»* приложение (или какая-либо его законченная часть) анализируется как целое. Этот метод используется для проверки того, что приложение (его часть) отвечает предъявляемым требованиям. Контроль качества методом *«белого* (*стеклянного*) *ящика»* осуществляется на уровне компонентов, из которых построено тестируемое приложение (его часть). Можно провести такую аналогию: если вы проверяете работу телевизора, просто включая и выключая его и переключая каналы, то вы применяете метод черного ящика; если же вы тестируете телевизор, анализируя его работу на уровне взаимодействия микросхем, то есть компонентов, из которых телевизор собран, вы применяете метод белого ящика.

Чаще всего о методах «черного» и «белого ящика» говорят в контексте тестирования. Однако эти методы применимы и к другим способам контроля качества. Метод «белого ящика» основан на рассмотрении анализируемого артефакта с точки зрения его структуры, формы и назначения; здесь применяются формальные методы и рассматриваемое в следующем разделе инспектирование. Метод «черного ящика» задается вопросом «Обладает ли построенный объект требуемым поведением?». При этом для тестирования по методу «черного ящика» необходимо иметь набор пар «вход–выход», чтобы можно было проверить правильность поведения.

5.12.3. Цели тестирования

Невозможно протестировать программу абсолютно во всех аспектах, поскольку число вариантов работы нетривиальной компьютерной программы может быть неограниченным.

Тестирование не может доказать *отсутствия* ошибок в программе. Тестирование может только показать *присутствие* ошибок.

Тестирование часто неправильно воспринимается как процесс подтверждения корректности кода, что можно выразить таким высказыванием: «Протестируй это, чтобы убедиться, что тут все правильно». Главная цель тестирования далека от подтверждения корректности. Цель тестирования не в том, чтобы показать удовлетворительную работу программы, а в том, чтобы четко определить, в чем работа программы неудовлетворительна!

Тестирование требует значительных затрат, и нужно стараться получить от этих затрат максимальную прибыль. Для данной тестируемой программы, чем больше дефектов будет найдено на каждый человеко-час тестирования, тем выше выигрыш от вложений в тестирование.

Целью тестирования является обнаружение как можно большего числа дефектов, как можно более серьезных и как можно раньше.

Согласно современным представлениям, тестирование оценивается более чем половиной времени, затрачиваемого на проект. Наградой за нахождение дефекта на ранней стадии процесса является по крайней мере десятикратная экономия по сравнению с обнаружением этого же дефекта на этапе интеграции или, еще хуже, после отправки заказчику. Следовательно, мы должны тестировать *рано* и *часто*.

5.12.4. Модульное тестирование

Модульное тестирование является ранним типом тестирования. Следующий уровень состоит из *интегрального тестирования*. Наконец, приемосдаточные испытания валидируют финальный продукт. Эти виды тестирования описаны в последующих разделах.

5.12.5. Значение модульного тестирования

Цель модульного тестирования — проверить отдельные элементы, в то время как цель всех других видов тестирования обычно заключается в проверке функциональности все системы в целом. Функции обычно являются наименьшими частями программы, к которым может быть применено модульное тестирование. Следующим по величине элементом является модуль (класс в объектно-ориентированном случае). Иногда комбинации модулей рассматриваются в целях тестирования как «модули».

Модульное тестирование является дополнением к инспектированию и использованию формальных методов проверки корректности.

5.12.6. План модульного тестирования

Типичный план модульного тестирования, основанный на стандарте IEEE 1008-1987, состоит в следующих шагах.

1. Спланировать общий подход, ресурсы и расписание.
2. Определить характеристики, которые следует протестировать, исходя из требований.
3. Обновить общий план.
4. Разработать набор тестов.
5. Реализовать обновленный план и проект.
6. Выполнить тестовые процедуры.
7. Проверить окончание работы.
8. Оценить тестирование и модули.

5.12.7. Разбиение на классы эквивалентности для тестирования «черного ящика»

Поскольку у нас нет возможности протестировать все комбинации входных данных, мы ищем представительные варианты тестов. Проблема заключается в нахождении наилучшего представления бесконечного множества возможностей наиболее представительным

конечным множеством. *Разбиение на классы эквивалентности* — это разбиение множества входных тестовых данных на подмножества таким образом, чтобы при условии успешного выполнения теста для одного элемента подмножества остальные элементы подмножества также с большой вероятностью прошли бы тест успешно. Например, при тестировании функции `setName( String )` в классе `GameCharacter` успешное завершение тестового вызова `setName( "Harry" )` означает, что у нас, вероятно, не будет проблем при тестировании функции `setName()` с любой строкой из пяти символов. Более того, мы, вероятно, можем расширить это утверждение на все имена с не менее чем одним и не более чем `maxNumCharsInName()` символами.

5.12.8. Анализ граничных значений для тестирования «черного ящика»

Особое внимание следует уделять значениями, которые лежат на границах классов эквивалентности. Например, в функции `setName( String )`, упоминавшейся выше, границами являются строки нулевой длины и строки длиннее `maxNumCharsInName()`.

В тестировании проекта значения, лежащие за пределами допустимых границ (например, ввод нелопустимых данных), также используются в качестве тестовых данных. Тестовые данные генерируются после того, как будут установлены границы эквивалентных классов.

5.12.9. Использование утверждений для тестирования «белого ящика»

Очень полезным является выявление утверждений (assert) о значениях переменных программы в определенных точках. Каждое имеющееся утверждение в программе должно быть проверено по крайней мере одним тестом. Во многих случаях утверждения остаются инвариантными (неизменными) в стратегически важных блоках кода. Часто бывает полезно добавлять в исходный код команды, сообщающие о том, действительно ли утверждение, который мы предполагаем истинным, сохраняется таковым во время работы программы. Эта техника «белого ящика» называется *ассерторическим тестированием*.

5.12.10. Рассмотрение путей для тестирования «белого ящика»

Рассмотрение возможных путей выполнения программы гарантировать, чтобы при выполнении набора тестов программа проходила по каждой каждую ветви каждого решения и каждая итерация каждого цикла выполнилась хотя бы один раз. Этот подход не

так сложен для циклов `for`, однако он представляет некоторые сложности для циклов `while`.

Иногда все возможные варианты можно просчитать, иногда их можно разбить на типовые группы. Однако в некоторых случаях полное рассмотрение решений с помощью циклов `while` практически невозможно. Однако циклы `while` часто допускают применение формальных методов и инспектирования. Это иллюстрирует взаимно дополнительную природу формальных методов, инспектирования и тестирования.

5.12.11. Использование случайных величин в тестировании

Вообще говоря, мы используем случайные входные данные при тестировании с целью избежать возникновения необъективных экземпляров тестов. Когда выбран тип теста (например, построены классы эквивалентности) и определены границы данных, для возможных входных данных теста существует некоторая свобода. В идеале эти входные данные должны выбираться случайным образом.

Продолжим пример из раздела 4.5.1.3. Для функции `setName( String )` целесообразно подготовить два теста.

- Выбрать целое *i* больше нуля и не больше, чем `maxNumCharsInName()`, случайным образом. Для *j=0…i* выбрать случайным образом букву и установить в *name[j]* эту букву. Вызвать `setName( name )`. Ожидаемый результат– успешное выполнение.
- Выбрать целое *i* больше, чем `maxNumCharsInName()`, случайным образом. Для *j=0…i* выбрать случайным образом букву и установить в *name[j]* эту букву. Вызвать `setName( name )`. Ожидаемый результат – сообщение об ошибке.

