МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ГОУ ВПО «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ИНЖЕНЕРНО-ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

на правах рукописи

КОЛТУНОВА ЕКАТЕРИНА ВЛАДИМИРОВНА

ВЫБОР МЕТОДОВ, МОДЕЛЕЙ И СТАНДАРТОВ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Специальность 08.00.13 – математические и инструментальные методы экономики

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата экономических наук

С-Петербург

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ **4**

ГЛАВА 1. МЕТОДИЧЕСКИЕ ОСНОВЫ И ОСОБЕННОСТИ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПО В СПЕЦИАЛИЗИРОВАННЫХ ПС **13**

1.1. Эволюция и современное состояние теории управления разработкой программного обеспечения 13

1.2. Основные проблемы выбора методов, моделей и стандартов управления разработкой ПО 16

1.3. Классификация инструментальных средств управления разработкой ПО 20

1.4. Концепции теории управления в методах, моделях и стандартах 28

ГЛАВА 2. АНАЛИЗ МОДЕЛЕЙ И СТАНДАРТОВ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ **54**

2.1. Анализ структуры методологий и стандартов управления компанией 55

2.2. Модели жизненного цикла ПО в методологиях и стандартах 62

2.3. Структура проектных методологий 70

2.4. Анализ методов и инструментальных средств получения количественных оценок 73

ГЛАВА 3. ОПТИМИЗАЦИЯ ВЫБОРА И АДАПТАЦИИ МЕТОДОВ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ **86**

3.1. Организация процессов выбора 86

3.2. Влияние внутренних особенностей организации и воздействия внешней среды на выбор 88

3.3. Выбор с использованием метода верификации принципов 104

3.4. Выбор моделей и стандартов для создания системы менеджмента качества 107

3.5. Выбор системы метрик в управлении разработкой ПО 109

3.6. Информационное и техническое обеспечение выбора 126

3.7. Референтные модели компании-разработчика программного обеспечения 137

ЗАКЛЮЧЕНИЕ **171**

ОСНОВНЫЕ ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИ **174**

ЛИТЕРАТУРА **176**

ГЛАВА 4. ПРИЛОЖЕНИЯ **192**

4.1. Управление проектами: определения и структура 192

4.2. Терминология в области качества 194

4.3. Управление процессами: термины и определения 195

4.4. Модели жизненного цикла 196

4.5. Модели процессов 200

4.6. Модели рисков 206

4.7. Структуры методологий 207

4.8. Выбор методологий 212

4.9. Метрики 213

ИЛЛЮСТРАЦИИ

Рис. 1. Классификация методов, моделей и стандартов разработки программного обеспечения 25
Рис. 2. Классификация программного обеспечения 99
Рис. 3. Модель программы использования метрик SATC [265, стр. 2] 112
Рис. 4. Структура документации компании – разработчика ПО 129
Рис. 5. Классификация внутрифирменных стандартов 129
Рис. 6. Нотация eEPC (использованные элементы) 139
Рис. 7. Модель структуры знаний компании 140
Рис. 8 Карта знаний 141
Рис. 9. Модель организационной структуры 142
Рис. 10 Дерево функций (структура процессов) 144
Рис. 11. Процесс анализа 146
Рис. 12. Процесс проектирования 148
Рис. 13. Процесс разработки (общий) 152
Рис. 14. Собственный процесс разработки 153
Рис. 15. Процесс тестирования 158
Рис. 16. Процесс внедрения 160
Рис. 17. Процесс документирования 162
Рис. 18. Процесс управления проектом 167
Рис. 19. Процесс управления качеством 170
Рис. 20. Структура SPICE 211
Рис. 21. Выбор методологий семейства Crystal 213
Рис. 22. Модель оценки проекта 215

ТАБЛИЦЫ

Табл. 1. Специфика ИТ проектов и инструментальные средства 29
Табл. 2. Концепция управление проектами в специализированных моделях и стандартах 30
Табл. 3. CMMI. Области процессов управления проектами 31
Табл. 4. Сравнение характеристик качества ISO/IEC 9126 и Б. Боэм 33
Табл. 5 Анализ проектов с использованием данных метрик 75
Табл. 6. Классификация метрик 76
Табл. 7. Метрики: объект измерения и измеряемые характеристики 77
Табл. 8. Принципы разработки ПО 105
Табл. 9. Сравнительный анализ стандартов ISO 9000, SPICE и CMM 107
Табл. 10. Факторы масштабирования проекта 216

ВВЕДЕНИЕ

Актуальность исследования. Непрерывное изменение условий функционирования современных организаций, ускорившееся в ходе формирования информационной экономики, обусловило потребность в использовании новых концепций и методов управления. Развитие теории управления, создание новых методов, моделей, методик и других практических инструментов, определило задачу их эффективного выбора и адаптации. Специфика управления предпринимательскими структурами (ПС) в различных отраслях экономики требует адаптированных подходов и методов управления.

Наша работа посвящена решению задачи эффективного использования специализированных методов и моделей для **управления разработкой программного обеспечения**. В работе предлагаются новые подходы к решению задачи выбора адекватных методологий и стандартов, а также представлена их новая классификация, раскрывающая связь специализированных и общих методов, моделей и стандартов управления.

Разработка программного обеспечения – растущая отрасль, влияние которой на глобальную экономику существенно превосходит ее долю в мировом производстве. Использование программного обеспечения оказывает влияние на производительность труда во всех отраслях, информационные технологии являются основным катализатором управленческих инноваций, обеспечивающих экономический рост [179, 255].

Программное обеспечение является ядром технологий, формирующих информационное общество. Информационное общество определяется как общество, в котором процесс компьютеризации дает людям доступ к надежным источникам информации, избавляет их от рутинной работы, обеспечивает высокий уровень автоматизации производства. При этом продукт становится более «информационно ёмким» (будет содержать в своей стоимости более высокую долю инноваций, дизайна, маркетинга, телекоммуникаций, обработки информации). «…производство информационного продукта, а не продукта материального будет движущей силой образования и развития общества» [254, стр. 2].

Значимость информационных технологий для развития экономики России отражена в Федеральной целевой программе Электронная Россия 2002-2010 годы: «Развитие и широкое применение информационных и коммуникационных технологий (ИКТ) является глобальной тенденцией мирового развития и научно-технической революции последних десятилетий. Применение ИКТ имеет решающее значение для повышения конкурентоспособности экономики, расширения возможностей ее интеграции в мировую систему хозяйства, повышения эффективности государственного управления и местного самоуправления» [193]. Государственные инициативы, направленные на предоставление услуг гражданам и бизнесу с использованием ИТ, представленные в ряде проектов по созданию «Электронного правительства» предполагают существенные инвестиции в проектирование и разработку необходимых программных средств архитектуры их взаимодействия [100].

Анализируя экономический и социальный эффект информационных технологий на жизнь общества и определяя перспективы развития технологий, Д. Мощелла акцентирует внимание на проблеме разработке ПО как на одном из сдерживающих факторов: «Программное обеспечение само по себе следует рассматривать как одно из величайших и в высшей степени сложных достижений человеческого разума. При его создании каждое требующееся от компьютера действие следует преобразовать в набор специфических машинных команд. Не удивительно, что в программирование вовлекается все большая часть совокупного интеллектуального потенциала планеты. ... И сейчас, как и 60 лет назад, компьютерное программирование остается процессом исключительно трудоемким, сложным, ответственным и даже мучительным. … Разговоры о повышении уровня программирования и гарантиях качества конечного продукта ведутся в среди специалистов уже более 40 лет, но они вряд ли приблизили нас к решению проблемы, которая, похоже, пребудет с нами всегда» [146 стр. 37].

Проектный характер разработки ПО и высокий уровень рисков обусловили развитие специализированных проектных методологий, призванных обеспечить качество ПО, повысить производительность труда

разработчиков и снизить проектные риски за счет повторяемости процессов, использования количественных методов оценок и наличия процедур контроля и оптимизации. Это делает необходимым использование специализированных методов управления разработкой, что и обуславливает актуальность выбранной темы.

Актуальность темы исследования подтверждается устойчивой неблагоприятной статистикой выполнения проектов разработки ПО и внедрения ИС. Ежегодные исследования Standish Group показывают сохранение исключительно высокой доли проектов внедрения информационных систем (ИС), завершившихся с превышением сроков и бюджета, а так же неуспешных проектов. Более 30% всех проектов заканчиваются неуспешно, более 50% завершаются с превышением бюджета в два раза, и только 15% проектов выполняются в срок и в рамках бюджета [282].

Таким образом, задачи, связанные с созданием эффективных процессов разработки программного обеспечения, становятся типовыми, а необходимость формализации процессов, их оценки и анализа, требует применения специализированных инструментальных средств.

Существует проблема теоретического обобщения и использования опыта, накопленного в индустрии. Отсутствие структурированного подхода к управлению и использование неэффективных методологий характерно для многих компаний разработчиков ПО. Наша работа призвана содействовать решению обозначенных проблем.

Специфика теории управления разработкой ПО, как части теории управления. В настоящее время сформировалось ядро единой интернациональной теории управления (management theory), которая является центральным объектом изучения в курсах MBA (Master of Business Administration, магистр делового администрирования).

Термин "*Management*" – “управление”, происходящее от латинского *manu agere* "направлять рукой", характеризует процесс руководства всеми частями организации или бизнеса путем распределения и использования различного вида человеческих, финансовых, материальных и нематериальных ресурсов.

Российская школа управления, в отличие от американской (западной) школы, характеризуется универсальным подходом. Поэтому понятие *«управление»* шире, чем «менеджмент», включает управление и техническими, и социальными системами. «*Управлением»* называется воздействие на управляемую систему с целью обеспечения ее требуемого поведения.

Российская школа управления опирается на системный анализ и математические методы, в то время как западная (интернациональная) теория управления представляет собой непротиворечивый набор теорий, среди которых стратегическое управление, управление проектами, управление качеством, управление изменениями, управление финансами, управление маркетингом, логистика и т.д. Принципы и методы, сформулированные в рамках этих теорий, находят отражение в концепциях (Six Sigma, TQM, Business Process Reengineering, Project Management, Balanced Scorecards), стандартах (CobiT, ISO 9000), техниках (Activity Based Costing, Pert/CPM) и т.п.

Идентификация концептуальных основ теории управления в специализированных методологиях управления ИТ-проектами и ИТ-компаниями является исходным пунктом предлагаемого подхода к их классификации и выбору [108].

Технология создания ПО оставляет значительную степень свободы в выборе различных вариантов процесса разработки и подходов к управлению. Проектный характер разработки ПО и высокий уровень рисков обусловили развитие специализированных проектных методологий, а также использование методов управления рисками, призванных обеспечить качество ПО за счет повторяемости процессов, использования количественных методов оценок, наличия процедур контроля и оптимизации.

Состояние изученности проблемы. К настоящему времени накоплен и обобщен большой объем статистических данных по реализованным проектам разработке ПО, а также примененным методам и инструментам управления этими проектами. В то же время из-за отсутствия единой общепринятой системы классификации проектов и инструментальных средств, отсутствует

возможность объединить в единый массив для дальнейшего изучения данные, собранные организациями и отдельными исследователями.

Исследования в области теории управления разработкой ПО проводились в двух основных направлениях. С одной стороны разрабатывались методологии, основывающиеся на концепциях теории управления и представляющие собой ее специализацию с учетом особенностей процессов управления разработкой ПО. В этом направлении ведет работу институт Картеги Мелона (Carnegie Mellon University Software Engineering Institute, CMU SEI), а также ведущие компании-вендоры: IBM, HP, Microsoft, SAP, Oracle. С другой стороны происходит непрерывное теоретическое обобщение практического опыта реализации ИТ-проектов. Именно в этом направлении ведут свою работу многочисленные независимые исследователи-практики. Среди них, например, Ф. Брукс, который одним из первых выполнил анализ деятельности по управлению ИТ-проектами в своей работе «Мифический человеко-месяц». Он определил основные типы команд разработки ПО, выделил командные роли и оценил применимость различных команд к различным типам проектов. А. Коберн предложил структурированный подход к выбору адекватной проектной методологии, предложил набор методологий семейства Crystal и определил область применения этих методологий. Э. Йордон выделил условия применения методологий управления «критическими» проектами. М. Фаулер выполнил сравнительный анализ адаптивных проектных методологий и сформулировал рекомендации по процедуре их выбора.

Разработано большое количество программных инструментальных средств, поддерживающие использование популярных методологий управления ИТ-проектами.

В то же время исследования, направленные на выработку универсальных подходов к управлению ИТ-проектами, объединяющие теоретические и эмпирические методологии, нам не известны, что в свою очередь подтверждает актуальность исследования. Не достаточно разработана теория управления изменениями применительно к практике работы компании-разработчика ПО. Отдельные пионерские работы, посвященные вопросу согласования практики

операционного управления деятельностью ИТ-компании со стратегическим управлением ее деятельностью носят фрагментарный характер.

Цель диссертационной работы заключается в создании научных основ выбора и адаптации методов, моделей и стандартов управления разработкой программного обеспечения и методик формализации и совершенствования процессов разработки в специализированных предпринимательских структурах.

Объектом исследования являются процессы управления разработкой программного обеспечения (ПО), а также специализированные стандарты, модели и методологии, используемые для организации процессов разработки ПО.

Предметом исследования являются принципы и методы управления разработкой программного обеспечения в специализированных предпринимательских структурах.

Теоретическая база. Теоретической базой явились работы по управлению проектами, управлению качеством, проектированию и реинжинирингу бизнес-процессов, стратегическому и операционному менеджменту и маркетингу, а так же теоретические исследования в области управления разработкой программного обеспечения.

Мы использовали теорию конкурентных стратегий, сформулированную М. Портером и теорию управления качеством, разработанную Э. Демингом, Ф. Кросби, К. Исикавой и другими авторами.

Значительный вклад в развитие теории управления разработкой программным обеспечением внесли Б. Боэм, Ф. Брукс, Т. ДеМарко, А. Коберн, Т. МакКейб, В. Липаев, У. Ройс, Р. Томсетт, У. Хамфри, Э. Йордан, М. Фаулер, Дж. Мак-Карти, Л. Константайн, М. Полк, Дж. Рабо, Г. Буч, А. Якобсон, Ф. Крачтен, К. Бек и др.

Крупные компании-разработчики программного обеспечения также внесли значительный вклад в развитие теории управления разработкой ПО. Среди них Microsoft, Oracle, SAP, HP, IBM и многие другие.

Методы исследования. Основными методами исследования являются монографический метод и системный анализ, а также метод экспертных оценок.

Информационная база. Информационной базой работы являются технические отчеты и стандарты, регламентирующие процессы создания и этапы жизненного цикла ПО, требования к качеству ПО, методические разработки, выполненные Институтом программной инженерии (SEI, Software Engineering Institute), комитетами Международной организации по стандартизации (ISO, International Organization for Standardization), комитами института инженеров (IEEE , Institute of Electrical and Electronics Engineers, Inc. Computer Society Board of Governors), ассоциацией вычислительных машин (ACM, Association for Computing Machinery), министерством обороны США (DoD, Department of Defense).

В работе мы использовали так же результаты исследований и статистические данные аналитических агентств IDC, Gartner и Standish Group.

Научная новизна. На основе проведенных исследований были получены следующие результаты, составляющие научную новизну работы:

1. выявлены и развиты положения теории управления разработкой ПО как составной части теории управления, проведен сравнительный анализ и выполнена классификация существующих методов, моделей и стандартов, используемых в управлении разработкой ПО;
2. предложены новые методы выбора метрик, моделей процессов и моделей жизненного цикла, организационных стандартов и проектных методологий управления разработкой ПО: метод верификации принципов, метод анализа компании, учитывающий особенности бизнес-стратегии и культуры предпринимательской структуры, метод анализа используемой технологии проекта;
3. разработаны методики формализации и непрерывного улучшения процессов разработки ПО. Разработана классификация документации типовой компании-разработчика ПО и классификация внутрифирменных стандартов;
4. разработаны новые референтные модели процессов ИТ-компании, осуществляющей разработку заказных бизнес-приложений.

Практическая значимость и апробация работы. Результаты работы могут использоваться компаниями-разработчиками ПО для решения задачи

выбора и адаптации методов, моделей и стандартов управления ИТ-проектами, а так же создания и совершенствования системы управления качеством.

Предлагаемые рекомендации прошли апробацию в компании «Компьютерные Системы для Бизнеса» (CSBI) в 1999-2000 г.г.

Результаты работы докладывались на научно-практической конференции студентов и аспирантов СПбГИЭА в апреле 1999 г, семинаре молодых экономистов, Весенней конференции молодых экономистов 1999 г. СПбГУ, регулярных семинарах для ИТ директоров и руководителей компании Eventum «Разработка и реализация ИТ стратегии» и «Эффективное управление ИТ проектами» в 2004-2005 г.г., семинаре «Информационные технологии для Российской промышленности: новые решения» в марте 2005 г и семинаре «ИТ для современного бизнеса» компании Nowadays в апреле 2005 г. , на научно-практическом семинаре «Выбор методов, моделей и стандартов» в Британском Совете в ноябре 2006 г.

Объем и структура диссертации. Диссертация состоит из введения, трех глав, заключения, библиографического списка и приложений. Общий объем работы составляет 216 страниц, включая 12 таблиц и 28 рисунков.

Во введении определяется предмет исследования и обосновывается актуальность исследования, сформулированы цели и задачи исследования, сформулированы гипотезы, определяющие направления исследования.

В первой главе «Теоретические основы управления разработкой программного обеспечения» приведен анализ эволюции и современного состояния теории управления разработкой ПО. В первой главе выполнен анализ влияния концепций управления на теорию управления разработкой ПО. Там же представлена разработанная классификация методов и моделей управления разработкой ПО.

Во второй главе «Анализ методов, моделей и стандартов управления разработкой программного обеспечения» приведен сравнительный анализ современных инструментальных средств управления разработкой ПО: стандартов этапов жизненного цикла разработки ПО, моделей процессов, систем менеджмента качества, методологий разработки ПО.

Третья глава «Оптимизация выбора и адаптации методов управления разработкой программного обеспечения» содержит научно обоснованные рекомендации по выбору и адаптации существующих методов, моделей и стандартов для управления разработкой ПО. Предложена методика построения эффективных процессов разработки, основанная на теории организационных изменений. Даны практические рекомендации по созданию регулярной процедуры выбора и адаптации инструментальных средств и созданию библиотеки инструментальных средств. Представлены референтные модели процессов ИТ-компании, разработчика заказных информационных систем для бизнеса.

В заключении сформулированы основные выводы и предложения диссертационного исследования.

1. МЕТОДИЧЕСКИЕ ОСНОВЫ И ОСОБЕННОСТИ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПО В СПЕЦИАЛИЗИРОВАННЫХ ПС

1.1. Эволюция и современное состояние теории управления разработкой программного обеспечения

Эволюция походов к управлению разработкой ПО, обусловленная развитием технологий разработки и накоплением опыта, привела к формированию теории управления разработкой ПО в ее современном виде.