5.12.12. Планирование модульных тестов

Систематический подход в тестировании необходим, поскольку число потенциальных модулей, нуждающихся в тестировании, обычно очень велико. Достаточно легко сказать, что «каждая часть работы должна быть протестирована», однако эта фраза несет в себе мало смысла, поскольку на этап тестирования выделяется лишь ограниченное количество ресурсов. Шаги планирования модульного тестирования перечислены ниже.

- **Определить принципы модульного тестирования.** Первый вопрос заключается в определении того, какие модули мы будем рассматривать, и кто будет их тестировать. Для объектно-ориентированных проектов обычная организация модульного тестирования заключается в тестировании методов каждого класса, затем классов каждого пакета, затем пакета в целом. Тестирование

модуля в идеале планируется и выполняется человеком, не участвовавшим в разработке. Модульное тестирование иногда планируется и исполняется организацией контроля качества. Хотя достоинством такого подхода является независимость тестирования, в этом случае от инженеров контроля качества требуется понимание проекта в деталях. Некоторые организации-разработчики не поддерживают эту возможность, и модульное тестирование часто остается за группой разработчиков и выполняется по их собственному усмотрению. В любом случае тесты делаются доступными для инспектирования и для возможного повторного использования.

- **Определить способ документирования модульных тестов.** Документирование модульных тестов состоит из тестовых процедур, входных данных, кода, исполняющего тест, и выходных данных. Модульные тесты можно упаковывать вместе с кодом либо в отдельных документах. Достоинство упаковки тестов вместе с кодом заключается в удобстве. Недостатком является увеличение объема кода. Кроме того, для выполнения модульных тестов используются драйверы тестов и заглушки. Они также документируются для будущего использования.
- **Определить границы модульного тестирования.** Поскольку протестировать «все» невозможно, границы тестирования должны быть сознательно определены. Например, «не менее чем по одному тесту на допустимое значение, граничное значение и недопустимое значение». Границы того, что относится к модульному тестированию, также должны быть определены. Например, входит ли сюда тестирование пакетов, или оно должно относиться к другому типу тестирования? Разработчики заранее определяют границы тестирования, в том числе и момент, когда процесс тестирования должен быть завершен. Например, следует ли тестировать каждый модуль одинаковое количество времени или до обнаружения первых трех ошибок?
- **Определить, как и где получать тестовые входные данные.** Мы обсудили допустимые, граничные и недопустимые входные тестовые данные. Также необходима некоторая случайная генерация данных. По возможности используются инструменты, генерирующие входные тестовые данные посредством анализа исходного кода и обнаружения граничных значений данных и ветвления. Вдобавок значительный объем тестовых данных можно получить из предыдущих версий программы, стандартных источников, промышленных контрольных задач и т. д. Все это документируется для будущих ссылок и повторного использования.
- **Оценить необходимые ресурсы.** Как всегда при планировании, мы определяем трудозатраты и время, необходимое для выполнения модульного тестирования. Основным источником этой оценки

являются накопленные данные. Хотя модульное тестирование часто связано с процессом разработки, отдельное его выполнение предоставляет бесценную информацию.

- **Организовать учет времени, дефектов, их тип и источник.** Участвующие инженеры определяют четкую форму, в которой они будут вести учет затраченного на модульное тестирование времени, учет найденных ошибок и их типов. Полученные данные используются для оценки состояния программы и предсказания конечного качества работы и сроков окончания. Данные также становятся частью учетных записей истории проекта.

5.12.13. Интеграция и системное тестирование

Поскольку программные продукты довольно сложны по своей структуре, их формируют из частей, которые создаются независимо, а затем собираются в единое целое. Интеграция как раз и относится к процессу сборки. Различные виды тестов проводятся как над частично собранным приложением, так и над всем продуктом в целом.

Фаза интеграции водопадного процесса часто преподносит неприятные сюрпризы, связанные с несовместимостью интегрируемых частей. По этой причине инкрементальный процесс разработки программного обеспечения, в частности, старается избегать сборки большого количества элементов, используя последовательную интеграцию с помощью многочисленных итераций.

Во избежание этих потенциально возможных потерь информации используется непрекращающееся тестирование и интеграция. Несмотря на то, что тестирование модулей как части целого приложения в реальном окружении имеет огромное значение (это называется системное тестирование), оно не заменяет тщательного тестирования каждого модуля в отдельности перед добавлением его к системе.

5.12.14. Верификация, валидация и системное тестирование

Верификация позволяет определить, правильно ли мы создаем приложение.

Другими словами, действительно ли мы на текущей фазе создаем именно те артефакты, что были специфицированы? Применительно к интеграции верификация приравнивается к подтверждению того, что мы собираем именно те компоненты, которые мы планировали собрать, и именно тем способом, каким это было запланировано. Такая проверка может быть произведена при помощи инспектирования результатов интеграции.

Валидация позволяет выяснить, правильный ли результат у нас получается.

Другими словами, удовлетворяет ли наш продукт требованиям? На фазе интеграции это проверяется с помощью *системного тестирования*.

После завершения сборки тщательное тестирование требует, чтобы мы сначала выполнили модульные тесты функций (методов) и модулей (классов или пакетов). В этот раз, однако, эти тесты следует пройти в некотором контексте, а не изолированно друг от друга. Здесь требуется меньше драйверов и заглушек, что приводит к меньшему количеству сложностей и ошибок. Если мы тестируем финальную сборку, то вообще не следует использовать драйверы или заглушки.

Когда код системы интегрирован становится возможным протестировать части в контексте всей системы вместо использования автономного подхода. Чтобы сфокусировать тестирование на разработанных частях программы, нам придется продумать подходящие входные данные.

Системное тестирование – это тестирование все системы в целом.

Существует несколько видов системного тестирования.

Контекстное модульное тестирование. После того как система собрана, становится возможным повторно протестировать *модули* (например, пакеты) в контексте системы.

Тестирование интерфейсов подразумевает повторную валидацию интерфейсов между модулями.

Цель *регрессионного тестирования* заключается в проверке того, что добавления к системе не уменьшили ее возможностей. Другими словами, при добавлении в систему нового модуля, следует сначала убедиться, что все старые модули не потеряли работоспособность. Только когда новый модуль прошел регрессионное тестирование, можно тестировать его работу в контексте системы.

Нагрузочное тестирование. Системное тестирование также валидирует требования, как функциональные, так и нефункциональные. Нефункциональные требования включают в себя требования к рабочим характеристикам, таким как скорость работы и использование ресурсов.

Тестирование удобства и простоты использования валидирует приемлемость программы для ее конечных пользователей.

Тестирование инсталляции выполняется при установке программы на целевых платформах.

Приемосдаточные тесты выполняются клиентом для валидации приемлемости программы.

Далее мы подведем итоги и обсудим типы тестирования более подробно.

5.12.15. Процесс интеграции

Простейший вид интеграции состоит из добавления новых элементов к базису (существующему коду) на каждой итерации инкрементального процесса или на каждом витке спирального процесса.

Фаза реализации состоит из кодирования новых частей, после которого эти новые части интегрируются в базис.

По завершении разработки архитектуры важно определить легкость, с которой части будут интегрироваться в проект. В случае программных разработок редко удается завершить отдельные программные модули до их интеграции в проект. Таким образом, в программные сборки часто бывает необходимо интегрировать в частично сконструированные модули

Хотя типовой процесс сборки имеет недостаток, заключающийся в работе с незавершенными модулями, он имеет и преимущество, состоящее в выполнении интеграции на ранних стадиях процесса разработки. Это помогает уменьшить риск, связанный с интегрированием завершенных крупных модулей.