Появились коммерческие компоненты повторного использования, операционные системы, базы данных, языки высокого уровня, автоматизированные средства анализа, проектирования и разработки. Индустриальным стандартом стали управляемые и измеряемые процессы. Логика развития программной инженерии, мотивированная, прежде всего, борьбой со сложностью, прошла через серию революционных смен программистских парадигм: алгоритмическая абстракция – абстракция данных – абстракция типов данных привела к объектно-ориентированной парадигме [210]. Появление технологии и стандартов сервисно-ориентированная архитектуры (SOA, Service-oriented architecture) привело к созданию новой группы специализированных методологий [250].

Ключевым фактором, влияющим на развитие теории управления разработкой ПО, является экономический рост и технологическое развитие отрасли. Разнообразие бизнес-моделей предпринимательских структур, разрабатывающих программное обеспечение, обусловило разнообразие подходов к организации производственного процесса и вспомогательной деятельности. Развитие ИТ-компаний, разрабатывающих заказное ПО, рост крупных разработчиков готовых программных продуктов, появление большого числа небольших поставщиков «нишевых» программных продуктов, возникновение «экосистем» программных решений и услуг – выявленные тенденции не только актуализируют проблему выбора адекватных инструментальных средств управления разработкой ПО, но и являются стимулом к дальнейшему развитию теории управления разработкой ПО.

Среди значимых событий в истории развития теории управления разработкой ПО можно выделить следующие события:

- 1968 г. Уинстон Ройс описал две основные модели жизненного цикла разработки ПО: каскадную и итеративную модели [275].
- 1975 г. Фредерика Брукса в своей работе «Мифический человеко -месяц» описал основные проблемы проектов разработки ПО и предложил их решения, он обратил внимание на существенные отличия в производительности отдельных разработчиков и предложил эффективную модель команды проекта [25].
- 1977 Алан Албрехт (IBM) предложил метрику Function Point (Функциональные точки) для оценки объема ПО. Метрика FP позволила количественно оценивать объем работы на ранних стадиях проекта разработки ПО.
- 1979 г. Флетчер Баклей (Fletcher Buckley) из IEEE Computer Society инициировал разработку стандарта обеспечение качества ПО IEEE 730.
- 1979 г. Кристофер Александер разработал теорию «паттернов» (шаблонов программирования) [207].
- 1981 г. Барри Боэм предложил модель COCOMO (Constructive Cost Model, конструктивная модель оценки стоимости) [209]
- 1993 г. Марк Полк с коллегами из университета Карнеги-Мелона разработали модель зрелости технологических процессов CMU SEI CMM и др. [266].
- 1993 г. Кен Швабер и Майкл Бидл предложили методологию быстрой Scrum и подход «Адаптивный (быстрый) процесс быстрой разработки» (Agile Software Development) [277].
- 1994 г. Уотс Хэмфри предложил «Персональный процесс разработки» (PSP, Personal Software process) [241]
- 1994 г. Джим Рабо, Гради Буч и Айвар Якобсон (three amigos & Rutional Software Corporation) объединили свои объектные нотации и создали UML (Unified Modelling language, унифицированный язык разработки) [212]. Сегодня – стандарт ISO/IEC 19501:2005 [246].

- 1995 г. Филипп Крачтен (Rational Software Corporation) разработал «Рациональный унифицированный процесс» (RUP, Rational Unified Process) [122].
- 1996 г. Кент Бек предложил методологию «экстремального программирвания» (XP, eXtreme Programming) [18].
- 1996 г. Роб Томсетт описал основные переговорные модели в ИТ-проектах [287].
- 2001 г. Группа сторонников методологии Agile Software Development Кент Бек, Алистер Коберн, Джим Хайсмит и др. опубликовали манифест быстрой адаптивной разработки ПО «Agile Manifesto» [256].
- 2004 г. М. Кусумано сформулировал бизнес-стратегии компаний-разработчиков программного обеспечения (software company) [219].

Ф. Брукс сформулировал проблему оценки производительности труда в сфере разработки программного обеспечения, и определил основные направления повышения производительности труда разработчиков [177]. Б. Боэм разработал совокупность методов прогнозирования, оценки трудоемкости и длительности проектов разработки программного обеспечения (метрик), получивших название COCOMO (Constructive Cost Model) [209].

М. Полк и команда специалистов SEI разработали модель зрелости технологических процессов (CMM, Capability Maturity Model). Э. Йордон исследовал особенности выполнения проектов, выполняющихся в условиях дефицита ресурсов, а так же специфику проектов, связанных с Интернет [84, 85]. Т. ДеМарко и Г. Вейнберг исследовали проблему влияния межличностных отношений на процессы разработки, разработали эффективные модели управления межличностными взаимодействиями в команде разработки. Л. Константайн системно рассмотрел проблему управления человеческими ресурсами в ИТ-проектах. Дж. Мак-Карти предложил метод выстраивания эффективного диалога в ходе собраний команды ИТ-проекта [140].

У. Ройс выполнил системное исследование методологий управления разработкой программным обеспечением, основанных на рабочих продуктах и выполнил анализ принципов, заложенных в основу различных методологий

[162]. Т. МакКейб разработал методологию тестирования программного обеспечения, основанную на метриках сложности [258]. Дж. Хайсмит и А. Коберн сформулировали принципы быстрой адаптивной разработки [256]. А. Коберн провел сравнительный анализ методологий, основанных на рабочих продуктах, и методологий, основанных на действиях, и определил подходы к выбору методологий и области их применения.

В. Хамфри разработал технику повышения производительности труда программистов и снижения количества ошибок, основанную на самоконтроле и персональном планировании (PSP, Personal Software Process). Р. Томсетт осуществил анализ основных переговорных моделей, используемых в ходе определения требований, сроков и необходимых ресурсов для программных проектов.

Г. Буч, А. Джекобсон и Д. Рамбо разработали унифицированный язык моделирования описания требований и проектных решений для объектно-ориентированного проектирования и программирования, (UML, Unified Modeling Language), а Ф. Кратчен разработал первый итеративный процесс - рациональный процесс разработки (RUP, Rational Unified Process).

Майкл Кусумано выполнил анализ бизнес-стратегий компаний разработчиков и выявил тренд перехода от продуктовой к сервисной модели для крупных компаний.

Разнообразие созданных методик, стандартов и методологий обусловило необходимость создания классификации инструментальных средств.

1.2. Основные проблемы выбора методов, моделей и стандартов управления разработкой ПО

Применение современной теории управления для управления программными проектами и компаниями-разработчиками предполагает решение задачи выбора и адаптации методов, моделей и стандартов.

С проблемой создания эффективной системы управления тесно связаны такие вопросы, как:

- Создание системы обучения менеджеров и разработчиков теории управления разработкой ПО;

- Внедрение в практику работы технических средств, необходимых для управления разработкой ПО;
- Использование согласованного глоссария предметной области, обеспечивающего единство понимания терминов в области управления разработкой ПО;
- Понимание требований внешней среды к организации процессов разработки (например, требования к сертификации по CMM/CMMI со стороны клиентов);
- Создание системы управления использованием инструментальными средствами, включая интеграцию с системой финансового управления компанией. Необходимо планировать расходы на внедрение и использование методов, моделей и стандартов и оценивать их эффективность.

Выбор адекватных проектных методологий, моделей ЖЦ ПО, метрик проектов и других инструментальных средств управления ПО является сложной задачей для компаний-разработчиков ПО и проектных команд. Сложность задачи обусловлена большим количеством инструментальных средств, сложными взаимосвязями между ними, а также непрерывным развитием теории управления ПО.

Подходы к выбору методологий, моделей и стандартов зависят от предполагаемой цели использования. Они могут применяться следующим образом:

- Как эталон (желаемое состояние организации процессов, выполнения работ) «эффективная методология – серебряная пуля»;
- Для обоснования существующей практики управления проектами. Как доказательство (обоснование) правильности принятых решений;
- Как руководство по достижению целей «методология»;
- Как набор требований (ограничений), предъявляемых внешней средой: клиентами, партнерами, собственниками, государственными организациями, профессиональными объединениями - «стандарт».
- Для объяснения (понимания) существующей практики. Используется терминология, концепции, идеи - «модель».

- Для обеспечения «эффективного» диалога между менеджментом, участниками проектных команд, клиентами и субподрядчиками «глоссарий».
- Для нахождения (смены) фокуса – центра приложения управленческих усилий, инвестиций в методологии.

В зависимости от целей использования интерес представляют различные смысловые части методологий, моделей и стандартов. Смысловыми частями являются:

- Терминология и определения;
- Идеи и концепции;
- Справочники и классификации;
- Модели процессов;
- Состав и структура документации;
- Алгоритмы оценки объемов, сложности и других параметров проекта;
- Модели команд;
- Истории успеха, примеры;

Истории успеха и практические примеры, признание в качестве международного стандарта обеспечивают эмоциональную привлекательность рекомендаций, стимулируют процессы обучения.

Обоснование рекомендаций методологии с помощью признанных концепций управления делает методологии полезными для подтверждения правильности существующих практик, совпадающих с рекомендуемыми. Наличие развернутых рекомендаций, включающих описание процессов, моделей команд, требований к рабочим продуктам необходимо для использования методологий в качестве руководства. Глоссарий и классификации используются для выстраивания эффективного диалога в ИТ-проектах.

Цель использования методологии определяет эффективность ее выбора в каждом отдельном случае. Например, CMMI может быть выбрана для целей выстраивания диалога, использования общей терминологии и в качестве «идеала» организации процессов в компании, но не подходить как практическое «руководство к действию» для управления ИТ-проектом. Она носит слишком абстрактный характер.

Предпосылкой успешного решения задачи выбора методологии является наличие необходимой и достаточной информации о ней самой и практике ее использования, и дополнительных рекомендаций по применению.

Постоянное развитие теории управления и совершенствование технологий разработки приводят к выработке новых принципов, на основе которых создаются модели и стандарты. Модель бизнеса компании, ее стратегия, культура компании, особенности выполняемых сегодня проектов (технологии, риски, доступность финансовых ресурсов) определяют выбор инструментальных средств.

Сложность выбора обусловлена:

- большим объемом методологий, сложностью их структуры;
- непрерывным развитием теории управления разработкой ПО, созданием новых методологий, стандартов, моделей, техник;
- отсутствием однозначного критерия выбора методологий;
- объективной необходимостью использования различных методологий для различных проектов в компании.

Для решения задачи эффективного использования существующих методов, стандартов, моделей необходимо выполнить их классификацию, провести сравнительный анализ, определить, какие задачи они решают, какова их область применения.

В настоящее время часто используется слабоструктурированный подход к выбору методологий, при котором неоправданно смешиваются методологии, имеющее различное назначение. Например, в [76] с одинаковых позиций оцениваются проектные методологии и модели зрелости (см. приложение 4.8.1)

В теории управления разработкой ПО модели процессов становятся важным объектом стандартизации. Особенности процессного подхода к управлению приводят к повышению значимости формализации процессов, выбранной модели процессов. Анализ моделей процессов позволяет выявить разницу в подходах к организации работы ИТ компании.

Для решения задачи эффективного выбора методов, моделей и стандартов целесообразно выполнить анализ существующих инструментальных средств и выявить сходные и отличительные черты. Рассмотрение сходств и отличий

предполагается выполнить в следующих направлениях: история создания, назначение структура, содержание составных частей.

Первым шагом в решении задачи выбора инструментальных средств должна стать их классификация.

Необходимым условием эффективного решения задачи выбора является осознание возможности и необходимости задачи выбора. На практике, ИТ-компании и проектные команды далеко не всегда представляют себе все существующие возможности выбора инструментальных средств, рассматривают их ограниченный набор, подходят к решению проблемы выбора без учета специфики компании и проекта.

Менеджмент и участники проектных команд должны сформировать единое понимание в отношении целей и методов управления ИТ-проектами. Такое понимание может сформироваться в результате непрерывного совместного обучения. М. Фаулер предлагает предоставить возможность разработчикам выбирать проектную методологию самостоятельно [190]. Решение проблемы выбора требует двух типов компетенций:

- во-первых, компетенций в области методологий,
- во-вторых, компетенций в области подходов к выбору.

Для организации эффективного диалога в ходе выбора методологий, участники должны использовать единый подход и согласованные критерии. Таким образом, задача эффективного выбора распадается на задачу создания множества выборов и создание методологии выбора.

Классификация задает структуру для дальнейшего анализа инструментальные средства, с целью выработки рекомендации по их выбору.

1.3. Классификация инструментальных средств управления разработкой ПО

Для создания классификации методов, моделей и стандартов определены различные признаки группировки и выделены группы по этим признакам.

<u>По характеру обоснования рекомендаций</u>: **концептуальные и эмпирические.** Иными словами, используя философскую терминологию, концептуальные модели получены рационально-логическим методом, а

эмпирические – чувственно-опытным. В основе концептуальных инструментальных средств лежат концепции теории менеджмента, такие как процессное управление и ре-инжиниринг бизнес-процессов, управление проектами, управление качеством. Примером концептуальной модели является модель зрелости технологических процессов SEI (Capability Maturity Model, CMM). Концептуальной является методология PRINCE и рациональный унифицированный процесс (Rational Unified Process, RUP).

Эмпирические методологии разработаны на основе теоретического обобщения успешных практик ИТ-проектов. Примерами эмпирических моделей служат SCRUM, XP, Crystal.

В зависимости от назначения: **модели зрелости и процессные модели, проектные методологии** и **индивидуальные и групповые практики**. Модель зрелости CMMI, модель оценки процессов SPICE и стандарт ISO 9000 используются для управления ИТ-компанией (подразделением). Проектные методологии: MSF, SCRUM, XP, используются для управления ИТ-проектами разработки ПО. Методологии внедрения информационных систем используются для организации проекта внедрения. Групповые и индивидуальные практики (PSP, Personal Software process; TSP, Team Software process) служат для непрерывного повышения эффективности работы команд и разработчиков.

В зависимости от условий реализации проекта: **прогнозируемые** и **адаптивные**.

Прогнозируемые методологии основываются на предпосылке о возможности и целесообразности детального планирования будущего. Для ИТ-проекта формулируются требования к разрабатываемой системе, формируется план проекта и определяется потребность в ресурсах. Изменения в плане проекта и требованиях считаются нежелательными. Адаптивные методологии нацелены на преодоление предполагаемой неполноты требований а также их постоянного изменения. Примером адаптивных методологий являются Scrum и Extreme Programming. Адаптивные методологии учитывают психологические особенности процесса разработки ПО. Одним из значимых факторов успеха

использования адаптивных методологий является высокая квалификация специалистов, в первую очередь – разработчиков.

По характеру знаний и фокусу: **инженерные, управленческие и интегрированные.**

Инженерные инструменты (software engineering) основываются на технологических принципах и направлены на совершенствование конечных продуктов, таких как программный код, тестовые примеры, прототипы, документация. Инженерные инструментальные необходимы квалифицированному разработчику.

Управленческие инструментальные средства (software project management and process management) основываются на принципах теории управления (менеджмента), в их основе лежат такие концепции, как тотальное управление качеством, управление проектами, управление знаниями.

Интегрированные инструментальные средства объединяют инженерные концепции и концепции управления.

На Рис. 1. представлена предлагаемая нами классификация инструментальных средств управления разработкой ПО. В центр классификации помещены две группы методологий: *методологии управления разработкой ПО и методологии внедрения информационных систем* (ИС). Методологии содержат рекомендации по использованию отдельных инструментов: метрик, технических стандартов, языков графического моделирования. Методологии включают в себя описание рекомендуемой модели жизненного цикла разработки (внедрения) ПО, описание команды проекта и ролей, а также используемых методик, техник. В зависимости от лежащей в основе методологии управления проектом разработки ПО модели жизненного цикла, проектные методологии варьируют от классической каскадной до итеративных методологий.

Методологий внедрения информационных систем представляют собой набор методологий, разработанных специально для внедрения той или иной информационной системы. В некоторых случая для одной системы может существовать несколько альтернативных методологий внедрения. Методологии

управления ИТ-проектами формируются на основе теории управления проектами. Модели управления ИТ-компаниями опираются на концепции тотального управления качеством и процессного управления.

В левой части классификации представлены универсальные *концепции менеджмента*: тотальное управление качеством (TQM), процессное управление и реинжиниринг бизнес-процессов (BPR) и управление проектами (PM), а также различные вычислительные техники. Самостоятельную группу образуют управленческие стандарты, среди которых наиболее известен стандарт ISO 9000.

Универсальные концепции менеджмента аккумулируют опыт и лучшие управленческие практики, которые стали основой методологий совершенствования деятельности компаний-разработчиков, таких как модели зрелости (CMM/CMMI), стандарты оценки и улучшения процессов (SPICE), и TickIT. Эти модели и стандарты регламентируют организационно-управленческую и технологическую среду, в условиях которой применяются проектные методологии.

Самостоятельный кластер представляют собой *индивидуальные техники*, среди которых выделяется собственный процесс разработки (PSP). В основе индивидуальных техник также лежат концепции управления.

Вычислительные *техники* представляют собой алгоритмы оценки, анализа и оптимизации, используемые в управлении компанией и отдельными проектами. Использование количественных подходов к управлению предполагает использование численных оценок характеристик проектов и процессов.

В правой части представлены *метрики* и *языки моделирования*. Метрики служат для получения фактических и плановых количественных оценок для процессов, проектов и продуктов. Использование метрик является косвенным признаком применения концепций управления качеством в разработке ПО и зрелости организации процессов разработки.

Языки графического моделирования используются для создания понятных и согласованных требований и проектных решений. Универсальный язык

моделирования (UML) позволяет однозначно транслировать графические нотации в проектный код, а также порождать графические описания на основе программного кода. Развитие графических нотаций, как составной части автоматизированного проектирования, оказывает влияние на проектные методологии.

В управлении проектами разработки ПО существует набор стандартных (типовых) задач, которые могут решаться одинаково, вне зависимости от технических и организационных особенностей проекта и выбранной для него методологии. Для решения этих задач предназначены *методики*. Методика служат для решения одной задачи и включают в себя описание области применения, алгоритмы использования, описание необходимых исходных данных. Методики могут предусматривать использование определенных метрик, языков моделирования и стандартов.