Тестирование упрощается после объединения всех реализаций вариантов использования в каждой сборке вместо тестирования частей вариантов использования. Разрабатывая относительно небольшие варианты использования, вы, прежде всего, упрощаете процесс добавления их в сборку. Поскольку пользовательские интерфейсы рано или поздно все равно придется тестировать, желательно, чтобы их самих или их ключевые части можно было встроить как можно раньше, тем самым обеспечив возможность тестирования развивающейся программы. Альтернативой является сборка временных интерфейсов для использования во время интегрального тестирования.

Типовая последовательность действий для интеграции программной системы показана на рис. 34

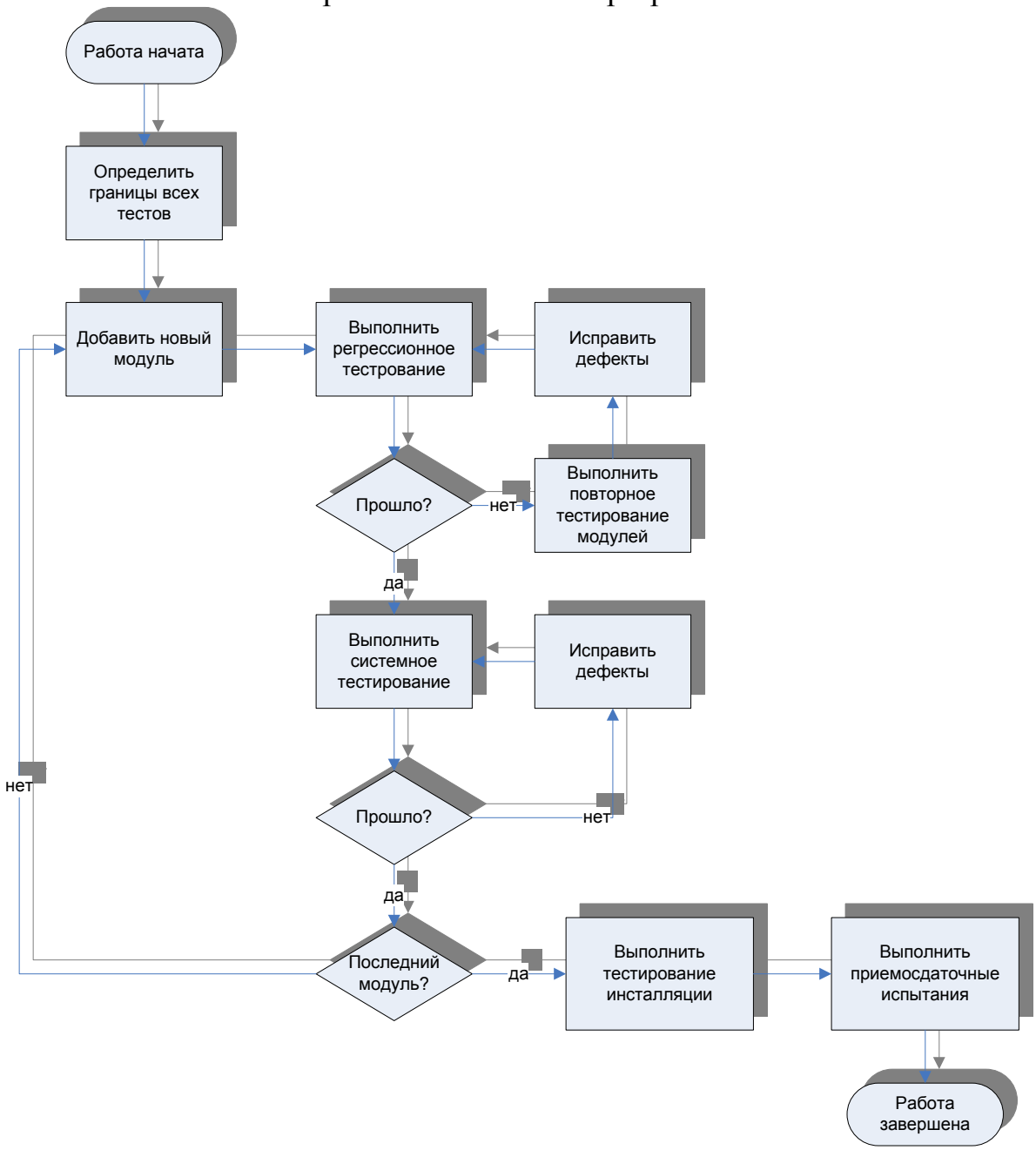

Рисунок 33. Блок-схема процесса интеграции.

5.12.16. Тестирование удобства и простоты использования

Хороший интерфейс может значительно повысить ценность программы. Тестирование *удобства и простоты использования* проверяет приемлемость программы для пользователей.

Основная задача тестирования удобства и простоты использования заключается в гарантии того, что программа удовлетворяет своим требованиям. Один из способов организации такого тестирования заключается в измерении степени удовлетворенности, полученной пользователями от применения программы.

Критерии оценки удобства и простоты использования должны быть сформулированы заранее. Например, мы можем потребовать, чтобы произвольная группа из n пользователей нашей программы оценила программу по условной шкале из m баллов, и чтобы среднее

значение оценки оказалось не меньше *k*. Необходимое количество пользователей, грануляция шкалы и минимальная оценка определяются статистически и зависят от ожидаемого числа конечных пользователей и желаемой вероятности достоверного заключения.

При разработке вопросников удобства и простоты использования задача заключается в получении данных, позволяющих инженерам направить усилия на исправление наиболее серьезных упущений, не злоупотребляя временем и терпением пользователей. Накопление данных по удобству и простоте использования может оказаться дорогим процессом, поскольку пользователи часто ожидают компенсацию за свое потраченное время и предоставление информации.

5.12.17. Регрессионное тестирование

Когда программа разрастается до больших размеров, повышается важность регрессионного тестирования. Это особенно заметно, когда в большую программу вносятся изменения и разработчикам нужно подтвердить тот факт, что изменение не повредило существующую функциональность. Первый вопрос, возникающий после интеграции изменения, обычно следующий: «Остался ли продукт тем же, чем он был раньше, с расширенной функциональностью?» Повторное полное инспектирование обычно нерационально, поэтому важным практическим методом получения ответа на этот вопрос является проверка того, что система продолжает проходить тот же определенный набор системных тестов, что и до изменений. Этот процесс верификации называется *регрессионным тестированием*.

Регрессионное тестирование проводится достаточно часто. Если время не позволяет выполнить регрессионное тестирование, выбираются тесты, которые система после внесения изменений с наибольшей вероятностью не пройдет.

5.12.18. Приемосдаточное тестирование

Разрабатывающая программу организация и организация-заказчик являются двумя сторонами, заключившими контракт. После завершения работ мудрый разработчик получает окончательное утверждение заказчика, согласно которому можно начинать поставку программы. *Приемосдаточные тесты* разрабатываются для убеждения клиента в том, что указанная программа действительно создана. Приемосдаточные тесты могут ничем не отличаться от системных тестов, созданных разработчиком, но на этот раз они должны быть официально засвидетельствованы организацией-заказчиком и проверены на целевых платформах.

5.12.19. Тестирование инсталляции

Тот факт, что мы протестировали программу в нашей собственной среде, вовсе не означает, что программа будет корректно работать в

среде заказчика, поскольку при переходе из одной среды в другую открываются необъятные просторы для новых ошибок. *Тестирование инсталляции* состоит из тестирования программы в целевой программно-аппаратной конфигурации. Это влечет за собой инсталляцию программы в целевой среде и выполнение комплекта системных тестов.

5.12.20. Альфа- и бета-версии

Во многих случаях предполагаемые пользователи хотят наряду с заказчиками участвовать в процессе системного тестирования. Этот процесс управляется с помощью альфа- и бета-версий.

Альфа-версии передаются внутренним пользователям или строго отобранной надежной группе внешних пользователей для раннего предвыпускного использования.

Назначение альфа–версий — предоставить организации-разработчику обратную связь и информацию о дефектах от группы людей, превосходящей тестеров в количестве, без изменения репутации пока не выпущенного продукта. После распространения альфа-версии выпускается бета-версия.

Бета-версии раздаются части сообщества заказчиков с учетом того, что заказчики должны будут докладывать об обнаруженных ошибках.

Кроме того, альфа- и бета-версии используются для убеждения потенциальных клиентов в том, что помимо обещаний разработчика существует уже почти готовый продукт. Иногда распространение предварительного выпуска продукта является стратегическим приемом, используемым для внушения покупателям мысли о том, что они не должны покупать конкурирующие продукты, а вместо этого следует подождать выхода программы, которая проходит стадию бета-тестирования.

Основной мотивацией альфа- и бета-тестирования является получение более полной информации о продукте. Разработчики могут получить информацию о программе (обычно об ее пользовательском интерфейсе), чтобы в будущем иметь возможность учесть мнение пользователей. Пользователи получают возможность обдумать покупку этой программы.

5.12.21. Критерии завершения интеграции и тестирования

Рано или поздно циклический процесс интеграции и тестирования должен быть остановлен.

Критерием остановки являются условия, при которых продукт следует допустить к приемосдаточному тестированию.

Если эти условия не определены заранее, тестирование обычно проводят до тех пор, пока не закончится время, но это не самый

эффективный способ использования времени тестирования. Примером критерия остановки может быть: «Максимум две ошибки среднего или низкого уровня найдено за неделю бета-тестирования». Приведем классификацию критериев остановки.

- **Завершение конкретной методики тестирования**. Например, «все тесты покрывающие пути в графе переходов программы успешно пройдены».
- **Оценка границ количества дефектов по категориям**. Например, «95% утверждений (asserts) в программе выполняются».
- **Скорость нахождения ошибок**. Например, «не более 2 дефектов не более чем средней важности на последние 100 часов тестирования».
- **Общее число обнаруженных ошибок**. Например, «не более 5% существующих ошибок остались не найденными».

В последнем пункте упоминаются *оставшиеся ошибки*, но как мы можем оценить число оставшихся ошибок? Один из способов называется «засев». Он состоит в искусственном добавлении некоторого количества ошибок в программу и определения их процентного соотношения среди ошибок, найденных независимым тестером за определенный срок. Это число затем используется для оценки числа оставшихся дефектов.

Например, если 40 из 50 посеянных ошибок найдены за период тестирования, можно оценить количество необнаруженных ошибок на каждую обнаруженную как 10/40 = 0,25. Таким образом, если за тот же период было найдено 100 «непосеянных» ошибок, то в системе осталось порядка 100×0,25 = 25 необнаруженных ошибок.

5.12.22. Документирование интеграции и тестирования

Стандарт ANSI/IEEE для тестовой документации включает следующие разделы.

1. *Введение* – излагаются используемые принципы тестирования.
2. *План тестирования* объясняет, как следует организовать персонал, программы и оборудование, чтобы выполнить тестирование.
3. *Проект тестирования* отражает следующий уровень детализации после плана тестирования. Он раскрывает значение соответствующих программных элементов, описывает порядок, в котором их следует тестировать, называет тестовые варианты, которые следует применить.
4. *Тестовые варианты* состоят из наборов входных данных, которые должны использоваться для выполнения теста.
5. *Тестовые процедуры* — это полные подробные шаги выполнения плана тестирования. Сюда входят все

процедуры настройки, имена необходимых файлов с исходными данными и объектным кодом, выходные файлы, log-файлы, файлы с тестовыми вариантами и подробные отчеты. Без подробной пошаговой документации тестирования такие тесты невозможно достоверно воспроизвести, и дефект, возможно, вообще не удастся выявить повторно. В этом случае ошибку исправить трудно, если не невозможно.

6. *Отчет о проведении тестирования элементов* резюмирует запускаемые тесты, список ответственных лиц, используемые версии продукта и т. д.
7. *Журнал испытаний* представляет собой подробный текущий отчет о полученной во время тестов информации. Он может оказаться полезен при попытке воспроизвести ситуации, в которых тест завершился неудачно.
8. *Отчет о происшествиях* уточняет заслуживающие внимание события, происшедшие во время тестирования. Примерами могут быть отклонения от нормальной работы программы и допущенные в процессе тестирования ошибки.

5.12.23. Метрики для интегрального и системного тестирования

Ниже приведены метрики, взятые из стандарта IEEE 982.1-1998 — стандартного словаря метрик.

- **IEEE 1. Плотность отказов** = [Число уникальных отказов, найденных при тестировании] / [Число строк кода].
- **IEEE 2. Плотность дефектов** = [Число уникальных дефектов, найденных при тестировании] / [Число строк кода].
- **IEEE 5. Функциональный охват теста** = [Число протестированных функциональных требований] / [Общее число требований]
- **IEEE 10. Индекс зрелости программы** = [M – Fa – Fc – Fd] / M, где M — число частей имеющегося базиса; Fa — число добавленных частей в текущем базисе по сравнению с предыдущим базисом; Fc — число частей в текущем базисе, измененных по сравнению с предыдущим базисом; Fd — число частей, удаленных из предыдущего базиса. «Частями» могут быть функции, классы, пакеты, модули и т. д. Развитые программы имеют индекс зрелости, близкий к единице. Это означает, что число затронутых частей невелико по сравнению с общим числом компонентов.
- **IEEE 18. Надежность работы** выражается вероятностью того, что в k произвольных случаях работы программа вернет корректный результат. Эта величина оценивается

через выполнение некоторого числа запусков программы (*N*) и вычисления числа случаев успешной работы (*S*). Вероятность успеха, таким образом, вычисляется как *S*/*N*, а вероятность возможности отработать *k* раз успешно — как произведение вероятностей каждого успешного запуска, то есть [S/N]×[S/N]×… ×[S/N], или [S/N]^k. Входные данные для каждого случая выбираются произвольно и независимо от предыдущего запуска.

- **IEEE 20. Среднее время обнаружения k отказов**. Это значение вычисляется аналогично надежности работы (см. IEEE 18 выше).
- **IEEE 22. Оценка числа оставшихся отказов** (методом засева). Эта оценка получена путем «засеивания» в программу *N* произвольных отказов. Если *s* — число найденных засеянных отказов, а *f* — число других отказов, найденных за тот же период тестирования, оценка равна $f \times N / s$.
- **IEEE 24. Охват теста** = [[Число реализованных требований] / [Число требований]] × [[Число протестированных программных примитивов] / [Общее число примитивов в программе]] × 100. Это число оценивает законченность выполненного тестирования. Программные примитивы — это тестируемые модули программы. Сюда относятся методы, классы и пакеты.
- **IEEE 30. Среднее время наработки на отказ** (MTTF — Mean-Time-to-Failure). Измеряется посредством запоминания промежутков времени между всеми парами замеченных последовательных ошибок и их усреднения. При измерении промежутков обычно используется фактическое истекшее время, а не время центрального процессора.
- **IEEE 36. Точность тестирования** = f/N, где N — это число засеянных отказов, а f — это число засеянных ошибок, найденных во время тестирования. Этот показатель оценивает надежность процесса тестирования и представляет собой побочный продукт описанной выше метрики 22.