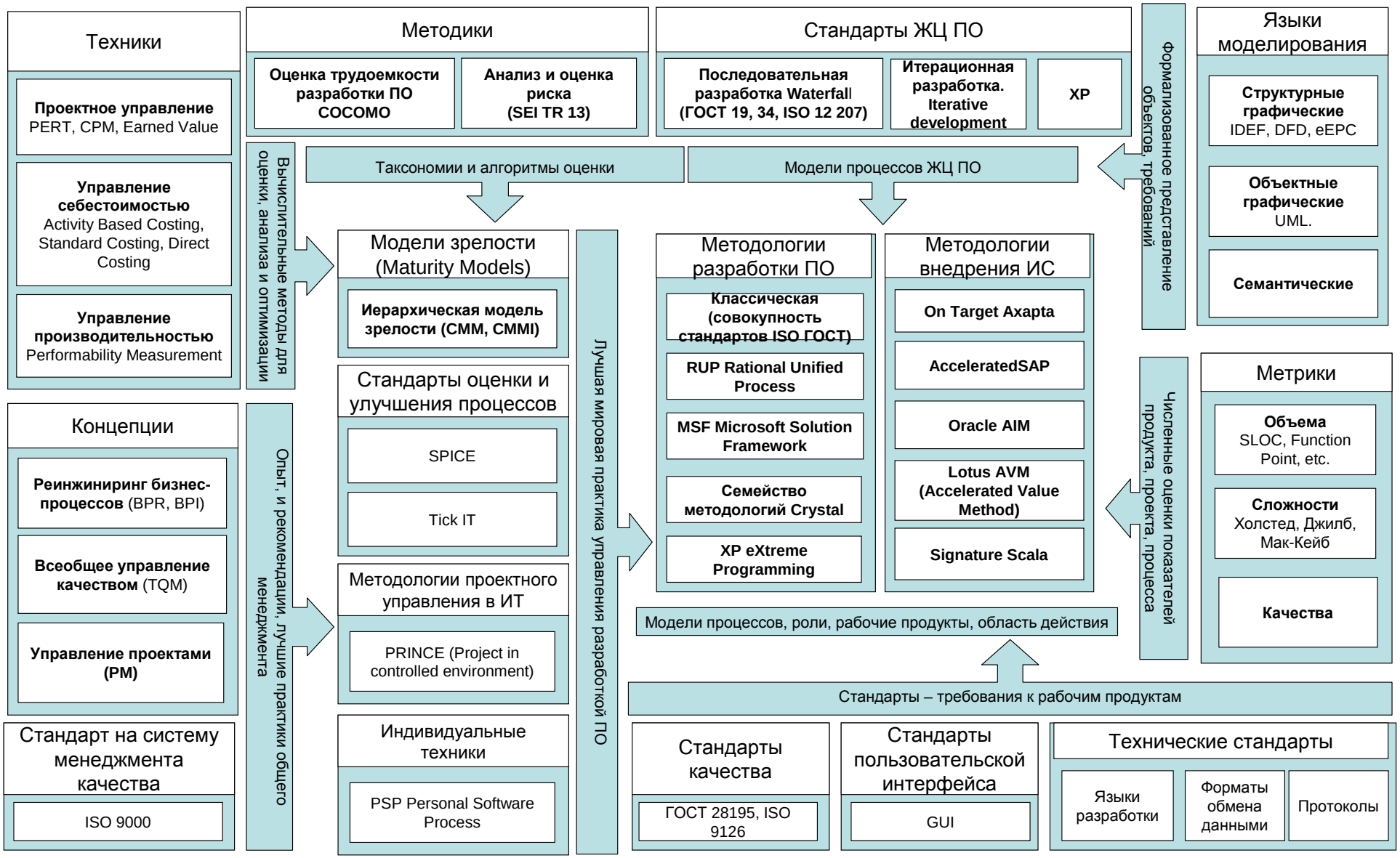


Рис. 1. Классификация методов, моделей и стандартов разработки программного обеспечения

Методики содержат полные рекомендации по решению отдельных проектных задач, таких как управление рисками, или оценка трудоемкости проекта. Примерами методик служат SEI Risk Evaluation Method [278] или COCOMO [209].

В нижней части классификации представлены группы стандартов, регламентирующих различные аспекты разработки ПО. Стандарты разрабатываются международными и государственными организациями по стандартизации, отраслевыми комитетами, исследовательскими институтами, крупными компаниями, такие как ISO (International Organization for Standardization), SEI (Software Engineering Institute), DoD (Department of Defense USA), IEEE (Институт Электронной и Электротехнической Инженерии, Institute of Electrical and Electronics Engineers), МЭК (Международная Электротехническая комиссия), а так ИТ-компании: Bell, Hewlett Packard, Sun Microsystems, IBA, Oracle, Microsoft и др. Стандарты, регламентирующие требования к процессам разработки и выходным продуктам, дополняют методологии управления разработкой ПО. Объектами стандартизации в сфере ИТ являются:

- Конструкторская документация (состав, структура, требования к оформлению);
- Стандарты кодирования и оформления программных текстов;
- Терминология и определения;
- Модели процессов;
- Модели жизненного цикла;
- Требования к безопасности хранения и передачи информации и способы ее обеспечения;
- Качество программного обеспечения, характеристики качества, методы получения данных по качеству;
- Графические и нотации и инструменты формализованного описания требований и технических решений.

Форматы хранения данных, обмена и передачи данных. Модели и стандарты, регламентирующие процессы и жизненный цикл, лежат в основе системы менеджмента качества компании, используются при разработке проектных планов. Возникает потребность в сравнительном анализе моделей и

стандартов, имеющих сходное назначение. Необходимо выявить сходства и отличия в подходах, структуре, области применения.

В результате анализ методологий управления разработкой ПО были выявлены две группы методологий, которые отличаются целью их использования, историей создания и назначением.

Первая группа представляет собой методологии, целью которых является успешное выполнение отдельного проекта. К этой группе относится большинство проектных методологий, и практически все адаптивные методологии. Логика успешного функционирования организации выглядит как рост компетенций, создание технических активов за счет последовательного выполнения успешных проектов.

Вторая группа включает в себя методологии, обеспечивающие устойчивое функционирование компании-разработчика ПО, нацелены на последовательное развитие компетенций. К этой группе относятся модели зрелости (CMM, CMMI). Логика успешного функционирования предполагает создание, контроль и непрерывное улучшение способностей организации к выполнению проектов, и, как следствие, успешное выполнение проектов.

Две выделенные группы отличаются не только целью использования, но и историей создания и развития, а также практикой использования.

Проектные методологии представляют собой ядро теории управления разработкой ПО. К существующей классификации в зависимости от используемой в ней модели жизненного цикла (водопадные (каскадные) и итеративные методологии) добавилась более общая классификация на прогнозируемы и адаптивные методологии. К адаптивным относятся все методологии, которые удовлетворяют требованиям, сформулированным в Манифесте адаптивной разработки.

Прогнозируемые (предикативные) методологии фокусируются на детальном планировании будущего. Известны запланированные задачи и ресурсы на весь срок проекта. Команда с трудом реагирует на возможные измненения. План оптимизирован исходя из состава работ и существующих требований. Изменение требований может привести к существенному изменению плана, а также дизайна

проекта. Часто создается специальный комитет по «управлению изменениями» (change control board) чтобы в проекте учитывались только самые важные требования (ГОСТ 19, 32).

Адаптивные методологии нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. Когда меняются требования, команда разработчиков тоже меняется. Команда, участвующая в адаптивной разработке, с трудом может предсказать будущее проекта. Существует точный план лишь на ближайшее время. Более удаленные во времени планы существуют лишь как декларации о целях проекта, ожидаемых затратах и результатах. Среди адаптивных методологий: (Scrum, Crystal, Extreme Programming, Adaptive Software Development, DSDM, Feature Driven Development, Lean software development)

Проектные методологии в зависимости от решаемой задачи делятся на методологии разработки ПО и методологии внедрения информационных систем.

1.4. Концепции теории управления в методах, моделях и стандартах

1.4.1. Концепция управления проектами

Управление проектами возникло как направление в менеджменте, призванное удовлетворить потребности организаций в достижении специфических целей в условиях высокой сложности и риска.

В основе концепции управления проектами лежит понятие «проект». Определения понятия «проект» в различных стандартах дано в приложении 4.1. Управление проектом представляет собой определение, установление, регулирование и развитие связей между элементами проекта, обеспечивающих достижение поставленных перед проектом целей [203 стр.43]. Описание лучших практик в области управления проектами представлены в PMBok – руководстве к своду знаний по управлению проектами [166].

Разработка программного обеспечения является индустрией, в которой доминирует проектная модель организации производства. Идеи концепции управления проектами лежат в основе проектных методологий в ИТ. В. Липаев применительно к ИТ-индустрии, определяет проект как комплекс формально

организованных мероприятий по достижению единой цели создания сложной системы с заданными характеристиками качества при ограниченных ресурсах [135].

Специфика ИТ-проектов определяет применимость концепции управления проектами, ее отдельных компонентов. Приведем основные специфические проблемы ИТ-проекта и используемые для их решения инструментальные средства.

Табл. 1. Специфика ИТ проектов и инструментальные средства

ОСНОВНЫЕ ПРОБЛЕМЫ УПРАВЛЕНИЯ ИТ ПРОЕКТОМ	ПОДХОДЫ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА, ИСПОЛЬЗУЕМЫЕ ДЛЯ ИХ РЕШЕНИЯ
Высокая неопределенность в отношении сроков выполнения работ и необходимых ресурсов	Сравнительные/экспертные оценки Итеративный процесс разработки Прототипирование
Сложность оценки объема и качества выполненных работ до полного завершения проекта	Использование подходов ISO 9000/CMM/CMMI – зрелый процесс как гарантия качества разработки
Сложность формализации требований к результатам проекта	ГОСТ 19, 34 – разработка ТЗ, ТП, РП Использование CASE средств / UML/RUP Использование референтных моделей, шаблонов Прототипирование / Экстремальное программировании XP
Технологическая гибкость процесса разработки – наличие нескольких принципиально разных планов разработки.	Различные типовые модели ЖЦ ПО, каскадная, водоворотная, RUP, SCRUM, XP

В Табл. 1 представлены основные используемые на практике методы, модели и стандарты, которые используются для решения соответствующих типичных проблем управления ИТ-проектами.

Мы выделяем две группы проектных методологических рекомендаций, в зависимости от подхода:

- основанные на подходах TQM, процессном управлении и непрерывных улучшениях и
- основанные на взаимодействии, итерациях и командной работе.

Табл. 2. Концепция управление проектами в специализированных моделях и стандартах

ЭЛЕМЕНТ КОНЦЕПЦИИ УПРАВЛЕНИЯ ПРОЕКТАМИ	МОДЕЛИ И СТАНДАРТЫ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПО
Определение границ и структуры проекта (scope, взаимосвязь между задачами, ресурсами и необходимым временем - проектный треугольник)	Документ инициации проекта (PID) в методологии PRINCE Стандарт на ТЗ, ТП (ГОСТ 19, 34) Прототипирование (RAD) UML Use Case CMMI SP 1.1-1 Estimate the Scope of the Project
Управление ресурсами проекта, формирование и контроль бюджета проекта (Earned Value)	CMMI SP 2.1-1 Establish the Budget and Schedule. Определение бюджета и расписания.
Определение роли менеджера проекта	Проектные методологии (PRINCE) Процессы управления проектами CMMI
Методы и техники формирования плана работ (фазы/этапы проекта – WBS)	Модели ЖЦ ПО Модели процессов проектных методологий CMMI SP 1.3-1 Define Project Life Cycle. Определение жизненного цикла проекта.
Оценка трудоемкости выполнения проекта	Метрики объема, сложности Метрика функциональных точек (Function Point), Конструктивная модель оценки стоимости (COCOMO) Экспертные оценки CMMI Оценка рабочих продуктов и атрибутов задач (Establish Estimates of Work Product and Task Attributes) Determine Estimates of Effort and Cost
Методы идентификации и минимизации рисков	Таксономия рисков SEI Управление рисками и классификация рисков в MSF
Методы повышения эффективности командной работы	Методологии групповой разработки (XP, SCRUM) Обучение
Техники контроля за выполнением проекта и оценки затрат (Earned Value)	Метрики FP, сложности, LOC CMMI. Количественное управление проектом

В модели зрелости технологических процессов CMMI “проект” – это управляемый набор взаимосвязанных ресурсов, который обеспечивает выпуск одного или нескольких продуктов для клиента или конечного пользователя [266]. К области управления проектами относятся следующие области процессов (Process Areas):

Табл. 3. CMMI. Области процессов управления проектами

Базовые процессы	Дополнительные процессы
Планирование проекта	Интегрированное управление поставщиками
Мониторинг и контроль проекта	Интегрированное управление проектами
Управление договорами с поставщиками	Интегрированное управление командой
	Управление рисками
	Количественное управление проектами

1.4.2. Концепция управления качеством в управлении разработкой ПО

Рассмотрим специфику использования концепции управления качеством и подходов TQM в управлении разработкой ПО.

Тотальное (глобальное) управление качеством (TQM, Total Quality Management) сформировалось как самостоятельная дисциплина в 50-60 г. Впервые успешно примененная на практике в Японии, концепция TQM получила широкое распространение в США и во всем остальном мире.

Тотальное управление качеством – это система непрерывного улучшения, основанная на использовании вовлеченного управления и сфокусированная на потребностях клиентов. TQM объединяет совокупность принципов, статистических методов и методик повышения качества. Среди ключевых элементов TQM вовлечение сотрудников, обучение, командная работа, статистические методы, долгосрочные цели. В основе TQM лежит понимание, что система, а не люди, является причиной неэффективности.

Концепция TQM нашла свое отражение в системе стандартов качества серии ISO 9000. Составной частью стандарта ISO 9000 является словарь, качества, определяющий основные понятия. Определения словаря ISO даны в приложении:

Задача управления качеством распадается на задачу управления качеством соответствующих объектов: качества продукции, качества процессов, качества организации и качества ее составных частей, включая качество персонала. Ключевой задачей обеспечения качества продукции является выявление потребностей потребителей. В проектах разработки ПО задачи требования формулируются в спецификациях, моделях, прототипах, сценариях использования. Определение требований осуществляется в ходе предпроектного обследования, анализа, разработки прототипа. Функциональность и технические характеристики ПО сравниваются с требуемыми (ожидаемыми) характеристиками на этапе тестирования.

Особенности управления качеством ПО обусловлены следующими факторами: *отсутствием повторяемости процессов и жестко заданной технологии, индивидуальным характером производства (проектная модель)*.

Среди определений качества ПО:

- отсутствие ошибок (дефектов);
- соответствие спецификации [217];
- пригодность к использованию (fitness for use) [247];
- удобство использования (комфортабельность).

Основными параметрами качества ПО считаются: функциональная полнота, безопасность информации, простота эксплуатации, не требующая специальных знаний в области информационных технологий, эргономичность пользовательского интерфейса, минимизация затрат на эксплуатацию, развитие и модернизацию [39].

ГОСТ 28195-89 [49] устанавливает общие положения по оценке качества программ, определяет состав параметров, которые должны контролироваться на различных этапах ЖЦ ПО, а так же устанавливает иерархические связи между ними. Оценка качества определяется как выбор номенклатуры показателей качества оцениваемого ПО, определение значений этих показателей и их сравнение с базовыми значениями.

Качество ПО определяется не только реализацией функциональных и технических требований, но возможностью модификации ПО, защитой от неправильного использования, легкости интеграции с другими приложениями.

Международный стандарт ISO/IEC 9126 регламентирует *модель качества ПО*, которая позволяет формулировать требования к качеству на этапе проектирования, отслеживать характеристики качества в процессе разработки и эксплуатации.

В Табл. 4. представлены результаты сравнения показателей качества, выделяемых Б. Боэмом [24], с составом показателей стандарта ISO 9126. Основные отличия составляет появление характеристики «функциональность» и исключение характеристики «оцениваемость».

Табл. 4. Сравнение характеристик качества ISO/IEC 9126 и Б. Боэм

Характеристики качества ISO/IEC 9126	Характеристики качества Б. Боэм [24, стр. 22]
Функциональность	Надежность
Надежность	Учет человеческого фактора Понятность
Удобство эксплуатации	Эффективность
Эффективность	Модифицируемость
Модифицируемость	Мобильность
Мобильность	Оцениваемость

Понятие качества ПО тесно связано с определением приоритетов требований. Эдвард Йордон утверждает, что одним из аспектов, порождающих трудности в разработке ПО, становится требование *идеального качества*, выражаемого в терминах количества ошибок, переносимости, независимости от платформы, и других. Идеальное качество недостижимо в большинстве проектов, поэтому необходимо прийти к соглашению относительно того, какое качество является *достаточно хорошим*, то есть определить требования к «достаточно хорошему» ПО *(good enough software*) [84, стр. 185].

Общие подходы методологии TQM (Total Quality Management) могут быть использованы при разработке ПО, но применение традиционных инструментов, к которым относятся гистограммы, временные ряды, диаграммы Парето, причинно-следственные диаграммы, контрольные листки, контрольные карты, диаграммы

рассеяния, требует использования количественных оценок, которые предполагают сбор метрик, что предполагает высокий уровень зрелости процессов разработки. Идеи TQM содержатся в методологиях управления ИТ-компанией. Идея непрерывного улучшения на основе процессного подхода и измеримых результатов является центральной частью CMMI. Признанным положением TQM стало понимание необходимости культурных изменений в организации: принятии цели постоянного улучшения, уничтожения границ между функциональными отделами, организации постоянного обучения для всех сотрудников.

Концепция TQM, стандарты ISO 9000 и методологии управления проектами и совершенствования деятельности компании (CMM/CMMI, SPICE и пр.) основываются на предпосылке о наличие прямой связи между организацией процесса разработки и качеством ПО: система, процессы, организация, а не люди, определяют уровень качества.

В вопросе оценки зависимости качества продукта от организации процесса существует альтернативная точка зрения. Например, Джеффри Воас (Jeffrey Voas) в статье «Качество ПО: восемь мифов», анализируя связь между качеством ПО, процессами разработки, использованием формальных методов, метрик, CASE-средств, утверждает, что предположение об однозначной связи межу официальными рейтингами процессов и качеством производимого ПО ошибочно, а использование TQM, которое действительно приносит успех в традиционных областях индустрии, не всегда оправдано в промышленном программирование, которое остается творческим процессом со значительным элементом неопределенности [37].

На практике использование идей тотального управления качеством в разработке ПО представляет собой компромисс между творческим характером деятельности и потребностью в создании устойчивых и повторяющихся процессов разработки. На решение оказывает влияние в первую очередь бизнес-модель (заказная разработки или продуктовая модель), а также объем проектов, характер рисков разработки и эксплуатации ПО.

1.4.3. Концепция управления бизнес-процессами и реинжиниринга бизнес-процессов

Концепция процессного управления является ключевым элементом в методологиях оперативного и стратегического менеджмента, на нее опирается концепция управления качеством TQM. В методологии стратегических карт (Strategy Map) Роберт Каплан и Дэвид Нортон устанавливают связь между целями собственников, которые достигаются путем предоставления ценности для клиентов на основе бизнес-процессов организации, протекающих в заданной культурной, технологической и организационной среде [91]. Таким образом, бизнес-процессы оказываются в центре методологии Strategy Map.

Процесс (от лат. processus - продвижение), 1) последовательная смена состояний стадий развития. 2) Совокупность последовательных действий для достижения какого-либо результата (например, производственный П. - последовательная смена трудовых операций) [23].

Определения процессов содержатся в приложении 4.3.1. В области разработки ПО существует несколько специализированных подходов к выделению процессов. Например, Уокер Ройс выделяет три уровня процесса [162, стр. 41]:

Метапроцесс - политика, приемы и практика, которые присущи некоторой организации при ведении интенсивного бизнеса, связанного с ПО. В центре этого процесса находится экономика организации, долговременная стратегия и возврат инвестиций в ПО.

Макропроцесс - политика, приемы и практика, которые присущи некоторому проекту по созданию законченного ПО с учетом определенных ограничений по стоимости, срокам и качеству. В центре этого процесса находится создание адекватного варианта метапроцесса для конкретного набора ограничений.

Микропроцесс - политика, приемы и практика, присущие команде разработчиков некоторого проекта и направленные на получение результатов в процессе создания ПО. Главным для микропроцесса является создание промежуточного продукта адекватного качества с адекватными функциональными

возможностями настолько экономично и быстро, насколько это осуществимо на практике.

В классической работе Фредерика Брукса Мифический человеко-месяц выделяются три концептуальных вида деятельности в проектах разработки ПО [25, стр. 191]:

- формулирование формальных конструкций (дизайн, проектирование);
- воплощения в реальном материале (программирование, документирование);
- диалог с пользователями в реальной жизни (внедрение, поддержка, обучение).

В результате концептуального проектирования создается сущность программы (essence), а результатом воплощения является второстепенная составляющая (accident). Такое разделение на создание сущности и воплощение обусловлено технологией разработки того времени (50-70 годы), отсутствием интерактивных средств разработки, позволяющих объединить проектирование и программирование, а также существенным удельным весом трудозатрат на техническую работу (до 90% времени). Брукс предлагал сфокусироваться на повышении эффективности процессов создания «сущности», уделять больше внимания концептуальному замыслу.

На наш взгляд, в настоящее время все сложнее становится разделять собственно творческую и техническую работу по созданию программного обеспечения, процессы проектирования и программирования часто пересекаются.

Для повышения производительности труда разработчиков Ф. Брукс считал необходимым выделить затраты, необходимые для изготовления концептуальных конструкций и для точного и упорядоченного их представления. Дальнейший анализ методологий управления разработкой ПО показал, что идеи Ф. Брукса до сих пор не нашли в них адекватного отражения.

Определение процесса создания автоматизированной системы приводится в ГОСТ 34.601. *Процесс создания автоматизированной системы* представляет собой совокупность упорядоченных во времени, взаимосвязанных, объединенных в стадии и этапы работы, выполнение которых необходимо и достаточно для

создания АС (автоматизированной системы), соответствующим заданным требованиям.

Майкл Хаммер и Джеймс Чампи предложили концепцию ре-инжиниринга бизнес-процессов (Business Process Reengineering, BPR), благодаря которой процессное управление было интегрировано в систему управления корпораций [197]. BPR стал способом радикального повышения эффективности бизнеса, снижения издержек и повышения качества услуг. Суть BPR в реинтеграции процессов – корректировка крайности специализации и разделения труда, преодоление отрицательных эффектов подходов Смита-Маркса-Гилбрета.

Питер Друкер писал *«Новая технология, в отличие от раннего индивидуального производства, не фокусирует внимание на мастерстве исполнителя, как объединяющем принципе работе. Кроме того, в отличие от выдвинутой Фордом концепции массового производства, она не считает продукт организующим принципом. Она фокусирует внимание на самом процессе, который рассматривается как единое и гармоничное целое. Цель – добиться оптимизации процесса – процесса, который будет производить самую разнообразную продукцию с наивысшей степенью стабильности, при наименьших затратах и с наименьшими усилиями»* [65].

Развитие концепции процессного управления обусловило развитие техник графического и аналитического моделирования бизнес-процессов, появление сопутствующего программного обеспечения.

Современная теория управления процессами выделяет три уровня усилий по улучшению бизнес-процессов: непрерывное улучшение процессов, ре-дизайн процессов и реинжиниринг процессов (см. приложение 4.3.2.)

В управлении разработкой ПО концепция управления бизнес-процессами находит свое отражение в моделях зрелости (CMM/CMMI) и стандартах улучшения процессов (SPICE). Управление процессами приводит к улучшению функционирования организации, повышению эффективности отдельных процессов и организации в целом. Управление проектом направлено на выполнение целей проекта с использованием доступных ресурсов и в заданные сроки.

Процессы существуют как шаблоны последовательности действий (классы). В ходе выполнения проекта осуществляется реализация процесса - появляется экземпляр процесса – объект (process instance).

Управление процессами включает в себя управление шаблонами процессов (создание моделей процессов) и управление практической реализацией (оперативное управление). Для создания моделей процессов используются формализованные нотации, среди которых выделяется класс графических нотаций.

Под формальным описанием понимается любое структурированное описание процесса, как с использованием средств графического моделирования (IDEF0, EPC, Activity Diagram), так и вербального описания.

Бизнес-процессы пересекают границы функциональных подразделений, выполняются на разных уровнях в организации. Наблюдения показывают, что чем выше уровень управления, тем сложнее идентифицировать процесс, которому принадлежит выполняемая функция. Причиной тому служит совмещение нескольких целей при выполнении одного действия. Все большую часть времени занимает неформализованное общение с сотрудниками и клиентами, обмен информацией относительно условий контрактов, рисков, согласования сроков. Отдельную проблему представляет собой построение процесса, пересекающего иерархические уровни в организации (выходы функций вышестоящего уровня являются входами или механизмом управления функции нижестоящего уровня). Эта проблема осложняется наличием следующих факторов: потеря информации (утаивание, искажение информации, особенно информации об отклонениях и рисках), меньшая структурированность деятельности на более высоких уровнях управления.

Мы выделяем процессы двух типов: *предопределенные* и *стихийные*. *Предопределенные процессы* – это процессы, которые были спроектированы менеджментом для выполнения бизнес-задач компании. *Стихийные процессы* – это процессы, которые возникают в результате адаптации сотрудников и организационных единиц к изменившимся условиям, предназначены для выполнения новых требований, решения новых задач. Стихийные процессы возникают как изменения (модификации) в уже существующих процессах более

высокого уровня. В отдельных случаях менеджмент может не знать о существовании нового стихийного процесса.

Модель процессов представляет собой иерархически упорядоченный список процессов.

Для практического использования процессного подхода и понимания взаимосвязи между концепциями управления и процессами в организации имеет смысл выделить следующие три группы процессов:

- Процессы управления проектами — планирование, организация и контроль проекта;
- Процессы разработки продукта — проектирование, разработка, тестирование и внедрение ПО, разработка документации;
- Организационные процессы – направленные на улучшение работы всей компании, процессы обучения, процессы изменений.

Процессы управления проектами являются типовыми, их регламентирует методология управления проектами. Процессы разработки продукта связаны с выбранной моделью ЖЦ ПО, которая определяется выбором методологии управления процессом разработки ПО. Некоторые проектные методологии содержат детальное описание процессов.

Можно ли выделить процессы управления ПО из совокупности процессов, или управление представлено как функция (функции) в процессах анализа, проектирования и разработки? Любая деятельность, связанная с планированием, организацией, учетом, контролем (анализом) относится к управлению по определению. Но, в разработке ПО планирование тесно связано с проектированием, и определение сроков выполнения с использованием данных метрик, и, следовательно, принадлежит к процессам разработки. Вернее, задача *планирования* включает в себя несколько отдельных подзадач, между которыми существуют сложные взаимосвязи: определение состава работ, определение сроков сдачи отдельных этапов и поставляемых на каждом этапе продуктов, распределение ресурсов, оценка продолжительности отдельны работ. Подзадача может иметь свой критерий оптимальности, например, при определении сроков

сдачи этапов может оптимизироваться денежный поток проекта, при распределении ресурсов оптимизируется загрузка ресурсов.

Для выполнения подзадачи требуется специальная квалификация, то есть происходит дробление функции планирования, что делает актуальной задачу координации этой функции. Исходными данными для планирования служат: модели жизненного цикла (ЖЦ), модели процессов, типовые архитектуры, шаблоны проектирования, данные метрик.

Учет представляет собой сбор метрик. Функция *контроля* в разработке ПО распадается на функцию *тестирования* и функцию *отслеживания выполнения*. Но проектирование, сбор метрик и тестирование не являются функциями управления. Таким образом, мы видим, что традиционное определение функции управления, которое использовалось в индустриальном производстве, требует уточнения для компаний, занимающихся разработкой ПО.

Все процессы в компании делаться на две группы: связанные и не связанные непосредственно с проектами разработки ПО. При составлении плана проекта учитываются только те функции (работы), которые непосредственно связаны с проектированием и разработкой. Связь с остальными функциями осуществляется через доступность ресурсов.

Отсутствие формализованного описания процесса, отсутствие повторяемости процессов представляются значимой проблемой, с которой сталкиваются организации-разработчики ПО.

Проблема формализации бизнес-процессов актуальна для всех компаний и организаций. Внедрение информационной системы и (или) сертификация системы менеджмента качества требуют формализации бизнес-процессов. Формальное описание процессов необходимо для обеспечения повторяемости процесса, однозначного понимания процессов всеми сотрудниками, возможности измерения характеристик процессов.

В основе организационных процессов лежат идеи TQM. Модели процессов уровня компании содержатся в SPICE и CMMI. Модели процессов уровня проекта содержатся в проектных методологиях RUP, MSF, Prince.

Проектные методологии и методологии улучшения деятельности ИТ-компаний предлагают модели процессов. Модели процессов ISO 12207, CMMI, SPICE, RUP, MSF, Oracle CDM, Prince, XP, PSP представлены в приложении 4.5.

1.4.4. Управление рискам в разработке ПО

Риски присущи проектам по разработке ПО. Отклонение проектов от запланированных бюджетов и сроков, а также невыполнение требований является фактической реализацией проектных рисков. Потребность в управлении рисками оказывает существенное влияние на методологию управления разработкой ПО. Примером специализированной методики управления рисками служит Метод оценки рисков SEI (Software Risk Evaluation Method) включает в себя классификацию (таксономию) рисков, и описание процессов, позволяющих оценить риски и их минимизировать.

Управление рисками является составной частью управления проектами. Как составная часть методологии управления проектами, управление рисками имеет богатую методологическую поддержку [278, 203]. Вместе с тем, риски в управлении проектами разработки ПО имеют особенную природу, что обуславливает потребность адаптации типовых подходов к управлению рисками в проектах.

Риск определяется как *мера или ситуация возможных потерь*. Математически риск может быть представлен как математическое ожидание потерь в денежном или ином выражении. Более детальное представление риска предполагает задание распределения вероятности ожидаемой величины (потока доходов, прибыли, или иных финансовых или технических характеристик);

Под неопределенностью понимается неполнота или неточность информации, например, информации об условиях реализации проекта. Неопределенность, связанная с возможностью возникновения в ходе реализации проекта неблагоприятных ситуаций и последствий, характеризуется понятием риска [204, стр.184]. Потерями в проекте могут быть ухудшение качества конечного продукта, увеличение затрат на разработку, невозможность завершения проекта.

Управление рисками включает в себя установление причин рисков (факторов риска), установление взаимосвязей между ними и отдельными видами риска, прогнозирование рисков, оценку рисков, разработку способов предотвращения рисков.

Михаил Корольков, директор по ИТ компании Siemens Business Services, С-Петербург, считает, что основной причиной рисков в разработке программного обеспечения является распространенная недооценка сложности этого вида деятельности[1]. Относительно низкая доля постоянных затрат, возможность самостоятельного изучения программирования, и, как следствие, иллюзия легкости создания продукта приводят к тому, что проекты начинаются без предварительного проектирования, выполняются недостаточно квалифицированными разработчиками (специалистами без специального образования в сфере разработки программного обеспечения).

Наличие специального образования у программистов не является обязательным требованием в большинстве компаний при приеме на работу, основным условием является знание конкретных языков и средств разработки. Например, высшее образование в области ИТ на российском рынке требуется только для 60% вакансий программистов [165].

Разработку программного обеспечения можно сравнить с проектированием дома, а строительство сравнимо с компиляцией. Но стоимость строительства значительно превосходит стоимость проектирования, в то время как стоимость компиляции ничтожно мала. Именно высокая стоимость строительства и исправления ошибок на более поздних этапах приводит к тому, что проектированию в строительстве уделяется большое внимание, и строительство не начинается до того, как не будут готовы детальные планы, причем требования к квалификации проектировщика являются достаточно жесткими. В то же время в разработке ПО затраты на исправление ошибок, внесенных и исправленных на различных этапах не столь очевидны. Не смотря на то, что существуют работы, посвященные изучению этого вопроса (например, 24), на практике многие

[1] По результатам интервью 05.02.2002

компании разработчики ПО, особенно небольшие, пренебрегают этапом технического проектирования - пропускают этот этап, или ограничиваются эскизным проектом.

Таким образом, мы можем выделить две категории рисков в зависимости от характера факторов рисков:

Добровольные риски – риски, которые участники проектов принимают на себя из-за желания получить дополнительную прибыть, сократить сроки разработки;

Неизбежные риски – риски, обусловленные творческим характером деятельности, изменением внешних условий, форс-мажорные риски.

Классификации рисков являются инструментом, применяемым для идентификации рисков. Риски можно классифицировать по разным критериям: по субъекту, который будет нести бремя потерь (компания-разработчик, сотрудники компании-разработчика, заказчик, пользователь), по природе источника рисков (технологические, личные (персональные), «добровольные», бизнес-риски). Риск может варьировать от катастрофического, приводящего к потере данных, временной или постоянной утрате работоспособности системы, до незначительного, состоящего в появлении кратковременных сбоев.

Так же можно классифицировать риск по вероятности возникновения: маловероятный, ожидаемый, высокая вероятность потерь. По характеру проявления последствий рисковой ситуации можно *статический* и *динамический* риски. Статический риск характеризуется одномоментным единовременным убытком, например, в результате порчи оборудования или прекращения проекта. Динамический риск представляет собой риск, который вызывает снижение (ухудшение) значений потоковых величин, например, рост заработной платы или увеличение процентной ставки.

Анализируя причины риска можно выделить *внешние* и *внутренни*е риски. Внешние риски имеют свои причины за границами компании, следовательно, компания-разработчик не может на них воздействовать. К внешним рискам можно отнести риски, связанные с действиями государства, например, введение новых налогов, установление ограничительных мер, введение лицензирования, риски,

обусловленные действиями конкурентов, например, создание альтернативного продукта, или ценовая конкуренция. Клиенты также представляют собой источник риска, который может быть отнесен к внешним рискам. Типичным примером такого риска является отсутствие у заказчика необходимой квалификации для определения собственных требований.

Внутренние риски обусловлены причинами, находящимися внутри организации. Часть рисков обусловлены существующей структурой и организацией работы, например, структура собственности влияет на способность к привлечению капитала, и, следовательно, влияет на финансовые риски.

В реальных проектах можно обнаружить сложную сетевую структуру факторов рисков, когда один фактор риска, действуя в комбинации с другими факторами, является причиной нескольких рисков. При этом один фактор может быть внешним, например, квалификация заказчика, а другой - внутренним, например, собственный опыт компании-разработчика в данной сфере. Только комплексный анализ всех факторов, влияющих на реализацию конкретной рисковой ситуации, позволяет дать качественную и количественную оценку данному риску, разработать меры по его уменьшению (предотвращению).

В том случае, если разработчик обладает большими знаниями в определенной сфере, имеет библиотеку типовых решений, референтных моделей бизнес-процессов, то даже при наличие определенной квалификации у заказчика, риск невыполнения действительных бизнес-задач в результате реализации проекта можно считать маловероятным, поскольку исполнитель компенсирует недостаток компетенции заказчика собственными знаниями. Нужно отметить, что существует определенная мера, до которой различные факторы могут оказывать компенсирующее воздействие. Очевидно, что наличие у заказчика способности к пониманию принципов функционирования ПО, желание использовать преимущества, которые дают информационные технологии для совершенствования собственных процессов, не может быть полностью заменено опытом исполнителя в данной сфере бизнеса.

Существующие модели рисков не отражают причинно-следственные связи между факторами рисков и рисками, а представляют собой одноуровневые либо

иерархические списки возможных рисков. Например, в модели рисков SEI приводится около 200 вопросов относительно различных аспектов разработки ПО, сгруппированных по трем направлениям: продукт, среда разработки, проектные ограничения). Предлагаемые классификаторы позволяют достаточно полно выявлять риски, но не дают целостной картины рисков, с учетом обуславливающих их факторов.

Наличие значительного числа факторов риска и большого числа связей между ними усложняет создание универсальной модели рисков. Теоретически такая модель может быть построена (примером общей модели риска, является модель рисков SEI [278]).

Методология MSF предлагает свой подход к управлению рисками и классификацию рисков по источникам и возможным последствиям (Приложение 4.6.1.). Для каждого риска определяется рейтинг риска, равный произведению его вероятности на влияние. На основе анализа разрабатывается план борьбы с десятью главными рисками, для каждого плана исследуются пять основных параметров: информация – достаточно ли мы знаем о риске, приемлемость – можем ли мы смириться с последствиями риска, исключение – можем ли мы избежать риска, снижение – можно ли уменьшить вероятность риска и / или его последствия и план действий – что можно сделать, если риск реализовался.

Структура классификации рисков MSF двумерна, одним измерением являются факторы рисков, другим - возможные последствия. Для управления выделяются области управления рисками (mitigation areas).

Используя несколько различных классификаций, построенных на основе различных признаков классификации, можно более полно выявить потенциальные риски проекта.

В модели оценки рисков, предлагаемой SEI, представлена таксономия рисков (taxonomy of software risk), включающая в себя описание классов, элементов и атрибутов рисков [278, стр. A-70], а так же список привязанных к таксономии вопросов, позволяющих выявить потенциальные риски.

Таксономия рисков методологии SEI представлена в Приложении Модель выделяет три класса факторов рисков: разработка продукта, среда разработки и

проектные ограничения. Предлагаемый подход сразу позволяет идентифицировать природу факторов риска, кроме того, используя предложенные классы можно распределить задачу идентификации (определения) и анализа рисков между сотрудниками.

На наш взгляд, таксономия рисков SEI является достаточно полной, хотя и недостаточно хорошо структурирована (сбалансирована), например, атрибуты продукта в разделе «A.1 Требования», «B.2 Система разработки» сходны с характеристиками качества, представляют собой определенные атрибуты продукта или процесса, а в то время как для «B.4 Методы управления» атрибутами являются направления деятельности, а для «C.3 Программные интерфейсы» атрибутами являются участники проекта (заказчик, субподрядчики, менеджмент). В результате между атрибутами риска устанавливаются многочисленные отношения, одни атрибуты частично включают другие.

Эффективное управление рисками требует выделения наиболее значимых рисков (рисков с максимальным математическим ожиданием потерь), для фокусирования внимания менеджмента и осуществления активных действий. После идентификации рисков, необходимо их проанализировать, сгруппировать, используя признаки классификации и факторы, обуславливающие риски.

Модель рисков SEI содержит описание функций управления риском, все функции разбиваются на две группы: первичные и поддерживающие. К первичным функциям относятся определение риска, его спецификация, оценка, консолидация и предотвращение. К поддерживающим функциям относятся: планирование и координация, оценка и утверждение, обучение и коммуникация. Для каждой функции приводится описание назначения, возможных способов и механизмов ее реализации.

Большую практическую ценность имеет предложенный SEI подход к уменьшению рисков. Предлагаются конкретные методы управлении рисками, такие как области управлении риском (mitigation area), карта риска (risk map).

Области управления риском представляют собой логические группы рисков, объединенных с целью эффективного управления. Используя эти области можно управлять рисками с учетом целей проекта. Для каждого вида проекта существуют

свои области, типичный набор включает в себя: управление системными требованиями, планирование и отслеживание проекта, координация, конфигурационное управление, контроль и проверка.