5.13. Инспектирование и обзоры

5.13.1. Инспектирование и обзоры

Инспектирование — это техника «белого ящика» для обеспечения качества. Инспектирование состоит в проверке частей проекта (требований, результатов проектирования, программного кода и т. п.) на

наличие дефектов. Инспектирования проводит группа коллег автора работы. Мы советуем начинать инспектирование очень рано, так как оно должно начинаться вместе с появлением первой документации по проекту. Поскольку инспектирование изначально было предложено для улучшения качества программного кода, то его часто называют *инспектированием кода*, однако было показано, что инспектирование достигает наилучших результатов, если начинается как можно раньше, еще задолго до появления текста программ.

Основная идея инспектирования хорошо описана Фейганом (M. Fagan), который заметил, что автор в большинстве случаев способен исправить дефект своей работы, когда тот обнаружен. Таким образом, инспектирование необходимо использовать хотя бы для того, чтобы автор замечал дефекты в своей работе до того, как представит ее для использования другими. Подразумевается, что инспектирование выполнятся коллегами по разработке.

Принцип инспектирования может быть обобщен четырьмя правилами.

- **Вскрытие дефектов**. Из инспектирования намеренно исключается исправление дефектов. Процесс исправления предоставлен автору. Во время инспектирования ни минуты времени не должно быть потрачено на обсуждение способов исправления дефекта. Все обсуждения должны проходить после окончания инспектирования.
- **Участие коллег**. Инспектирование проводится внутри группы разработчиков программного обеспечения и не предполагает вовлечения отношений начальник — подчиненный. Инспектируется текущая работа, а не способности ее автора. Автор несет ответственность только за конечный продукт, тогда как инспектирование проводится до того, как работа сдана. Однако работа, представленная автором для инспектирования, должна быть лучшим вариантом, но ни в коем случае не черновиком. Было бы пустой тратой ресурсов группы искать, находить и описывать дефекты, которые автор, приложив некоторые усилия, в состоянии найти сам.
- **Распределение ролей**. Каждый из участников проекта берет на себя одну из следующих ролей. При нехватке кадров один человек может выполнять сразу две роли. Обычно ведущий может одновременно быть и секретарем. Однако для достижения беспристрастной проверки автор не должен выполнять никаких других ролей.
    - *Ведущий* ответственен за правильное проведение инспектирования. Ведущий также является инспектирующим.
    - *Автор* несет ответственность за свою работу и за исправление всех найденных в ней дефектов. Автор является одновременно и инспектирующим.

- *Секретарь* отвечает за учет описания и классификацию дефектов, как это принято в команде. Секретарь также участвует в инспектировании.

- **Тщательная подготовка**. Участникам инспектирования необходимо подготовиться к нему так же детально, как и самому автору. Инспектирования не являются просто посиделками, докладами руководству или образовательными семинарами. Инспектирующие должны работать на том же уровне детализации, что и автор.

В процессе разработки инспектирование должно быть начато как можно раньше. Например, сразу же должны подвергнуться инспектированию требования к приложению. Аналогично, могут быть проверены планы управления проектом и конфигурациями.

Для проведения инспектирования требуется выполнение следующих шагов.

1. **Планирование**. Планирование включает в себя выбор инспектируемой части работы, назначение инспектирующей группы, выбор метрик, по которым будет проводиться инспектирование, а также выбор инструментов для сбора и анализа полученных данных.
2. **Подготовка**. Инспектирующие проверяют работу в полном объеме на своих рабочих местах (например, проверяют, что инспектируемый программный код соответствует детальному проекту). Инспектирующие обычно заносят все обнаруженные дефекты в базу данных (например, доступную через сеть) вместе с описаниями и классификацией. Это помогает избежать дублирования и минимизирует время на собрания. Некоторые предпочитают фиксировать дефекты на бумаге, другие считают информативной метрикой количество инспектирующих, обнаруживших данный дефект.
3. **Инспекционное собрание**. Как только каждый из участников процесса готов, проводится *инспекционное собрание*, в ходе которого участники выполняют свои роли, проводится обсуждение всего материала и все замеченные дефекты фиксируются.
4. **Доработка**. Как правило, автор в состоянии исправить сам все дефекты. Это называется фазой *доработки*. Если инспекционное собрание решает, что дефекты настолько серьезны, что требуется повторное инспектирование данного объекта, то объект еще раз запускается в процесс.
5. **Заключительное совещание**. Когда дефекты исправлены, проводится короткое совещание с участием автора, на котором убеждаются в том, что все дефекты исправлены. Однако это не предполагает детальной повторной ревизии всей работы. Все исправления остаются на совести автора, ответственного за свою работу.

6. **Совещание по улучшению**. Как и после других процессов, группа обычно встречается для обсуждения самого процесса инспектирования и решает, как он может быть улучшен.

В различных источниках отмечается, что инспектирование, несмотря на большие затраты времени экспертов и дороговизну проведения, вполне окупает себя. Если начать подсчитывать стоимость инспектирования, то можно схватиться за голову, однако всегда следует сравнивать эту стоимость со стоимостью пропущенных дефектов. В конечном итоге, все дефекты должны быть найдены и исправлены. Если это не делается вскорости после того, как дефект был внесен, то стоимость исправления катастрофически возрастает. В книге Брауде приведены данные по сравнению стоимости обнаружения и исправления дефекта во время инспектирования и во время интеграции (сборки и совместного тестирования всех частей), которые воспроизведены в табл. 1.

Таблица 1. Оценка количества времени, затрачиваемого на один дефект

	Дефект, найденный во время инспектирования	Дефект, найденный во время интеграции
Количество часов на отыскание	0,7–2	0,2–10
Количество часов на исправление	0,3–1,2	9+
Всего:	1,0–3,2	9,2–19+

Помимо инспектирования, многие организации применяют *обзоры*. Обзоры — это собрания, на которых обсуждается как уже завершенная, так и текущая работа. Примером тому является обзор, на котором может обсуждаться альтернативная архитектура приложения. Хотя обзоры являются неотъемлемой частью проекта, они не требуют такой детальной подготовки, как инспектирование. К тому же участникам обзоров, в отличие от ситуации с инспектированием, не надо играть никаких ролей. Тем не менее, обзоры необходимо проводить, причем регулярно с участием максимального возможно числа разработчиков. Обзоры задают «пульс» проекта, подтверждают его жизнеспособность и служат фактором положительной мотивации персонала.

Регулярные обзоры — это самое надежное средство добиться того, чтобы все были «в курсе» и могли избегать ошибок, порождаемых непониманием общего контекста проекта.

5.14. Выводы

Изучая раздел “Хорошие практики управления качеством процесса разработки ПО. ITIL и ITSM”, мы пришли к выводам о том, что рекомендуется применять:

- Мероприятия, направленные на контроль и обеспечение качества артефактов проекта
- Правила хорошего стиля программирования
- Стандартизация при написании кода программного обеспечения
- Упорядоченное назначение имен: классов, библиотек классов, методов класса, функций, аргументов функций и методов класса, переменных , свойств класса, указателей, ссылок, глобальных переменных, статических переменных, констант, макроопределений #define, констант const, перечислений enum, типов данных, меток, файлов.
- Форматирование текста программного кода: общие требования, максимальная длина строки, использование для организации отступов табуляции и пробелов, пунктуация, выравнивание блока объявлений, расстановка фигурных и круглых скобок. Форматирование операторов if-else, switch-case. Применение комментариев, а также методов и функций и многое другое.
- Документирование кода программы.Тестирование. Процесс контроля качества. Методы «белого ящика» и «черного ящика».Модульное тестирование . Разбиение на классы эквивалентности для тестирования «черного ящика».
- Использование в тестировании ПО случайных величин, планирование модульных тестов, интеграция и системное тестирование, dВерификация, валидация и системное тестирование, процесс интеграции.Тестирование удобства и простоты использования, регрессионное тестирование, приемосдаточное тестирование, тестирование инсталляции.. Альфа- и бета-версии, критерии завершения интеграции и тестирования, документирование интеграции и тестирования, назначение и использование метрик для интегрального и системного тестирования.
- Инспектирование и обзоры.

5.15. Вопросы и задания для самостоятельной работы студента по теме «Мероприятия, направленные на контроль и обеспечение качества артефактов проекта»

Сформулируйте Ваше определение понятий: «правила хорошего стиля программирования, стандартизация, имена, форматирование исходного кода, документирование исходного кода, операторы, мобильность кода, правильная организация программ, основы тестирования, процесс контроля качества, методы «белого ящика» и «черного ящика», интеграция и системное тестирование, документирование интеграции и тестирования, инспектирование и обзоры»

Литература по теме «Мероприятия, направленные на контроль и обеспечение качества артефактов проекта»

[1] Брауде Э. Технология разработки программного обеспечения. СПб, Питер, 2004.
[2] http://www.intuit.ru/ Электронные лекции Андрея Николаевича Терехова

Тема 6. Управление программными проектами

6.1. Стандарты CMM/CMMI

Неудачи при разработках программного обеспечения, к которым относятся невыполнение исполнителями работ в определенные сроки, наличие в проекте множества ошибок реализации, высокая трудоемкость при интеграции разработанного ПО с программными системами заказчика, а то и просто невыполнение работ в полном объеме, всегда связаны с допущенными ошибками при управлении проектом. К ошибкам неудачного проектирования ПО можно отнести, плохое планирование, использование недостаточно проверенных технологий, ошибки при управлении персоналом и недостаточная квалификация коллектива разработчиков.

Для упорядочения процессов производства ПО менеджеры компаний-разработчиков внедряют и применяют различные методики управления предприятием, в том числе и при управлении проектами. Процесс разработки проекта ПО выполняется в виде этапов работ, связанных с программной инженерией (проектирование, программирование, тестирование, документирование, внедрение и сопровождение). Управление проектом разработки ПО заключается в планировании и управлении процессами программной инженерии, для достижения целей проекта, в соответствии с качественными показателями разработки. В книге “Управление программным проектом на практике” [23], Панкаж Джалота (Pankaj Jalote) описывает процесс производства ПО так: ”…Что такое процесс? Технически процесс для задачи включает последовательность шагов, которой нужно придерживаться при выполнении данной задачи. Однако для любой организации процессы, которые она рекомендует использовать своим инженерам и менеджерам проектов,— не просто последовательности шагов: они заключают в себе то, что инженеры и менеджеры проектов узнали об успешно выполненных проектах. Через процессы преимущества накопленного опыта передаются всем сотрудникам организации, в том числе новичкам. Такие процессы помогают менеджерам и инженерам подражать прошлым успехам и избегать просчетов, ведущих к провалам»... ”.

Для управления проектами в области разработки программного обеспечения разработаны международные стандарты:

- ISO 12207 в части определения процессов жизненного цикла и их характеристик;
- серия стандартов ISO 9000:2000 в части управления качеством разработки;

- группа стандартов CMM/CMMI в части определения уровня зрелости организации-разработчика в отношении способности разработки качественного программного обеспечения;
- ISO/IEC15504 (SPICE) в части самооценки организации-разработчика.

К наиболее удачным стандартам относят группу стандартов CMM (Capability Maturity Model) и CMMI (Capability Maturity Model Integration (CMMI) for Systems Engineering / Software Engineering / Integrated Product and Process Development), а также стандарт ISO/IEC 15504 (SPICE).

Зрелость организации является основным понятием группы стандартов CMM/CMMI. Зрелой организацией называется такая организация, в которой имеются четко определенные процедуры для создания продуктов ПО и процедуры по управлению проектами. Имеющиеся процедуры уточняются и совершенствуются в “пилотных” проектах или с помощью анализа себестоимости разработки. Накопленный опыт в зрелой компании позволяет выполнять анализ себестоимости и определять сроки выполнения работ. В зрелой компании существуют стандарты на процессы разработки программной инженерии. Инфраструктура и корпоративная культура зрелой компании направлены на процессы деятельности всей компании по разработке и сопровождению проектов ПО. В зрелых организациях выполнение проектов производится в соответствии с разработанными процессами, что позволяет получать результаты выполнения проектов, зависящие от технологии прохождения процесса. Таким образом, зрелые процессы позволяют точнее планировать результаты и осуществлять контролирование проектов.

Незрелость организации, в части управления процессами разработки ПО, означает, что проекты выполняются без необходимых правил и методик, а результаты проекта в значительной степени зависят от возможностей команды и менеджера проекта.

Основная идея стандарта состоит в использования модели CMM (Capability Maturity Model — модель зрелости возможностей), состоящей из пяти уровней зрелости компании. Деление на уровни позволяет последовательно внедрять CMM, используя стандарт в качестве руководства, которое может обеспечить постоянное совершенствование процесса разработки.

Система понятий стандарта CMM содержит ключевые элементы процессов разработки ПО на разных уровнях зрелости организации. Перед организацией, как бы, открывается путь к достижению более высокой зрелости, и этот путь включает пять уровней, которые представляют собой точно определенные стадии, или плато, которые в CMM называются уровнями зрелости (maturity levels). Каждый уровень зрелости определяет конкретные характеристики процессов разработки

ПО, и более высокие уровни зрелости обладают более развитыми характеристиками, свойственными более зрелым процессам.

Рисунок 34. Уровни зрелости организации по стандарту CMM

Соответствие компании параметрам конкретного уровня модели CMM позволяет проводить сравнение с другими возможными уровнями развития этой компании сравнивать по уровням.

Таблица 2.

Список уровней зрелости компании, в соответствии с положениями модели CMM

№ уровня	***Название уровня***	***Ключевые области процесса***
1	Начальный	Если организация находится на этом уровне, то ключевых областей процессов для нее не предусмотрено
2	Повторяющийся	Управление программными конфигурациями. Обеспечение качества программных продуктов. Управление контрактами подрядчиков. Контроль за ходом проектов. Планирование программных проектов. Управление требованиями
3	Определенный	Экспертные оценки. Координация взаимодействий проектных групп. Инженерия программного продукта. Комплексный менеджмент ПО. Программа

		обучения персонала. Определение организационного процесса. Область действия организационного процесса
4	Управляемый	Менеджмент качества ПО. Управление процессом на основе количественных методов
5	Оптимизируемый	Управление изменением процесса. Управление технологическими изменениями. Предотвращение дефектов

Ожидаемые результаты проекта, когда он выполняется с использованием процесса, есть устойчивость его процесса (process capability). Фактические результаты в проекте, выполнявшемся с использованием процесса, это производительность его процесса (process performance). Таким образом производительность процесса зависит от его устойчивости. Для улучшения производительности процессов, необходимо повышать устойчивость процессам, а сам процесс должен становиться более зрелым.

С некоторыми свободно распространяемыми материалами по CMM можно познакомиться на сайте Software Engineering Institute (http://www.sei.cmu.edu/cmm).