Карта риска используется для выработки стратегий управления риском и отдельных действий. С помощью карты рисков отдельные риски сопоставляются с областями управления и целями проекта.

1.4.4.1. Потеря и искажение информации как источник риска

Передача информации между уровнями управления вызывает искажение информации. Это искажение не всегда должно иметь негативную оценку. Можно говорить о целенаправленном преобразовании информации, что бы она в большей степени удовлетворяла действительным потребностям вышележащего уровня управления. Наиболее существенной проблемой является недостаточность информации в ситуации, когда нижележащие уровни не имеют первичных данных для предоставления информации, запрашиваемой менеджментом.

Отдельную проблему предоставляет собой передача информации о рисках. Отношение к информации, которая позволяет оценить риск, зависит от уровня лица, принимающего решения, в организационной структуре. На каждом уровне существует желание получать полную информацию с предыдущих уровней, но не передавать ее на вышележащие уровни. Информация, позволяющая оценить риск, с легкостью распространяется по организационной структуре сверху-вниз, но с большим трудом осуществляет движение снизу-вверх.

На любом этапе проекта по разработке ПО существует неопределенность относительно текущего состояния объекта. То есть имеет место неопределенность не только относительно будущего, но и относительно настоящего, и прошлого. Как правило, отдельные участники проекта имеют представления о возможном риске, но существует фрагментация распределения информации о риске, из-за отсутствия коммуникаций отсутствует целостная картина, описывающая возможные виды риска и связанные с ним последствия.

Следует особо отметить проблему искажения информации в связи с давлением, оказываемым на разработчиков и менеджеров проектов, и отсутствием

возможности обосновать какие-либо временные оценки с помощью количественных методов.

Ф. Брукс писал о разработке программного обеспечения *«… липовые графики, нацеленные на желательную дату, встречаются здесь значительно чаще, чем в любых других инженерных отраслях. Очень тяжело, рискуя потерять рабочее место, с энергией и любознательностью отстаивать срок, который определен без применения каких-либо количественных методов при недостатке данных и подкреплен, в основном, интуицией менеджеров»* [25 стр. 29]. Таким образом, риски объективно обусловлены отсутствием количественных методов оценки трудоемкости работ, времени выполнения, а так же метрических данных.

Искажение информации существует на всех этапах разработки, но на стадии заключения контракта, первоначальной оценки продолжительности проекта и необходимых ресурсов существуют распространенные стратегии переговоров, реализуемые как заказчика, так и исполнителями. В основе этих стратегий лежит умышленное искажение информации, в результате повышаются риски проекта.

Роб Томсет (Rob Thomsett) описывает эти стратегии (переговорные игры), в результате которых менеджер проекта дает нереалистичную оценку проекта, или под давлением вышестоящего руководства или заказчиков (угадай число, которое я задумал, «двойной плевок пустышкой», «испанская инквизиция», игра не понижение), или сознательно дает ложную информацию для достижения собственных целей, завышая оценки («удвой и добавь еще»), занижая их («месть») или предоставляя правдивую информацию порциями («китайская пытка водой»). Для придания большей достоверности ложной информации могут использоваться метрики ПО («туманы и миражи»), а так же демонстрация незавершенного ПО – прототипов, которые невозможно эксплуатировать (спрятанное качество) [287].

Все рассмотренные стратегии (игры, политики) приводят к искажению информации, увеличивают степень неопределенности, увеличивают, таким образом, риски проекта. Их отличительной особенностью является тот факт, что один или несколько участников проекта сознательно увеличивают риски, при этом максимизируется не ожидаемая прибыль проекта или иные характеристики, а достигаются индивидуальны цели участников проекта. Эти риски наиболее

сложны для управления, поскольку их минимизацией должны заниматься те сотрудники, которые сами являются источниками факторов рисков.

1.4.4.2. Методы управления рисками

Управление риском представляет собой систематическое определение, анализ и контроль над рисками, которые угрожают активам организации или ее производительным способностям. Стандарт SPICE выделяет управление рисками как самостоятельный процесс (PRO.6 Manage Risks). Целью этого процесса является систематическое определение и уменьшение рисков в течение жизненного цикла проекта. Управление риском предполагает оценку эффективности усилий по уменьшению рисков. Кроме собственного процесса управления рисками, стандарт SPICE содержит несколько базовых практик, связанных с управление рисками: CUS 6.1 Определение операционных рисков, PRO 2.6 Определение начальных рисков проекта.

Существуют два принципиально разных подхода к управлению рисками: *реактивное* и *проактивное* (превентивное) управление рисками. Превентивное управление рисками предполагает наличие продуманного плана и процесса управления рисками до их реализации, то есть до возникновения проблемы.

Негативной тенденцией в разработке ПО стало перемещение функции управления рисками из сферы управления отдельным проектом в сферу тестирования ПО. Большой проблемой является получения целостной картины рисков, поскольку идентификация рисков в разработке является задачей отдела тестирования, а потребителями этой информации является менеджмент и разработчики.

Существует мнение (которое зачастую разделяют заказчики), что наличие формализованного, структурированного, повторяющегося («зрелого») процесса разработки позволяет уменьшить риски. Но честное отношение разработчика к клиенту не гарантируется даже идеальной организацией процесса. Стратегии, предусматривающее умышленное искажение информации рассмотрены в п. 1.4.4.1. Выраженные в этическом кодексе и (или) политике качества

обязательства позволяют оценить отношение разработчика к проблеме риска, дополнительной гарантией служит успешный опыт в прошлых проектах.

Очевидно, что подходы к управлению риском будут отличаться в зависимости от существующих *приоритетов*. Например, компания Microsoft рекомендует рассматривать в качестве основного приоритета *сроки разработки*. Ориентация на выпуск в срок означает отношение к дате выпуска продукта как к императиву [171 стр. 130]. На этапе планирования рассматриваются все аспекты компромиссного треугольника: время, цена, функциональность. Когда же дата выпуска согласована, этот элемент компромиссного треугольника больше не используется при принятии каких-либо решений корректирующего характера.

Очевидным преимуществом данного подхода является *мобилизирующий* эффект, обеспечивающих мотивацию проектной группы и наличие единственного критерия для принятия решений. Функциональные возможности продукта упорядочиваются по приоритетам, чтобы при необходимости можно было отказаться от реализации характеристик с низким приоритетом. Не смотря на то, что качество формально не является элементом проектного треугольника, и не становиться объектом компромисса в явном виде, но функциональность влияет на качество. Ориентация на выпуск в срок противоречит подходам компаний, ориентированных на классические ценности TQM.

Например, подход Hitachi Software Engineering показывает, что японские компании имеют иные приоритеты. «В том случае, если мы не обеспечим качество, клиент будет вспоминать об этом еще и еще раз. Но если мы задержимся с выпуском, клиент будет недоволен только однажды, и, возможно, скоро забудет об этом. В том случае, если мы недооценим издержки, клиент не будет жаловаться, поскольку никогда не узнает об этом. Мы возьмем на себя превышение стоимости» [293 стр. 210]». Эта цитата не только раскрывает приоритеты компании, но и показывает, каким образом распределятся последствия рисков между участниками проекта.

Управление риском реализуется на различных уровнях: при выборе маркетинговых, инновационных и предпринимательских стратегий, при проектировании бизнес-процессов, в ходе реализации отдельного проекта. Таким

образом, можно выделить *стратегическое* и *оперативное* управление рисками. Выделяя стратегическое и оперативное управление рисками, следует отметить, что связь между ними реализуется с помощью организации процессов, в нашем случае, процессов разработки программного обеспечения. Кроме того, организация процесса является необходимым условием целенаправленного и систематического управления рисками.

Политика компании в области качества является одним из инструментов стратегического управления рисками, она содержит в себе декларацию обязательств компании по отношению к клиентам. Таким образом, гарантируются не только характеристики товаров и услуг, но и распределяются риски между разработчиком, заказчиком и пользователем. На основе политики качества с учетом реальных рыночных условий заключаются контракты на разработку ПО и обслуживание, в этих контрактах детализируются отдельные виды риска, периоды времени, когда могут проявиться соответствующие риски, а так же определяется сторона, которая будет нести бремя потерь в случае реализации рисковой ситуации, и, в идеальном случае, планируются мероприятия по компенсации результатов рисков.

Политику качества дополняет «этический кодекс» разработчика, который содержит обязательства разработчика. Важность политики качества и этического кодекса в управлении рисками обусловлена тем обстоятельством, что в некоторых сферах деятельности (в том числе и в разработке ПО), существует неравномерное распределение результатов и рисков. Доходы (их большая часть), полученные в случае реализации удачного исхода, достаются разработчику, в то время как убытки (потери) в случае неудачи лежаться на плечи заказчика (или пользователя). Одновременно с тем, исполнитель может выбирать более рискованные варианты и способы, не информируя о том заказчика. В этом случае заказчик может столкнуться с результатами рисковой ситуации неожиданно, и более того, результаты могут быть такими, что заказчик никогда бы не согласился взять на себя подобные риски.

Безусловно, заказчик имеет определенную заинтересованность в благополучие исполнителя. Лучшее финансовое положение исполнителя позволит

ему повысить качество своей продукции, обеспечить качественное сервисное обслуживание. Поэтому заказчик заинтересован в наличие у исполнителя эффективного процесса управления рисками, который бы давал полную, достоверную и своевременную картину риска (включая качественный и количественный анализ), обеспечивал двустороннюю коммуникацию как внутри компании-разработчика, так и с заказчиками и подрядчиками.

Управление рисками включает в себя как создание инфраструктуры управления рисками, так и непосредственное управление рисками в ходе реализации проектов. Создание инфраструктуры предполагает анализ существующих процессов с точки зрения эффективности управления рисками, перепроектирование процессов, приобретение и освоение инструментов управления рисками, создание базы знаний, существенных для анализа рисков, создание и развитие средств, методов обмена информации о рисках.

1.4.4.3. Управление рисками в проектах

Управление рисками в ходе проекта включает в себя следующие действия, представляющие собой логически связанный цикл управления:

- Определение рисков. Определение (выявление) на этом этапе полезно использование различных классификаций рисков.
- Анализ рисков. Анализ может проводиться для отдельного проекта на этапе принятия проекта, в ходе его реализации и для совокупности проектов.
- Планирование конкретных действий по предотвращению рисков или уменьшению их последствий. Эти действия могут изменять первоначальный план проекта, приводить к перераспределению ресурсов и (или) изменению состава работ.
- Отслеживание состояния рисков. Возможно выделение нескольких категорий риска, отличающихся по значимости, и установление различной периодичности для различных категорий
- Контроль предполагает корректировку проверку эффективности предпринятых мер по минимизации рисков, планов минимизации рисков.

- Коммуникация представляет собой важнейший элемент успешного управления рисками. Коммуникация находится в центре цикла управления рисками, включается в каждый из вышеперечисленных этапов. Коммуникация обеспечивает обмен информации о рисках между всеми участниками проекта.

Существует зависимость между рисками и выбранной моделью ЖЦ. Модель водопада способствует обнаружению негативных последствий рисков на более поздних этапах [171, стр. 108]. В случае ориентации на выпуск в срок, возникновение проблем на более поздних этапах неизбежно приведен к сокращению этапа тестирования, и, следовательно, к росту числа необнаруженных ошибок в выпущенном продукте. Именно по этому ориентация на выпуск в срок требует применения итеративных моделей ЖЦ ПО.

Некоторые авторы говорят об абсолютном преимуществе вариаций итеративного ЖЦ (рациональный унифицированный процесс, модель процесса разработки MSF). Соображения управления рисками влияют на выбор модели жизненного цикла. Если существует уверенность в том, что требования клиента будут выявлены и реализованы верно в ходе одной итерации, а выбранная технология обеспечит надежную работу приложения, то модель водопада может быть наилучшим вариантом, ее преимуществом станет простота управления проектом, эффективность приложения усилий проектировщиков и разработчиков.

2. ПРИНЦИПЫ И МЕТОДЫ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Анализ методологий, моделей и стандартов выполняется с использование выделенных в предыдущей главе основных групп. Основное влияние на их содержание и, соответственно, структуру, оказывает их основное назначение. Нами выделены две группы:

- *методологии управления проектами;*
- *методологии управления организацией.*

Проектные методологии содержат сходный набор элементов: описание модели жизненного цикла разработки ПО, описание ролей проектной команды, описание разрабатываемых артефактов и рекомендации по использованию вспомогательных инструментальных средств, таких как языки графического моделирована, метрики и т.д. Основное отличие между проектными методологиями связано с применением различных моделей жизненного цикла (ЖЦ ПО).

Методологии управления ИТ-организацией, как правило, имеют более сложную структуру. К этой группе относятся модели зрелости (CMM, CMMI), стандарт оценки возможности процессов (SPICE), индивидуальный процесс разработки (PSP, TSP). Объединяющим принципом служит наличие рекомендуемых процессов, которые служат не для достижения непосредственных целей проектов разработки ПО, но для достижения надежных и предсказуемых результатов в любом из проектов, который выполняет организация. В основе всех методологий этой группы лежат идеи TQM. Дополняют процессные методологии оценки процессов. Например, для CMMI разработана методологии SCAMPI, формализующая процедуру оценки процессов [280]. Модель IDEAL представляет собой методологию улучшения процессов разработки ПО [260]. Структура методологий этой группы постоянно развивается, постоянно идет поиск наиболее понятного представления рекомендаций. Например, модель CMMI имеет два варианта представления (непрерывное – continues и последовательное – staged).

2.1. Анализ структуры методологий и стандартов управления компанией

Анализ структуры моделей и стандартов этой группы показывает сходство их структуры, как правило, они содержат глоссарий, модель процессов, модель зрелости – описание желаемого состояния практик и процедур с возможностью оценить соответствие состояния организации требованиям стандарта. В основе структурированного изложения современных методологий этой группы лежат так называемые «цели». Практика использования понятия «цель» для построения логической структуры методологий находит сегодня широкое применение, не ограничивается сферой разработки ПО. Например, стандарт Cobit (Control Objectives for Information and related technologies, Стандарт управления, контроля и аудита информационных и связанных технологий) построен вокруг понятия цель [218].

В популярных моделях и стандартах практически отсутствуют примеры использования лучшей практики, а также описания преимуществ выполнения тех или иных практик. По аналогии со стандартом Cobit следует ожидать развития методологий управления разработкой ПО и в этом направлении.

2.1.1. Модели зрелости технологических процессов

Первой среди моделей зрелости и наиболее известной является CMM (Capability Maturity Model — модель зрелости технологических процессов), разработанная институтом технологии программирования в 1993 г. (Software Engineering Institute, SEI) [266]. В 2002 г. была разработана интегрированная модель зрелости, имеющая расширенный набор практик и два варианта представления [214].

CMM представляет собой структурированную, взаимосвязанную и упорядоченную совокупность ключевых практик, выполнение которых обеспечивает достижение целей. Она содержит описание более 340 ключевых практик, которые позволяют не только реализовать основные процессы, но и создать условия для их успешного выполнения и дальнейшего улучшения.

Под ключевой практикой понимается или отдельная работа или совокупность условий, которые должны быть созданы. Ключевые практики областей процессов модели объединены в следующие группы – общие свойства (common features). Краткое описание структуры модели CMM представлено в Приложении 4.1.2

Согласно CMM организация может находиться на одном из пяти уровней зрелости, которые характеризуют возможный разброс результатов по качеству производимого ПО. Уровни зрелости процессов, и соответствующие им ключевые области процесса (key process areas) представлены в Приложении 4.7.2.1.

Аналогичную структуру имеют все последующие модели зрелости (P-CMM, Acqusition CMM). Каждый уровень зрелости описывается совокупностью ключевых областей — отдельных направлений деятельности в управлении разработкой ПО. Надо отменить, что большинство ключевых областей лежат в сфере вспомогательных процессов разработки.

Основной идеей моделей зрелости является идея последовательного перехода организации с одного уровня на другой. При этом переход может осуществляется только в строго определенной последовательности. На начальном уровне процессы разработки в основном не определены, успех зависит от усилий отдельных разработчиков. Организация, находящаяся на начальном (initial) уровне CMM, может разработать качественное ПО, но в начале разработки невозможно составить план-график проекта, а полученные в результате завершения проекта оценки трудоемкости реализации тех или иных функций не могут быть использованы для планирования следующего проекта, поскольку отсутствует повторяемость процесса, да и сам процесс не определен. Оценка уровня «зрелости» зависит от того, в какой степени процесс определен, управляется, измеряется, контролируется и осуществляется.

CMM позволяет решить две задачи:

- улучшения процессов разработки ПО (планирования, разработки, внесения изменений);

- оценки процессов разработки ПО (определения состояния текущих процессов, определение важных проблем, существующих в организации и способов их решения.

Модель зрелости используется заказчиком для оценки способности поставщика, позволяет оценить риск при заключении контрактов на разработку ПО. Модель CMM не содержит инструкций или рекомендаций по собственному внедрению, кроме общих рекомендаций, таких как не пропускать уровни.

В результате объединения подходов CMM и SPICE в 2002 г. SEI была разработана интегрированная модель зрелости CMMI [214].

CMMI имеет два варианта представления: непрерывный (continuous) и поэтапный (staged). Структура непрерывной модели CMMI представлена В Приложении 4.7.2.2.

В отличие от CMM, в непрерывном представлении CMMI ключевые практики группируются вокруг целей, а те, в свою очередь, сгруппированы в областях процессов.

Интегрированная модель зрелости технологических процессов (CMMI) выделяет четыре области процессов:

- управление процессами,
- управление проектами,
- разработка
- поддержка.

Модель CMMI является следующим шагом в процессе осмысления практик ИТ компании, в сравнении со стандартом SPICE, который будет рассмотрен далее.

2.1.2. Стандарт оценки возможностей процессов SPICE (ISO 15504)

Стандарт ISO 15504 SPICE (Software Process Improvement and Capability dEtermination) представляет собой руководство для оценки процессов разработки ПО. Основные цели SPICE:

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

В основу SPICE легли такие стандарты, как: STD (Compita), CMM (SEI), TRILLIUM (Bell), SQPA (HP), ISO 9001, ISO/IEC 12207-1-194, ISO/IEC 12119-1995 и другие.

Стандарт SPICE состоит из девяти частей. Описание содержания и структуры стандарта SPICE представлено в приложении 4.7.4

Модель процессов SPICE приставляет собой лишь перечень базовых и общих практик, но стандарт не предлагает описания логики процесса. Следует отметить, что практики SPICE, входящие в один процесс, не всегда могут быть логически объединены в действительный бизнес-процесс, иногда объединенные в рамках одного процесса практики в действительности принадлежат различным процессам. Например, процесс ORG.1 Организация бизнеса включает в себя следующие практики: формирование видения бизнеса, распространение видения бизнеса в организации, установление культурных ценностей, организация комплексных команд, организация мотивации, планирование карьеры. Очевидно, что перечисленные практики не могут быть объединены в некоторую упорядоченную последовательность. На наш взгляд, «процесс» в понимании стандарта SPICE не является, по сути, процессом, а служит лишь контейнером для объединения практик, которые не привязаны к ЖЦ ПО, а реализуются в бизнес-контексте.