Начальный уровень (initial level). Выполнение проекта управляемо посредством распределения ролей (разработчики и менеджер проекта). Начальный уровень описан в стандарте в качестве основы для сравнения со следующими уровнями. В организации начального уровня не существует стабильных условий для созданий качественного программного обеспечения. Результат любого проекта целиком и полностью зависит от личных качеств менеджера и опыта программистов, причем успех одного проекта может быть повторен только в случае назначения тех же менеджеров и программистов на следующий проект. Более того, если такие менеджеры или программисты уходят с предприятия, то с их уходом резко падает качество производимых программных продуктов. Чаще всего процесс разработки сводится к написанию кода и его минимальному тестированию.

Повторяющийся уровень (repeatable level).

Что бы организации достигнуть повторяющегося уровня CMM, в ней необходимо внедрить технологии управления проектами. Планирование проектами в зрелой организации основывается на накопленном опыте, в ней действуют стандарты на разрабатываемое ПО и функционирует специально созданная группа для обеспечения качества. Кроме того, зрелая организация способна взаимодействовать с субподрядчиками на тех же принципах.

Определенный уровень (defined level).

Для достижения организацией уровня “Определенный”, необходимо внедрить документированный (задокументированный) стандартный процесс создания и сопровождения программного обеспечения, который включает разработку ПО и управление проектами. На определенном уровне (в процессе стандартизации) происходит переход на наиболее эффективные практики и технологии. Создание и поддерживание стандарта CMM осуществляет созданная в организации специальная группа. Обязательным условием для достижения данного уровня является наличие на предприятии программы постоянного повышения квалификации и обучения сотрудников. На данном уровне CMM организация не зависит от качеств конкретных разработчиков. В ней также практически отсутствуют риски по дисквалификации данного уровня (скатывания на уровень ниже).

Управляемый уровень (manager level).

Что бы организации достигнуть “Управляемого” уровня, в ней устанавливаются количественные показатели качества на разрабатываемые виды продукции ПО и на производимые продукты в целом. Достигаемые на данном уровне CMM в организации результаты управления проектами выполняются за счет уменьшения отклонений значений параметров различных показателей проекта от заданных.

Оптимизируемый уровень (optimizing level).

На данном уровне развития CMM, в организации внедряются мероприятия по улучшению, как к существующим процессам, так и для оценки эффективности ввода новых технологий. Главной задачей всей организации на этом уровне CMM является постоянное улучшение существующих процессов. Улучшение процессов нацелено на предотвращение возможных ошибок или дефектов. Для уменьшения стоимости разработки программного обеспечения, применяются технологии повторного использования компонентов ПО.

В процессе сертификации организации проводится оценка соответствия всех ключевых областей по 10-балльной шкале. Для успешной квалификации данной ключевой области необходимо набрать не менее 6 баллов. Оценка ключевой области производится по следующим показателям:

- Заинтересованность руководства в данной области (планируется ли практическое внедрение данной ключевой области, существует ли понимание у руководства необходимости данной области и т. д.);
- Насколько широко данная область применяется в организации (например, оценке 4 балла соответствует фрагментное применение);
- Успешность использования данной области на практике (например, оценке 0 баллов соответствует полное отсутствие какого-либо эффекта, а оценка 8 баллов выставляется при

наличии систематического и измеримого положительного результата практически по всей организации).

Сертифицирование организации может касаться одного процесса или подразделения организации.

Стандарт CMM является собственностью Software Engineering Institute. Оценка качества процессов организаций может проводиться только специалистами, прошедшими специальное обучение и аккредитованными SEI. Разработка стандарта CMM выполняется самим институтом, без заметного влияния на нее остальной части программистского сообщества.

Для преодоления ряда проблем, обнаруженных в связи с использованием стандарта CMM был создан стандарт SEI – Capability Maturity Model Integrated (CMMI – Интегрированная модель оценки уровня зрелости процессов разработки). Стандарт CMMI создавался с целью объединения существующих вариантов стандарта CMM и исключения обнаруженных противоречий при практическом применении в различных сферах деятельности высокотехнологичных компаний. Для устранения необходимости «выравнивания» процессов организации и максимального учета ее деловых потребностей, стандарт CMMI имеет две формы представления – классическую (многоуровневую и соответствующую CMM), и новую, непрерывную. Непрерывная форма представления рассматривает не уровни зрелости (Maturity Levels), а уровни возможностей (Capability Levels), которые оцениваются для отдельных областей процессов (Process Areas, PA).

Таблица 3.

Соответствие уровней CMM/CMMI [24]

№ уровня	***Уровень зрелости CMM***	***Уровень зрелости многоуровневого представления CMMI***	***Уровень возможностей непрерывного представления CMMI***
0	-	-	Незавершенный
1	Начальный	Начальный	
2	Повторяющийся	Управляемый	Управляемый
3	Определенный	Определенный	Определенный
4	Управляемый	Управляемый количественно	Управляемый количественно
5	Оптимизируемый	Оптимизируемый	Оптимизируемый

6.1. Стандарт SPICE

Аббревиатура SPICE раскрывается как Software Process Improvement and Capability dEtermination. Основные цели применения стандарта SPICE таковы:

- удовлетворение растущих потребностей в оценке возможностей процессов производства программного обеспечения в подразделениях;
- гармонизация методов и моделей, используемых для оценки процессов.

Проект SPICE был начат ISO в июле 1991 года и к настоящему времени объединил лучшие из существующих в мире практик. В основу SPICE легли такие стандарты, как:

- STD (Compita);
- CMM (SEI);
- TRILLIUM (Bell);
- SQPA (HP);
- BootStrap (Esprit);
- HealthCheck (BT);
- ISO 9001;
- SAM (BT).

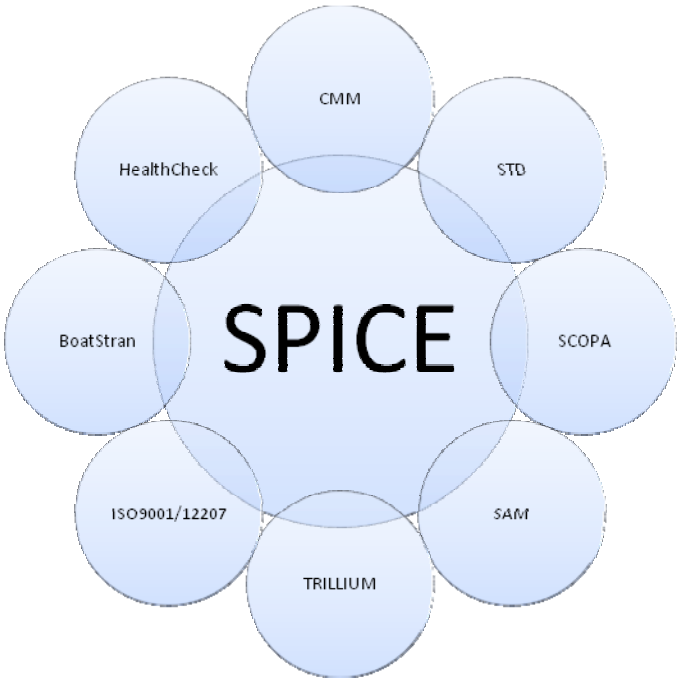

Рисунок 35. Стандарты, повлиявшие на разработку SPICE

В отличие от одномерной модели CMM, архитектура стандарта SPICE - двумерна и состоит из так называемых «уровней возможностей».

В стандарте SPICE насчитывается 6 уровней возможностей (плюс 9 атрибутов процессов и 32 правила менеджмента), 5 категорий процессов и 29 типовых процессов, а также более 200 наилучших известных практик (best known practices).

Основной идеей SPICE является оценивание возможностей не одним числом, как в CMM, а путем построения, так называемого *профиля организации*. Сама организация может проанализировать свою деятельность, выявить, какие из наилучших практик в ней применяются, а какие нет, и построить профиль — кривую, наглядно показывающую, в каких областях деятельности организация сильна, и где в ней узкие места. Наличие масштабируемого механизма самооценки является важнейшим достоинством SPICE.

Таблица 4.

Список уровней возможностей модели SPICE

Уровень	Управляемый процесс
Уровень 0	**Процесс не выполняется**
Уровень 1	**Выполняемый процесс**
1.1	Измерение производительности процесса
Уровень 2	**Управляемый процесс**
2.1	Управление производительностью
2.2	Управление созданием продуктов
Уровень 3	**Установленный процесс**
3.1	Документирование процесса
3.2	Отслеживание ресурсов процесса
Уровень 4	**Предсказуемый процесс**
4.1	Измерение процесса
4.2	Управление процессом
Уровень 5	**Оптимизирующий процесс**
5.1	Изменение процесса
5.2	Постоянное совершенствование

SPICE предлагает достаточно подробную модель, предоставляющую пользователям свободу в выборе путей к улучшению работы. Модель улучшения процессов в SPICE трехмерна, где по одной оси откладывается Эффективность работы (удовлетворенность заказчиков, качество продукции и продуктивность), по другой — Возможности персонала, и по третьей — Адекватность процесса. Таким образом, можно выбирать траекторию улучшения процесса в трехмерном пространстве, где улучшения по каждой из осей идут параллельно с улучшениями по другой. Собственно, параллельность не является требованием, это, скорее, рекомендация, позволяющая избежать серьезных перекосов в процессе производства.

Несмотря на то, что стандарт SPICE вобрал в себя все самое лучшее из целого ряда других стандартов, он не стал простым их объединением. Для того чтобы показать, чем же SPICE отличается от своих предшественников, целесообразно провести сопоставление SPICE и других наиболее известных стандартов.

Таблица 5.

Сравнение SPICE и ISO 9001

SPICE	ISO 9001
Развитая документация	Малая документация
Детальная модель	Абстрактная модель
Разработан для производства ПО	Разработан для производства в целом
Улучшение процессов и оценка возможностей	Сертификация
Требования к самооценке, руководство по применению	Не содержит
Дополняет ISO 9001	Дополнятся SPICE

Таблица 6.

Сравнение SPICE и CMM

SPICE	CMM
Двумерная структура	Последовательная, одномерная структура
Допускает гибкость в выработке стратегии улучшения	Содержит предопределенный путь развития
Уровни возможностей для каждого процесса	Единый уровень зрелости для всех процессов
Результаты требуют упрощения	Результаты легко понимаемы
Результаты очень подробные	Упрощенные результаты

6.2. Выводы

Изучая раздел “Управление программными проектами”, мы пришли к выводам о том, что:

- рекомендуется применять стандарты CMM/CMMI и SPICE.
- IT-организация может быть зрелой и незрелой и что степень зрелости можно измерить.

6.3. Вопросы и задания для самостоятельной работы студента по теме «Управление программными проектами»

Сформулируйте Ваше определение понятий: «правила хорошего стиля программирования, стандартизация, имена, форматирование исходного кода, документирование исходного кода, операторы, мобильность кода, правильная организация программ, основы тестирования, процесс контроля качества, методы «белого ящика» и «черного ящика», интеграция и системное тестирование, документирование интеграции и тестирования, инспектирование и обзоры»

Литература по теме «Управление программными проектами»

[1] Брауде Э. Технология разработки программного обеспечения. СПб, Питер, 2004.

[2] http://www.intuit.ru/department/se/introprogteach/
Электронные лекции Андрея Николаевича Терехова

Предметный указатель

К

Л

М

Н

О

П

Ц

Ш

Э

[1] Ильин А.А., Спасение в качестве. "Русский колокол". 1928. № 4. С. 3 – 4

[2] Энгельс Ф., см. Маркс К. и Энгельс Ф., Соч., 2 изд., т. 20

[3] Мищенко С.В., Пономарев С.В., Пономарева Е.С., Евлахин Р.Н., Мозгова Г.В., ИСТОРИЯ МЕТРОЛОГИИ, СТАНДАРТИЗАЦИИ, СЕРТИФИКАЦИИ И УПРАВЛЕНИЯ КАЧЕСТВОМ, Тамбов: Изд-во Тамб. гос. техн. ун-та, 2004 г.

[4] Момот А.И. "Менеджмент качества: Учебное пособие для вузов". Донецк: ДонГТУ, 2000 г.

[5] Новоторговый устав (Текст), http://school-collection.informika.ru/catalog/res/7A9A40A9-0A01-01B2-0125-EE49A0C34B8B/view/

[6] Хейли Артур. Колеса. АСТ, 2003 г.

[7] Taylor, F. W. (1911) The Principles of Scientific Management, Harper & Row, New York. Смотрите также http://marxists.nigilist.ru/reference/subject/economics/taylor/principles/ch01.htm и http://orel.rsl.ru/nettext/foreign/teilor/tailsod.htm

[8] Терещенко В.И., Организация и управление опыт США, Изд-во: Экономика, 1965 г.

[9] Адлер Ю. П., Е. Хунузиди И., Шпер В. Л., Методы постоянного совершенствования сквозь призму цикла Шухарта-Деминга, "Методы менеджмента качества", №3, 2005 г.

[10] Reclusive Linux founder opens. Friday, May 19, 2006 Posted: 0954 GMT (1754 HKT. http://edition.cnn.com/2006/BUSINESS/05/18/global.office.linustorvalds/

[11] Качество в XXI веке. Роль качества в обеспечении конкурентоспособности и устойчивого развития: пер. с англ. / ред.-сост. Конти Тито, Кондо Ёсло, Ватсон Грегори. - М.: Стандарты и качество, 2005 г.

[12] Лапидус В.А.. "Всеобщее качество (TQM) в российских компаниях". ОАО Типография Новости, 2002 г.

[13] Пэнди Питер С., Ньюмен Роберт П., Кэвенег Роланд Р., Курс на Шесть Сигм: Как General Electric, Motorola и другие ведущие компании мира совершенствуют свое мастерство, "ЛОРИ", 2002 г.

[14] Хамханова Д.Н. ОСНОВЫ КВАЛИМЕТРИИ. Учебное пособие для студентов специальностей 190800 «Метрология и метрологическое обеспечение», 072000 «Стандартизация и сертификация (по отраслям пищевой промышленности)» и 340100 «Управление качеством». Издательство ВСГТУ Улан-Удэ, 2003 г.

[15] ГОСТ Р ИСО 9000-2001

[16] Липаев Владимир, г.н.с. ИСП РАН. Оценка качества программных средств. ”Сетевой журнал” №3.2002 г.

[17] ГОСТ Р ИСО/МЭК 9126-93

[18] ГОСТ Р ИСО/МЭК 12207-99

[19] Седов Олег. "Управление ITIL", Computerworld Россия, 07.2007 г.

[20] Книга Форума itSMF. "Введение в ИТ Сервис-менеджмент", под редакцией Потоцкого М.Ю. (перевод на рус. язык). Открытые Системы. ISBN:9077212159

[21]http://www.microsoft.com/technet/solutionaccelerators/cits/mo/mof/default.mspx

[22] http://www-306.ibm.com/software/awdtools/rup/support/

[23] Джалота Панкаж. Управление программным проектом на практике. "ЛОРИ", 2005 г.

[24] Колдовский Вячеслав. Разработка ПО: стандарты качества, http://manager.net.ua/index.php?option=com_content&task=view&id=1079&Itemid=52