В соответствие со стандартом SPICE, для оценки процесса оценивается каждая входящая в него базовая практика (наличие практики, степень соответствия практики целям процесса), в результате составляется оценивающий вектор. Таким образом, стандарт SPICE не дает реальных оценок характеристикам процесса: сроку разработки ПО, производительности процесса разработки (тестирования) в функциональных точках и т.д. Кроме того, для проведения оценки не требуется наличия полного формализованного описания процесса, включая его логику (условия выполнения практики - условия входа и выхода: перехода к следующей работе, возврат к предыдущей). Стандарт SPICE предполагает оценку реальных экземпляров процессов, образом, получает свои характеристики в динамике, полученные оценки являются статическими (структурными) и характеризуют структуру процессов.

Модель улучшения процессов в SPICE трехмерна. По одной оси откладывается эффективность работы (удовлетворенность заказчиков, качество продукции и продуктивность), по другой - возможности персонала, и по третьей - адекватность процесса. Таким образом, траектория улучшения процесса выбирается в трехмерном пространстве, где улучшения по каждой из осей идут параллельно с улучшениями по другой. В основе модели улучшения SPICE лежит PDCA (Plan-Do-Check-Act) цикл Джурана.

Модель процессов SPICE представлена в Приложении 4.5.6. Она является наиболее полной моделью среди рассмотренных моделей.

2.1.3. Персональный процесс разработки (PSP)

Уотс Хэмфри (Watts Humphrey) в 1994 году в SEI разработал технику, позволяющую повысить результативность труда индивидуального разработчика ПО через изменение его собственного процесса разработки, которая была названа «Персональный процесс разработки» (PSP, Personal Software Process) [241]. Эта техника не заменяет, но дополняет модели и стандарты, регламентирующие работу компании либо проектной команды.

Качество работы группы разработчиков определяется качеством работы ее участников. Собственный процесс разработки (PSP, Personal Software Process) представляет собой технику, которую могут использовать разработчики, что бы улучшить качество и производительность своей собственной работы.

Оценивая перспективы практического использования своих рекомендаций, Хэмфри писал «… незначительное число разработчиков используют такие известные методы, как организованный подход к проектированию, просмотр кода или определенные процедуры тестирования. PSP основывается на изучении практики работы разработчиков (программистов). Распространение современных улучшенных способов часто замедляется по той причине, что разработчики должны быть уверены в их эффективности, прежде чем начать их применять» [242].

В программу обучения разработчиков, как правило, не включаются методы измерения эффективности собственной работы, способы планирования. Изучая

языки программирования, разработчики решают упрощенные задачи, в результате чего вырабатывается стиль работы, адаптированный для этого класса задач. Но, столкнувшись с крупными проектами, требующими одновременной работы большого числа специалистов, разработчики испытывают существенные сложности. PSP использует практики, применяемые в крупных проектах, и масштабирует их для использования в средних и малых проектах.

PSP включает в себя описание методов, а так же инструмент оценки эффективности использования этих методов в собственной работе. Возможность определить, оценить и отследить свою работу обеспечивает ее лучшее понимание, которое позволяет выбирать наиболее эффективные методы. В результате формируется процесс разработки, который может быть оценен и использован другими разработчиками. В отличие от других методик (CMM, MSF), PSP реализован не как стандарт, а как программа обучения разработчиков на специализированных курсах.

Базовый процесс предполагает учет времени, затрачиваемого разработчиком на различных стадиях, и количество обнаруживаемых дефектов. На этом уровне разработчики начинают использовать стандарты кодирования, метрики размера (сложности). Все предложения по улучшению процесса, выявленные проблемы, фиксируются в специальной форме предложения по улучшению процесса (PIP, Process Improvement Proposal) в структурированном виде.

Процесс планирования предполагает оценку времени и ресурсов, необходимого для выполнения задачи. Кроме того, составляется отчет по тестированию и отслеживается статус проекта. Для планирования необходимо понимать зависимость между размером и сложностью программы и временем ее создания.

Управление качеством предполагает более детальный анализ совершаемых ошибок, их классификацию, выявление причин появления. Предоставляются статистические данные об ошибках различных типов на тысячу строк кода для разработчиков с различным уровнем квалификации. Просмотр собственного кода осуществляется с использованием контрольных списков типичных ошибок.

Разрабатываются критерии, характеризующие законченность (завершенность) определенной стадии: проектирования, кодирования, документирования.

Циклический процесс предполагает адаптацию существующей технологии разработки для создания больших программ. Весть проект разбивается на несколько циклов, в ходе которых проектируется, разрабатывается и тестируется определенная часть проекта, например, может быть разработана базовая версия модуля, которая будет дорабатываться на следующих инерциях. Нужно заметить, что речь не идет об использовании определенной модели жизненного цикла. Для выполнения более крупных проектов Хамфри рекомендует переходить от собственного процесса разработки к командному процессу (TSP, Team Software Process).

Одной из основных метрик PSP является количество дефектов, обнаруживаемых на этапе просмотра проекта, кода, в процессе компиляции и в ходе тестирования в единицу времени. Эта метрика используется для демонстрации эффективности просмотра кода для обнаружения ошибок. PSP обуславливает рост производительности в больших проектах за счет сокращения необходимого времени тестирования всей системы, поскольку повышается качество отдельных компонентов[2]. Время тестирования напрямую зависит от количества ошибок, а абсолютное большинство ошибок допускаются небольшой частью команды (правило 20/80).

В том случае, если в отдельных компонентах обнаруживается большое количество ошибок, все ресурсы проекта расходуются на их исправление, в результате не уделяется необходимое внимание обеспечение другим аспектам качества (например, полноте документации или удобству интерфейса). Большое количество ошибок затрудняет планирование проекта, при определенном количестве ошибок на планирование не остается времени (ресурсов). Уменьшая количество в отдельном компоненте, разработчик облегчает задачу качественного выполнения всего проекта. Следует заметить, что получение преимущества от

[2] Под компонентом понимается отдельная процедура, модуль, и т.д. – любая составляющая, которая может быть протестирована отдельно.

снижения количества ошибок в компонентах зависит от организации процесса разработки в компании и должно планироваться менеджментом.

Сегодня компании-разработчики могут самостоятельно осваивать новые методы управления разработкой, такие как PSP. Описание PSP представляет собой подробную методику, кроме того, SEI предлагает курсы обучения, предназначенные как для студентов, так и для квалифицированных разработчиков.

2.2. Модели жизненного цикла ПО в методологиях и стандартах

Модели жизненного цикла ПО (ЖЦ ПО) являются центральной частью проектных методологий. Впервые понятие «модель жизненного цикла» ввел в широкий обиход Уинстон Ройс в 1968 г. Он описал каскадную и водопадную модели жизненного цикла разработки ПО, которые являются каркасом большинства проектов, выявил недостатки каскадной модели и определил, как она может быть трансформирована в итеративную модель [190]. Этапы водопадной модели У. Ройса представлены в приложении 4.4.1.

Модель жизненного цикла определяет состав и последовательность шагов по созданию и использованию ПО, отражающей его различные состояния, начиная с возникновения необходимости и заканчивая выходом из употребления: анализ, проектирование, кодирование и тестирование с указанием последовательности их выполнения, категорий исполнителей, и используемых методов. Методология PRINCE различает два понятия “product life span” и “project life cycle”, согласно приводимым определениям жизненный цикл продукта охватывает лишь часть жизненного цикла проекта, ограниченную периодом разработки.

Различают три формальные модели ЖЦ: каскадную, поэтапную с промежуточным контролем и спиральную. В соответствии с *каскадной моделью* переход на следующий этап может происходить только после завершения предыдущего. *Поэтапная модель с промежуточным контролем* — это итаративная модель с обратными связями между этапами.

Спиральная модель предполагает циклическое выполнение всех этапов каскадной модели, в результате чего реализуемость технических решений проверяется с помощью прототипов. Каждый виток спирали соответствует

созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы.

Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача - как можно быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований.

Модель процесса разработки представляет собой, по сути, описание жизненного цикла проекта разработки, если не учитывать вспомогательные и организационные процессы.

В дополнение к рассмотренным выше моделям ЖЦ ПО Рассел Арчибальд выделяет в качестве самостоятельных моделей жизненного цикла разработки ПО прототипирование, быструю разработку (RAD, Rapid Application Development), инкрементное построение [10].

Протипирование предполагает одновременную разработку функциональных требований и топологическое проектирование.

Быстрая разработка основана на использовании эволюционирующего прототипа, который не отбрасывается.

Инкрементное построение основано на разбиении большого объема проектно-конструкторских работ на последовательность более малых составляющих частей.

При этом все вышеперечисленные модели ЖЦ ПО он относит к «прогнозирующим моделям», которые ставят оптимизацию выше адаптивности, включая и спиральную модель.

К группе адаптивных методологий он относит XP, SCRUM и Адаптивную разработку программного обеспечения (Adaptive Software Development, ASD).

Адаптивные методологии основываются на итеративной и инкрементальной разработке. Итеративный процесс не является достижением последних десятилетий. Он возник практически одновременно с классическим последовательным подходом. В основе итеративного подхода лежат концепции

управления качеством, например, ориентированная на повышение качества методика Уолтера Шеварта из Bell Labs, состоящая из серии коротких циклов шагов по планированию, реализации, изучению и действию (plan-do-study-act, PDSA) [279]. Гилб [230, 231] и Ричард Залтнер исследовали PDSA применительно к разработке программного обеспечения. Брайана Рэнделла и Ф.В. Зерчера [270] и М.М. Леман [251] из IBM разработали рекомендации по методике итеративной разработки.

Существует несколько моделей и стандартов, в той или иной степени регламентирующих жизненный цикл, большинство из них относятся к заказному ПО (автоматизированным системам АС, и др.), и кроме непосредственно ЖЦ регламентируют также и процессы разработки:

ГОСТ 34.601-90 распространяется на автоматизированные системы и устанавливает стадии и этапы их создания. Кроме того, в стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют каскадной модели жизненного цикла.

ISO/IEC 12207:1995 стандарт на процессы и организацию жизненного цикла. Распространяется на все виды заказного ПО. Стандарт не содержит описания фаз, стадий этапов. Модель жизненного цикла - структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

Методика Oracle CDM (Custom Development Method) по разработке прикладных информационных систем под заказ - конкретный материал, детализированный до уровня заготовок проектных документов, рассчитанных на прямое использование в проектах АС с опорой на инструментарий Oracle. Степень адаптивности CDM ограничивается тремя моделями ЖЦ: "классическая" (предусмотрены все работы/задачи и этапы), "быстрая разработка" (Fast Track), еще более сильно ориентированная на использование инструментов моделирования и программирования Oracle, "облегченный подход",

рекомендуемый в случае малых проектов и возможности быстро прототипировать приложения.

Rational Unified Process (RUP) предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется циклом разработки, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы.

Суть работы в рамках RUP – это создание и сопровождение моделей, а не бумажных документов, поэтому этот процесс привязан к использованию конкретных средств моделирования (UML), а так же конкретной технологии проектирования и разработки (OOA, OOP).

Модель процесса разработки MSF сходна с RUP, так же включает четыре фазы: анализ, проектирование, разработка, стабилизация.

В водопадной модели все этапы выполняются последовательно один за другим, при этом следующий этап может начаться только после завершения предыдущего.

Стандарт ГОСТ 34.601 закрепляет основные идеи каскадной модели, содержит описание стадий и этапов создания автоматизированных систем. ГОСТ серии 19 содержат описания требований к программной документации, которая создается при каскадной разработке.

ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств. В модели стандарта ИСО/МЭК 12207-99 все работы, выполняемые в течение жизненного цикла программных средств, распределены по пяти основным, восьми вспомогательным и четырем организационным процессам. Часть процессов, регламентированных стандартом, выполняются не компанией-разработчиком, а заказчиком и/или аудитором. Модель не включает все процессы компании-разработчика, а только те, которые непосредственно связаны с выпуском программного обеспечения. Модель процессов стандарта 12207

представлена в Приложении 4.5.1 Следует отметить "динамический" характер стандарта определяется способом задания последовательности выполнения процессов и задач, при котором один процесс при необходимости вызывает другой или его часть.

Методика Oracle CDM (Custom Development Method) предлагает простую одноуровневую модель процессов. Модель CDM представляет интерес как образец модели процессов методологии разработки бизнес-систем. Анализ модели показывает высокий уровень прагматизма. Модель прямо обозначает основные виды работ, которые выполняются при проектировании, разработки и внедрении бизнес-систем. Например, выделены такие процессы, как исследование существующих систем (для решения вопросов интеграции, аппаратного обеспечения и т.д.), процесс переноса данных. Именно фокус на интеграции и данных отличает разработку бизнес-приложений.

CDM адаптирована для разработки информационных систем для бизнеса. Процессы состоят из последовательностей задач, задачи разных процессов взаимосвязаны явно указанными ссылками.

Рациональный унифицированный процесс (RUP). Рациональный унифицированный процесс (Rational Unified Process, RUP) был разработан Филиппом Кратченом по заказу Rational Software. RUP предлагает одну из наиболее полных моделей процессов итеративной разработки.

Внутри каждого процесса сосредоточены связанные между собой артефакты (программы и документы) и деятельности (задачи). На сегодняшний день RUP — это набор методов и практик для улучшения качества и эффективности процесса разработки программ, а также базовая архитектура и веб-инструментарий для обмена разработками через Интернет. Методология и инструментарий были переданы IBM сообществу разработчиков с открытым кодом бесплатно в 2005 г.

RUP обеспечивает строгий подход к распределению задач и ответственности внутри организации-разработчика. Повышение производительности коллективной разработки достигается за счет использования накопленного опыта по созданию ПО, посредством руководств, шаблонов и наставлений по пользованию

инструментальными средствами для всех критически важных работ, в течение жизненного цикла создания и сопровождения ПО.

Для использования RUP необходимо использовать специализированные инструменты организации взаимодействия между участниками проекта. Предоставляя каждому члену группы доступ к той же самой базе знаний, вне зависимости от того, разрабатывает ли он требования, проектирует, выполняет тестирование или управляет проектом - RUP гарантирует, что все члены группы используют общий язык моделирования, процесс, имеют согласованное видение того, как создавать ПО. В качестве языка моделирования в общей базе знаний используется Unified Modeling Language (UML), являющийся международным стандартом.

Особенностью RUP является то, что в результате работы над проектом создаются и совершенствуются модели. Вместо создания бумажных документов, RUP опирается на разработку и развитие семантически обогащенных моделей, всесторонне представляющих разрабатываемую систему. RUP – это руководство по тому, как эффективно использовать UML. Стандартный язык моделирования, используемый всеми членами группы, делает понятными для всех описания требований, проектирование и архитектуру системы.

RUP поддерживается инструментальными средствами, которые автоматизируют большие разделы процесса. Они используются для создания и совершенствования различных промежуточных продуктов на различных этапах процесса создания ПО, например, при визуальном моделировании, программировании, тестировании и т.д.

RUP – это конфигурируемый процесс, пригоден как для небольших групп разработчиков, так и для больших организаций, занимающихся созданием ПО.

RUP организует работу над проектом в терминах последовательности действий (workflows), продуктов деятельности, исполнителей и других статических аспектов процесса с одной стороны, и в терминах циклов, фаз, итераций и временных отметок завершения определенных этапов в создании ПО (milestones), т.е. в терминах динамических аспектов процесса, с другой. Модель процессов RUP представлена в Приложении 4.5.4

В отличие от процессов SPICE и ISO 12207, процессы RUP выполняются не последовательно в течение жизненного цикла, а несколько раз, в зависимости от числа итераций на каждой фазе. Существуют модификации RUP для конкретных средств разработки, например, IBM Web Sphere, Microsoft Web Solutions.

Модель жизненного цикла объектной кластерной разработки. Технология разработки влияет на выбор модели ЖЦ ПО. Наиболее популярной технологией разработки систем сегодня является объектно-ориентированное проектирование и программирование. Специфика объектного дизайна (инкапсуляция, наследование и полиморфизм) позволяет ускорить разработку и существенно снизить риски за счет параллельной работы на кластерами – логическими группами классов.

Модель жизненного цикла объектной разработки основана на параллельной работе над несколькими кластерами, использующих спецификации ранее завершенных кластеров. Описание деталей модели ЖЦ объектной разработки приведены в Приложении 4.4.2

Этапы проектирования и реализации не имеют четких границ. Данный подход, как и другие, требует анализа осуществимости (feasibility study), на основе которого принимается решение о выборе модели ЖЦ ПО.

Лучше, чем традиционные подходы, кластерная модель повышает эффективность управления проектом путем гибкого распределения ресурсов. Во избежание рассогласований необходимо регулярно отслеживать текущие состояния кластеров. Оптимальным является контроль руководителем проекта хода работ через определенное время, например, один раз в неделю. Тем самым гарантируется наличие на каждой стадии текущей демонстрационной версии, не обязательно охватывающей все аспекты системы, но готовой для показа клиентам, менеджерам и другим заинтересованным лицам. Кроме того, это позволяет своевременно информировать участников проекта и устранять любую несогласованность между кластерами.

Возможность параллельной разработки в рамках кластерной модели обеспечивается за счет механизмов скрытия информации ОО-метода. Кластеры могут зависеть друг от друга, например кластер графического интерфейса может нуждаться в классах коммуникационного кластера для реализации удаленного

терминала. Благодаря абстрактным данным можно работать над кластером, даже если кластеры, от которых он зависит, еще не завершены. Эта возможность реализуется при наличии законченной спецификации необходимых классов на основе их официального интерфейса, заданного в краткой форме или в виде отложенной версии.

Последний этап жизненного цикла кластера, обобщение, не имеет аналогов в традиционных подходах. Его цель - шлифовка классов для их подготовки к повторному использованию. Успех политики повторного использования требует наличия определенной *культуры повторного использования* у каждого участника, выделения достаточных ресурсов для расширения соответствующих возможностей исходных версий классов.

Включение обобщающего этапа в официальную модель процесса - вопрос политики. Успешная реализация практики повторного использования требует резервировать дополнительные средства и время для обобщения в каждом проекте.

Преимуществом кластерной модели является устранение резких переходов между отдельными этапами, использующими различную нотацию, системы взглядов и персонал (аналитики, проектировщики, программисты...). Несоответствия между анализом и проектированием, проектированием и реализацией, реализацией и развитием являются причиной роста затрат и увеличения количества ошибок в традиционной схеме разработки ПО.

С начала и до конца основой разработки являются классы, что гарантирует полное соответствие между описанием проблемы и ее решением. Прямое отображение упрощает диалог с заказчиками и пользователями и содействует развитию, поскольку все участники пользуются терминами одних и тех же основных концепций. Это та часть поддержки расширяемости, которую обеспечивает ОО-метод. Использование единой структуры облегчает корректировки, выполняемые в обратном направлении, неизбежные для поступательного в целом процесса разработки ПО.

2.3. Структура проектных методологий

Практически все проектные методологии содержат такие части, как описание рекомендуемой модели ЖЦ ПО, модель команды и модель процессов. Описание моделей процессов содержит описание рабочих продуктов. Современные проектные методологии содержат описание условия их эффективного использования, включая требования к характеристикам проекта (объем проекта, характер рисков, технологии разработки, квалификация персонала, а также географическое расположение команды). Кроме того, проектные методологии могут дополнять различные классификации, например, классификации рисков.

2.3.1. Методологии быстрой (адаптивной) разработки (SCRUM, XP, Crystal)

Все методологии быстрой (адаптивной, agile) разработки основываются на принципах, сформулированных в Agile Manifesto [256]. К методологиям быстрой разработки относятся методология SCRUM, семейство методологий Crystal и XP. Методология MSF практически полностью удовлетворяет требованиям Agile Manifesto, но вынесена нами в самостоятельный раздел, поскольку ее структура является более полной в сравнении с XP или SCRUM.

Кен Швабер и Майкл Бидл предложили методологию SCRUM [277]. В основе методологии лежит цикл разработки. Цикл разработки в SCRUM составляет 30 дней. Ежедневно в ходе одного цикла проводятся 15 минутные совещания, которые называются SCRUM, делается обзор сделанного с момента проведения последнего совещания, определяются проблемы и разрабатывается план работ до следующего совещания. В ходе одного 30-дневного цикла реализуются требования Sprint Backlog, упорядоченные в соответствие с приоритетами, установленными заказчиками.

Алистер Коберн разработал семейство методологий Crystal [113, 114]. В семейство входят четыре методологии Crystal Clear, Crystal Yellow, Crystal Orange и Crystal Red. Методологии отличаются размером команды разработчиков, количеством различных ролей в составе команды.

Методологии Crystal отличают следующие особенности:

- Построены вокруг людей и коммуникаций;
- Адаптируются, чтобы соответствовать параметрам. Представляют собой набор компонент, из которого команда проекта собирает методологию;
- Рассчитаны на команды, размещенные в одном месте;
- Существует возможность растягивания методологии.

Методологии Crystal могут при необходимости заимствовать элементы XP, PSP и др. В их основе лежит инкрементная разработка. Продолжительность инкремента 1-4 месяца. Рабочие семинары проводятся регулярно: до, после и желательно – в середине инкремента.

Основной чертой методологии являются не соблюдение практик, а поддержка коммуникаций и достижение двух целей кооперативной игры – выполнение проекта и реализация следующего проекта.

Методологии Crystal содержат рекомендации по их применению. В приложении 4.8.2 представлена структура методологий Crystal, позволяющая делать выбор методологии в зависимости от масштаба проекта и уровня риска.

Кент Бек формализовал методологию экстремального программирования [18]. Методология разработки XP (eXtreme Programming) имеет самый короткий цикл разработки и самую малочисленную команду. В методологи XP используются следующие понятия:

- *Спайк (Spike)* – очень простая программа, разрабатываемая, что бы исследовать возможные решения.
- *Пользовательские истории (User Stories)* – варианты использования системы, которые пишутся заказчикам.
- *Приемочный тест (Acceptance Test)* – вариант тестового примера, который разрабатывается для пользовательских историй. Тесты должны быть автоматизированы, потому что выполняются очень часто.

Отличительным элементом XP является обязательный рефакторинг – периодически проводимый реинжиниринг кода. Коллективное владение кодом и парное программирование также стали элементами методологии XP.

XP представляет собой достаточно жесткую с организационной точки зрения методологию, задает модель процесса разработки, состав разрабатываемых артефактов и структуру команды.

2.3.2. Система MSF

Система моделей MSF (Microsoft Solution Framework) была разработана как часть общей методологии Microsoft (Microsoft Enterprise Framework), которая так же включает в себя MRF (Microsoft Readiness Framework) и MOF (Microsoft Operations Framework). В настоящий момент не предусмотрена сертификация компаний на соответствие MSF, система моделей используется не как стандарт, а как руководство, отвечая не на вопрос, «что делать?», а предоставляя конкретные рекомендации и отвечая на вопрос «как?».

MSF используется на этапе проектирования, разработки приложений и внедрения приложений. Первоначально в момент создания в 1994 году MSF представляла собой описание решения проблем бизнеса с помощью технических средств, постепенно обобщая лучшую практику групп разработки, внедрения, клиентов и партнеров Microsoft.

В отличие от других методологий большая часть моделей представляют собой описание конкретных примеров разработки с комментариями.

MSF не является стандартом, на соответствие которому компания может пройти сертификацию, популяризация моделей обеспечивается их включением в состав обязательного экзамена MCSD “Анализ требований и проектирование приложений”.

2.3.3. Проектная методология PRINCE

В отличие от методологий, рассмотренных выше, фокусом методологии PRINCE является не процесс разработки ПО (и ЖЦ ПО как его основа), а процесс управления проектом, где разработка - это только составная часть. Существенное внимание в PRINCE уделяется инициации проекта и закрытию проекта.

Методология PRINCE *(PRojects IN Controlled Environments*, проект в контролируемой среде) была разработана в Великобритании в 1989 г. Central Computer and Telecommunications Agency (CCTA) по заказу UK Office of Government Commerce как стандарт ведения ИТ-проектов. Со временем границы использования этого стандарта расширились, и сегодня не ограничиваются только сферой ИТ. Методология PRINCE является расширением концепции управления

проектами. В 1996 г. вышел стандарт PRINCE2, который стал стандартом де-факто в управлении проектами в Великобритании. Методология имеет ограниченную сферу применения – описывает требования к организации проекта разработки и внедрения информационных систем для бизнеса (коммерческих приложений).

Проект направляется бизнес-сценарием («business case»), который определяет интересы организации (заказчика), требования к разрабатываемым продуктам и другим результатам. План организуется вокруг создаваемых продуктов, сфокусирован на получении результатов, а не выполнении определенного набора работ. Модель процессов PRINCE представлена в приложении 4.5.5.

PRINCE не является заменой методологий быстрой разработки (SCRUM, Crystal и XP), она может гармонично дополнять методологии этого класса. MSF содержит в себе практики, регламентированные PRINCE, эти методологии являются взаимозаменяемыми.

2.4. Анализ методов и инструментальных средств получения количественных оценок

Использование численных (количественных) оценок в управлении разработкой ПО является необходимым условием применения подходов TQM и процессного управления. Использование численных (статистических) методов для управления процессами разработки является показателем уровня зрелости процессов, обеспечивает более стабильные и надежные характеристики планов, используется для более точной оценки рисков и обеспечивает эффективное использование ресурсов. Статистические методы лежат в основе непрерывного управления качеством, предоставляют необходимую для улучшения процессов информацию (информацию об отклонениях). Согласно концепции TQM, численное определение целей является необходимым условием эффективного управления.

Для количественного измерения характеристик процессов и продуктов в сфере разработки ПО используются «метрики». Существует формально определение метрики как критерия, построенного на основе двух или более

взаимосвязанных мер, в этом случае синонимом понятия «метрика» будет понятие «индикатор». Мера (измерение) представляет собой результат единичного измерения некоторого изменяющегося значения, например размера, времени работы или количества ошибок. Например, метрикой производительности является количество функциональных точек (Functional Point, FP) в час, а метрикой качества – количество дефектов на тысячу строк кода. В некоторых случаях понятия «мера» (measure) и «метрика» (metrics) используются как взаимозаменяемые.

Метрики позволяют сравнивать работу отдельных сотрудников, отделов, отслеживать ход выполнения проектов. Метрики позволяют сравнивать показатели деятельности компании со средними значениями в отрасли (benchmarking). Консалтинговые компании и организации предоставляют эти данные на коммерческой основе [263]. Таким образом, метрики используются как для принятия оперативных решений в ходе выполнения проектов, так и стратегического управления.

В Табл. 5 представлен структурированный подход к использованию количественных методов в управлении разработкой ПО. В основе подхода лежит выделение объектов и категорий измерений.

Табл. 5 Объекты и категории измерения метрик

Объект	Категория измерения	Вопрос
График проекта и оценка степени выполнения	Степень выполнения отдельных работ	Будет ли проект завершен вовремя? Какова степень завершения отдельных работ?
Ресурсы и затраты	Персонал Финансовые результаты Доступные ресурсы	Проводилась ли работа в соответствии с планом? Велики ли отклонения? Укладывается ли проект в запланированный бюджет? Имеются ли в наличии необходимые ресурсы?
Качество продукта	Ошибки и дефекты Сложность Переписывание кода	Может ли ПО быть поставлено покупателю? Были ли решены предыдущие проблемы? Является ли разрабатываемое ПО тестопригодным и пригодным для дальнейшей поддержки? Какие усилия затрачиваются на внесение изменений и исправление ошибок?
Увеличение размера кода, Неизменность	Размер продукта и неизменность	Изменяется ли размер продукта и его функциональность? Изменяются ли требования к продукту?
Производительность	Зрелость процессов Производительность	Сможет ли разработчик выполнить требования с учетом существующих ограничений? Какова вероятность успешного завершения проекта (отдельной работы)? Соответствует ли производительность процессов (отдельных этапов, конкретных сотрудников) современным требованиям?
Техника	Использование имеющихся компьютеров и другого оборудования Влияние технологии	Соответствует ли производительность имеющихся технических средств существующим потребностям? Привело ли применение новой технологии к тем результатам, которые ожидались?

Стандарт ISO/IEC 9126 предполагает использование внешних (external) и внутренних (internal) метрик [51]. Внешние метрики позволяют оценить характеристики ПО в процессе функционирования внутри системы. Системой может быть тестовая либо реальная среда эксплуатации ПО. Внутренние метрики измеряют статические характеристики ПО (и других продуктов: спецификаций, планов тестирования, пользовательской документации и документации разработки). Внутренние метрики используются на ранних стадиях разработки: дизайн, программирование, а внешние – на стадии тестирования и эксплуатации.

В основе классификации метрик могут лежать цели, стадии (этапы) разработки, объект измерения, технология. В Табл. 6. представлена разработанная

нами двумерная классификация метрик, где одним измерением является объект измерения, а вторым – измеряемая характеристика.

Табл. 6. Классификация метрик

ОБЪЕКТ ИЗМЕРЕНИЯ	ОБЪЕМ	СЛОЖНОСТЬ	КАЧЕСТВО	РИСКИ
Программный продукт:				
Система (внешние метрики)	Размер дистрибутива, размер инсталлируемой системы	Количество одновременных запросов, распределенность, использование разных платформ	Время выполнения запроса, Метрики надежности (время работы без сбоев, время восстановления), Среднее время между отказами (MTFB) Корректность программы (степень соответствия спецификации)	Вероятность прекращения функционирования (ухудшения)
Код (внутренние метрики)	Строки кода (SLOC), Объектные метрики: количество классов, количество методов на класс	Цикломатическая сложность; Объектные метрики:	Документированность (отношение числа строк комментариев к объему); Ожидаемое число дефектов	Отношение числа ошибок (дефектов) к объему; отношение сложности к объему
Документация	Количество слов, предложений, таблиц, рисунков	Уровень образования, количество сокращений, специальных терминов	Полнота Непротиворечивость; Согласованность интерфейса в спецификациях программных компонент	-
Процесс	Число функций, роли процесса	-	Качество процесса: количество дефектов/объем (MAELOC); Количество дефектов, обнаруженное в единицу времени на различных этапах (просмотр, компиляция, тестирование)	-
Проект[3]	Метод функциональных точек (FPUG, Mark II)	Наличие циклов, ветвление	Время жизненного цикла; стоимость продукта	Время завершения, необходимые ресурсы (отклонение от бюджета и расписания)

Ристо Невалайнен[4] предлагает классификацию метрик, имеющую три измерения:

- *Измерение продукта:* размер, качество, удовлетворенность клиента;

[3] Проект можно представить как экземпляр процесса. В этом случае метрики проекта и процесса будут совпадать.

[4] Ристо Невалайнен (Risto Nevalainen) является одним из разработчиков стандарта SPICE, он участвует в проведении семинаров Центра программных систем (SPS).

- *Измерение процесса:* оценка уровня процесса, отклонения от требуемого (целевого) значения;

Измерение ресурса: время, трудоемкость, квалификация, приверженность менеджмента, характеристики инструментов, характеристики поведения.

Метрики объема и сложности используются для планирования продолжительности проекта, метрики качества и рисков позволяют качественно оценивать характеристики объектов (процессов, программных продуктов), добиваться их необходимого значения.

Объект измерения (оценки) выделяется в соответствие со стандартом ISO 9000 см Приложение 4.2.

Табл. 7. Метрики: объект измерения и измеряемые характеристики

Объект измерения	Характеристики
Физические строки кода	размер, повторное использование, исправления
Логические строки кода	
Рабочее время персонала	стоимость, распределение ресурсов, степень загрузки
Сроки выполнения отдельных этапов	Работы, расписание, степень выполнения, готовность к завершению, вехи
Проблемы и ошибки (дефекты)	качество, тенденция к улучшению

Управление процессами с помощью количественных оценок становится повторяемой практикой только на четвертом уровне по CMM (Level 4 Measurable), на котором находится не более 15% всех компаний разработчиков ПО. Использование численных методов предполагает не только численную оценку характеристик, но и установление численно-заданных целей в области качества ПО и характеристик процесса.

Сбор и накопление метрик требует использование специализированных автоматизированных средств.

2.4.1. Метрики программного кода

Подсчет строк кода (SLOC, software lines of code) является одной из первых метрик, использованных для оценки размера программного обеспечения. Эта

метрика использовалась для оценки производительности труда программистов, для оценки сложности программ, для прогнозирования времени тестирования. Преимуществами данной метрики является простота использования, понятность, а недостатками – зависимость от языка разработки, а так же негативное влияние на стиль программирования для получения более высоких значений данной метрики (см. п. 3.5.6).

Не смотря на простоту метрики, необходимо уточнить правила подсчета, определить, учитываются ли строки комментариев, пустые строки. Как и для любой другой метрики необходимо обеспечить ее последовательное применение. Очевидно, что метрика SLOC может использоваться только для сравнения ПО, разработанного с использованием единого языка программирования, либо разных языков, имеющих одинаковых уровень абстракции и синтаксис. Даже стандарты оформления программного кода оказывают влияние на метрику SLOC. Для сравнения ПО, разработанного с использованием различных языков программирования, используется альтернативная метрика MAELOC (Million assembly-equivalent lines of code) – метрика объема миллионов строк кода в эквивалентных командах языка ассемблера.

Объектные метрики. Объектно-ориентированный анализ, проектирование и разработка являются лидирующим направлением в программной индустрии в настоящий момент, находятся на этапе зрелого расцвета, разработаны не только теоретические основы, но и эффективные инструментальные средства объектного анализа и разработки[5]. Объектно-ориентированный подход состоит в формировании набора объектов, свойственных данной предметной области, в результате чего закладывается фундамент для решения всех задач в этой области. Следовательно, для оценки качества объектного кода или модели необходимо использовать специализированные метрики.

---

[5] Был разработан унифицированный язык моделирования (UML), ставший стандартом объектного анализа и проектирования. Большинство современных инструментов моделирования поддерживают UML, например Rational Rose, ARIS Toolset, Visio 2000 (и выше). Популярные языки разработки, такие как Java, C++ (C#), Visual Basic .Net являются полностью объектными.

Традиционные метрики предполагают независимую оценку кода и структур данных, в то же время объектные метрики позволяют одновременно оценивать функциональность и данные как единый объект. Объектные метрики используют такие понятия, как методы, классы, связывание, наследование. Метрики используются для оценки внутренней структуры объекта, его взаимодействия с другими сущностями (объектами), эффективности алгоритмов, и даже психологические метрики, характеризующие возможности разработчиков создавать, понимать, модифицировать и поддерживать ПО.

Традиционные метрики могут использоваться для оценки объектно-ориентированного ПО, например, для оценки методов объектов целесообразно использовать метрику цикломатической сложности, подсчет строк кода, подсчет комментариев. В приложении 4.9.1 приведены метрики оценки классов.

2.4.2. Метрики объема работы

Метрика функциональных точек (FP, Functional Points) используется для оценки необходимого времени разработки на ранних этапах проекта (на этапе концептуального и логического проектирования). Для использования метрики функциональных точек достаточно иметь требования к программному обеспечению. Точность оценки зависит от уровня детализации требований. Кроме того, используя терминологию FP (входы, выходы, запросы, внутренние и внешние хранилища данных[6]) можно согласовывать требования к программному обеспечению. Эта метрика так же используется для оценки производительности труда разработчиков и объема работы. FP может использоваться без изменения на всем протяжении жизненного цикла проекта. Использование прозрачного количественного алгоритма для оценки трудоемкости проекта делает более конструктивным диалог между заказчиком ПО и разработчиком.

Существует большое количество модификаций метода функциональных точек: IFPUG Алана Албрехта (Allan Albrecht), MARK II Чарьза Симонса (Charles Symons), Feature Points, Experience PRO. Отличие между ними состоит в составе

[6] «Хранилище данных» употребляется как перевод «data store», не имеет отношения к технологии Data Warehouse в данном контексте

элементов, числе уровней сложности и соответствующему количеству функциональных точек, следовательно, для сопоставления результатов оценки, полученных с помощью разных методов, необходимы дополнительные преобразования. MARK II предлагает подход на основе логических транзакций (logical transactions), состоящих из входов, процесса и выходов, что обеспечивает действительную независимость от языков и средств разработки. Не смотря на декларируемую независимость от средств разработки, существуют определенные предпочтения использования той или иной модификации FP, в зависимости от назначения ПО. Например, MARK II предназначен для оценки информационных бизнес-приложений, ориентированных на работу с данными, но не учитывает сложность алгоритмов, что необходимо для оценки научного и инженерного ПО, а так же не учитывает специфической сложности систем реального времени.

Целесообразно использовать один метод оценивания в организации. Между различными методами существует сопоставимость в заданных диапазонах значений, например, до 400 FP IFPUG и MARK II сопоставимы между собой, но при больших значениях MARK II дает большие оценки.

В настоящий момент Международная организация по стандартизации (ISO) разрабатывает стандарт ISO 14143, в котором содержится концепция метода функциональных точек и практика использования этого метода. Стандарт не содержит описания конкретного метода, а определяет характеристики, которыми должен обладать любой метод, который может быть отнесен к методу функциональных точек.

Метод функциональных точек имеет большое концептуальное значение в теории управления разработкой программного обеспечения, поскольку этот метод является единственной существующей альтернативой метрик, использующих количество строк кода, метод функциональных точек не зависит от языка разработки и может использоваться на ранних этапах проекта (например, метод функциональных точек используется в COCOMO II в модели прототипа, см. п. 2.4.4).

К недостаткам метода следует отнести существенную сложность использования, обусловленную необходимостью дополнительного обучения

сотрудников использованию специализированных средств сбора метрик. Кроме того, в отличие от подсчета строк кода, который может быть выполнен с использованием автоматических средств, метод функциональных точек основывается на экспертных оценках сложности, следовательно, точность оценок зависит от квалификации экспертов и будет повышаться со временем по мере роста квалификации экспертов.

На точность оценок влияет качество спецификаций (функциональных требований) и адекватность их отображения в конкретных технических решениях (предварительном техническом проекте), поскольку метод использует не сами функциональные требования, а характеристики программного решения.

Таким образом, использование метода может потребовать изменения в процессе разработки ПО, если до использования метода на этапе контрактных переговоров использовались только функциональных требования, то после перехода к оценке продолжительности проекта и объема работ с использованием функциональных точек потребуется предварительное проектирование (проектирование интерфейса, базы данных, бизнес-логики).

Очевидно, что большая точность оценок на ранних этапах потребует больших затрат в начале проекта, возможно, до подписания контракта. Существует определенная двойственность в использовании метрик, и в особенности, метода функциональных точек, как средства управления рисками. С одной стороны, используя метод функциональных точек можно получить более точные оценки, и избежать риска недооценки сложности (трудоемкости проекта), кроме того, этот метод косвенно приводит к лучшей детализации (полноте) спецификации, уменьшая риск невыполнения существенных требований. Но, с другой стороны, использование метрик увеличивает затраты, в том числе и затраты, которые фирма несет до заключения контракта, следовательно, увеличиваются риски.

2.4.3. Метрики сложности

Существует два аспекта оценки сложности: оценка сложности процесса и оценка сложности объектов разработки. Информационная сложность программ

характеризуется количеством различных типов и структур данных, поступающих на вход и выдаваемых программой.

Программные модули (функции, процедуры, методы классов) являются простыми компонентами, поэтому они доступны для количественного анализа. Программное обеспечение представляет собой совокупность программных компонент, поэтому его сложность характеризуется количеством связей между ними, сложностью взаимодействия. Измерение сложности необходимо не только для оценки потребности в ресурсах, но и для планирования тестирования, поскольку известно, что между сложностью и количеством ошибок существует прямая зависимость.

Распространенной метрикой сложности является метрика цикломатической сложности Мак Кейба (McCabe) [258]. Описание метрики приведено в приложении 4.9.2. Метрика сложности позволяет оценить потребность в тестировании и документировании. Существующая практика предполагает оценку количества ошибок и продолжительности тестирования в течение процедуры тестирования на основе существующего потока обнаруживаемых ошибок. Основными недостатками этого подхода является невозможность предсказать продолжительность тестирования до того, как произойдет существенное снижение интенсивности потока ошибок, а так же прогнозировать поток ошибок на стадии эксплуатации системы. При этом тестовые примеры строятся в соответствие со спецификациями. Распространенным методом тестирования является так же структурное тестирование, при котором выполняются все ветки алгоритмов. Используя понятие цикломатической сложности можно составить более эффективный план тестирования, при котором минимальное количество тестов обеспечивает проверку всех участков кода.

Кроме того, может существовать ограничение на уровень сложности, в этом случае модули (процедуры), имеющие сложность выше установленного уровня (например, цикломатическая сложность v(G) = 15) должны быть перепроектированы для упрощения. Причиной для такого подхода является существенный рост потенциального количества ошибок в коде, при достижении определенной сложности [258 стр. 13]. Существуют определенные правила

преобразования алгоритмов, позволяющие уменьшить цикломатическую сложность.

Кроме оценки сложности логики необходимо так же оценивать ее качество, что бы обеспечить легкость поддержки кода (maintainability) следует минимизировать количество управляющих структур. Метрика внутренней сложности (essence complexity metrics) позволяет оценить уровень структурированности кода. Оценка сложности программной системы (например, информационной системы) не может быть получена непосредственно из оценок сложности отдельных составляющих.

2.4.4. Метрики стоимости

Одной из известных моделей стоимости проекта является COCOMO (Constructive Cost Model) – модель прогнозирования затрат на разработку, впервые опубликованная Барри Боэмом (Barry Boehm) в 1981 году. Модель была обновлена в 1998 году, в ней было учтено повторное использование кода, объектно-ориентированный подход, а так же зрелость процессов разработки (COCOMO II) [209].

COCOMO II представляет собой модель использования совокупности различных метрик для прогнозирования стоимости разработки. В Приложении 4.9.3 представлена структура модели. Модель предполагает использование альтернативных способов оценки размера проекта, например подсчет строк кода (SLOC) и метод функциональных точек (FP). В случае использования функциональных точек модель предоставляет возможность перевода результатов этой метрики в SLOC, используя таблицу переводя для выбранного языка программирования [209 стр. 20].

Кроме того, модель учитывает, что на ранних этапах проекта может отсутствовать информации о платформе, разработчиках, требуемой функциональности. Модель учитывает точность оценки размера программного обеспечения на различных этапах проекта в зависимости от полноты информации, предлагается статистика по проектам, показывающая прогнозируемое и фактическое значение длительности проекта. Учитывая тот факт, что на ранних

этапах информации недостаточно, модель предлагает использовать укрупненные факторы затрат в начале проекта, переходя к детальным факторам на последующих стадиях. Предлагаются три модели прогнозирования стоимости: Модель прототипа (Application Composition Model), Модель этапа проектирования (Early Design) и Модель детальной разработки (Post Architecture).

Модель прототипа предназначена для создания прототипов приложения на самых ранних этапах разработки. Модель этапа проектирования предполагает использования метода функциональных точек и семи факторов затрат. Модель детальной разработки основывается на выбранной модели жизненного цикла, учитывает возможные риски разработки. Для оценки размера может применяться подсчет строк кода или метод функциональных точек, с учетом повторного использования. Модель детальной разработки использует семнадцать факторов затрат и пять факторов масштабирования проекта.

Отличительной особенностью модели является оценка организации: квалификации персонала, зрелости процессов для прогнозирования стоимости разработки. Интересно, что время выполнения проекта трудоемкостью один месяц будет более чем в три раза дольше при любом значении коэффициента масштабирования, длительность проекта при трудоемкости в пять месяцев оказывается сопоставима с трудоемкостью, в диапазоне от десяти до тысячи месяцев продолжительность проекта оказывается менее двадцати четырех месяцев. На наш взгляд аналитическая формула отражает реальное положение дел, результаты подтверждаются фактическими данными – действительно, не один проект не выполняется быстрее, чем за три месяца, но большинство сложных проектов удается завершить в течение двух лет. Таким образом, появляется инструмент оценки продолжительности проекта, который можно использовать на стадии оценки концепции проекта, оценки дизайна, подготовки контракта.

В модели учитывается эффект от повторного использования. Используя статистические данные NASA Software Engineering Laboratory по 3000 проектов повторного использования COCOMO использует «нелинейный эффект» повторного использования. Минимальные затраты на использование без модификации (затраты на оценку, выбор, сравнение) оказываются равны 5% в

большинстве случаев. Даже небольшие изменения вызывают значительный рост затрат, связанный с необходимостью изучения ПО, которое должно быть изменено, а так же необходимостью анализа интерфейсов.

Затраты возрастают от 5% до 55% при необходимости изменить от 0 до 10% кода, затем происходит снижение роста затрат и при увеличении числа необходимых изменений до 70% затраты составляю примерно 70% стоимости и дальше растут практически линейно.

3. ОПТИМИЗАЦИЯ ВЫБОРА И АДАПТАЦИИ МЕТОДОВ УПРАВЛЕНИЯ РАЗРАБОТКОЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Организация процессов выбора

Решение задачи выбора связано с решением ряда технических, организационных и концептуальных задач. Для решения задачи выбора оптимальной методологии необходимо:

- Сформировать понимание возможности и необходимости выбора.
- Обеспечить формирование библиотеки (архива) моделей, стандартов, техник. Объекты хранения должны быть классифицированы, определено их назначение, область применения, связь с другими элементами. Должны быть разработаны процедуры обновления архива, обеспечивающие получение новых версий материалов и включения новых материалов. Практически невозможно сформировать абсолютно полную и актуальную базу материалов, для этого потребовались бы слишком большие усилия. Поэтому необходимо определить требования к полноте и актуальности репозитория.
- Обеспечить процесс постоянной самоидентификации компании, включая определение ее культуры, маркетинговой стратегии, политику в области качества. Необходимо однозначно сформулировать те принципы и убеждения, которые лежат в основе выбора методологии. (Например, в виде тезисов-утверждений: только полное определение всех требований позволит успешно завершить проект, технический специалист может хорошо владеть только одним программным средством и т.д.).
- Обеспечить постоянное обучение и переобучение сотрудников компании, в первую очередь, менеджеров проектов и специалистов отдела качества.

Таким образом, необходимо создать, институализировать и непрерывно улучшать следующие процессы:

- обучения сотрудников инструментальным средствам;
- формирования библиотеки инструментальных средств;
- выбора проектных и процессных инструментальных средств (используя метод верификации принципов);
- адаптации проектных и процессных инструментальных средств (с учетом специфика проекта, требований клиента, бизнес-целей организации и т.д.);

- оценки существующей практик использования инструментальных средств (оценка процессов, оценка проектов).

Далее мы рассмотрим проблемы процесса выбора инструментальных средств и предлагаемые нами методы реализации его отдельных практик.

Адекватный выбор модели процессов является предпосылкой эффективного управления на основе процессного подхода. Выбирая модель процессов, и, следовательно, определенный стандарт, необходимо учитывать цели и существующие ограничения. Например, если предпринимаются первые попытки формализации процесса, можно воспользоваться моделью SPICE для сравнения существующей практики и рекомендаций стандарта. Модель процессов ISO 12207 следует использовать для согласования стадий и этапов ЖЦ ПО между клиентом и исполнителем. Модель процессов CDM позволит сконцентрироваться на специфике разработке информационных систем для бизнеса, проектировании структуры данных, взаимодействии с заказчиком, следовательно, ее можно использовать для построения (оптимизации) процессов разработки заказных систем для бизнеса.

Ни один процесс не может быть пригоден для всех случаев разработки ПО. Практический подход предполагает, что необходимо иметь возможность адаптации (конфигурирования) процесса для широкого спектра приложений. Процесс должен обеспечивать экономию при больших масштабах и возврат инвестиций за счет использования общего духа самого процесса, широкой автоматизации и общих архитектурных образцов и компонентов.

Сравнительный анализ теоретических и практических моделей процессов показывает, что основные отличия проявляются на уровне выделения процессов (модели процессов) и структуры (алгоритмов), в то время как конечный состав работ (функций) практически совпадает.

Основной функцией менеджмента является принятие решений [65, стр. 22]. Специфика управления разработкой ПО определяет наличие пограничной области решений, которые не могут быть однозначно классифицированы как чисто технические или управленческие, например, решение, какую технологию

использовать для разработки, какими средствами пользоваться для управления версиями или автоматизации тестирования. Большинство таких решений должны приниматься в два этапа, на первом определяется набор допустимых решений, и оцениваются технические возможности, затем выбирается окончательное решение с учетом стратегии компании, маркетинговых прогнозов. К управленческим решениям следует относить решения относительно программы метрик, совершенствования процессов разработки, обучения персонала.

Использование референтных моделей процессов позволяет существенно снизить трудозатраты и повысить качество формализации (моделировании) бизнес-процессов [220]. Этот подход хорошо зарекомендовал себя при внедрении информационных систем на промышленных предприятиях. К сожалению, большая часть рассмотренных нами моделей процессов не могут быть использованы как референтные, поскольку они содержат только перечни процессов, практик, но не включают в себя логику процесса.

Успешность использования стандартов и моделей зависит от реалистичности исходных ожиданий, которые с ними связывались. Нереалистичными являются ожидания, что формальное следование методологии управления ИТ-проектом и выполнение требований стандартов на СМК и моделей зрелости (CMMI) позволяет получить высокое качество программного продукта независимым от квалификации программистов и других специалистов, позволить полностью избежать зависимости проекта от конкретных исполнителей. Высокий уровень зрелости и процесса поваляет уменьшить риски потери или искажения информации, повысить эффективность работы.

3.2. Влияние внутренних особенностей организации и воздействия внешней среды на выбор

3.2.1. Влияние культуры организации на решение задачи выбора

В отличие от технологических процессов производственных компаний, процессы разработки программного обеспечения лишь в определенной степени ограничены технологией разработки. Существование объективной гибкости в построении процессов делает их в большей степени зависимыми от влияния

различных факторов, не связанных с непосредственно с технологией. Среди таких факторов выделяется «культура» компании.

Понятие «культура» эквивалентно совокупности понятий «традиции», «привычки» и «общепринятые ценности». Культура в наибольшей степени обусловлена историей компании, реализуемой маркетинговой стратегией, структурой собственности. Месторасположение компании (страна, город) и род деятельности влияют на культуру компании. Любая организация имеет свою культуру, даже вновь организованная компания.

Эдгар Шейн дает следующее определение понятия корпоративная культура: Корпоративная культура - «комплекс общих посылок, убеждений и ценностей, которые возникли в рамках организации, для того чтобы выстроить ее отношения с внешней и внутренней средой, и которые передают новым членам организации с целью управления их действиями в этих условиях» [205].

Культуру составляют существующие символы, привилегии, система отличий и вознаграждений. Существует большое количество работ, посвященных вопросом организационных изменений (change management), в том числе улучшению бизнес-процессов [150, 188, 197, 229], в которых анализируются элементы культуры компании, которые препятствуют изменениям. На существование связи между историей компании, карьерой менеджеров и организацией процессов указали Р. Каплан и Д. Нортон [248, стр. 12]. В результате наблюдения за крупными американскими компаниями они сделали вывод, что в каждой индустрии существуют один или несколько доминирующих процессов. Например, в нефтеразработке доминирует технический процесс очистки и основной фокус делается на стоимость и технологию, а маркетингу не уделяется должное внимание, в то время как производители товаров массового спроса чаще всего основное внимание уделяют процессам продаж и маркетингу, а не процессам совершенствования технологий и инновациям.

Разработчики высоких технологий имеют совершенные процессы проектирования, но процесс производства может нуждаться в улучшении. Традиционно руководители компаний приходят из подразделений, ответвленных

за приоритетные направления, что способствует сохранению приоритетов в компании.

Чарльз Хэнди (Charles Handy) описал четыре модели культуры организации и дал им имена греческих бого в [237]:

- Организация, управляемая личностями (как правило, основателем или собственником) – Зевс;
- Организация, управляемая процедурами и инструкциями – Аполлон;
- Проектная организация – Афина;
- Творческая организация – Дионис.

Как правило, та или иная модель редко встречается в чистом виде, обычно присутствуют черты нескольких моделей, но одна из них является доминирующей, следовательно, приведенная классификация позволяет выявить основные особенности организации. Организация, управляемая процедурами и инструкциями, сможет использовать преимущества подходов ISO 9000, CMMI, проектные организации могут выиграть от методологий PRINCE2, PMBOK, методологий управления разработкой ПО и внедрения ИС. Творческие организации смогут использовать части методологий для решения отдельных задач. Организации, управляемые личностями, смогут использовать методологии успешно, только если у менеджмента не меняется видение в отношении целесообразности использования формализованных подходов.

Следует заметить, что крупные компании-разработчики программного обеспечения, особенно предназначенного для военных целей, управляются с помощью процедур и инструкций, что обеспечивает выполнение целей, придает гармонию и устойчивость, но является препятствием изменениям, не позволяет быстро реагировать на кризисные ситуации. Личное управление обеспечивает наибольшую маневренность, но может стать фактором, сдерживающим рост организации. Кроме того, личное управление связано с высоким риском Наибольшую сложность для управления имеют творческие организации, но в некоторых случаях только эта модель обеспечивает выполнение задачи, поскольку

многие квалифицированные творческие специалисты предпочитают работать в условиях если не абсолютной, то значительной свободы.

В управлении проектом разработки ПО может преобладать один из подходов:

- Проектный подход (менеджерский);
- Технический подход (инженерный);
- Командный подход (психологический).

Отличие подходов обусловлено различным видением успешного выполнения проекта. Для проектного подхода это достаточность ресурсов, выполнение всех фаз/этапов проекта, соблюдение проектной методологии. Для технического подхода главным является выбор и реализация эффективной программной архитектуры, использование современных средств проектирования, разработки и тестирования. Командный подход видит залог успеха в установлении и сохранении эффективных коммуникаций между членами проектной команды, установлению «командного духа», а также высокой мотивации и уверенности в успехе проекта.

Нужно заметить, что во многих случаях выбор модели не осуществляется сознательно, а происходит в результате стихийного развития, например, организация может пройти путь от личного управления к проектному управлению, а затем прийти к бюрократической модели (что происходит с большинством крупных стареющих организаций), или уменьшить степень контроля и предоставить больше свободы сотрудникам, при этом не осознавая доминирующую на каждом этапе модель, и, следовательно, не учитывая ее слабые стороны.

В некоторых случаях компании осознанно используют сильные стороны моделей, например компания Microsoft предоставляет большую свободу рабочим группам, благодаря чему достигаются преимущества творческой организации, в то же время стратегические решения принимаются собственником единолично, обеспечивая динамичность личного управления. Можно сделать вывод, что организация должна целенаправленно использовать преимущества моделей,

применяя те или иные методы управления на различных уровнях управления, в различных проектах, подразделениях и направлениях деятельности. Следует учитывать модель культуры организации при создании (улучшении) процессов. Специфические особенности корпоративной культуры Microsoft и ее влияния на качество разрабатываемых приложений рассматривается в [5].

Отличительной чертой разработки ПО является потребность в высококвалифицированном персонале для выполнения всех видов работ, использование сложного оборудования. В этих условиях эффективная работа возможна только в обстановке сотрудничества, когда менеджмент не принуждает, а оказывает содействие. Процессы разработки должны строиться с учетом этих требований, то есть следует отказаться от функций жесткого контроля работы отдельных сотрудников, заменив их дружескими обзорами, обучением и аттестацией.

Коммуникации и командная работа требуют доверия и определенных навыков. Проблема коммуникации является основной в управлении рисками, а командная работа – это основа организации процесса разработки ПО. Таким образом, ми видим, что процессы и культура компании тесно взаимосвязаны.

Культура компании является основным фактором, обеспечивающим повторяемость процессов, устойчивость процессов к изменению/увеличению требований, сокращению сроков. Сотрудники компании должны понимать, почему процесс разработки построен именно таким образом и к каким последствиям может привести нарушение процесса. Только процесс, построенный на основе собственного опыта и требования заказчика будет повторяемым, и только для такого процесса существует возможность улучшения. Именно по этой причине для перехода с одного уровня CMM на другой требуется несколько лет.

Использования «тяжелых» процессов ведет к тому, что результаты проявляются на более поздних этапах. Например, вместо относительно коротких сроков разработки «тяжелые» процессы обеспечивают сокращение количества дефектов, и, соответственно, затрат на обслуживание. В большинстве компаний политические реальности таковы, что менеджеры и разработчики получают премии и прочие поощрения по результатам работы за месяц, квартал, год. Это

приводит к тому, что корпоративная культура во многих ИТ-организациях будет всячески сопротивляться тяжелым процессам и поддерживать легкие [85, стр. 139].

Отношения компании с ее клиентами и поставщиками являются частью культуры организации. Настоящее улучшение бизнес-процессов может быть достигнуто, если задействована все цепочка создания ценности – от поставщика до покупателя. Выстраивая свои отношения с контрагентами, рассматривая их как партнеров, разработчик должен точно определять все формы своего взаимодействия с ними. Следовательно, при организации процесса, построении модели логики процесса, необходимо большое внимание уделять граничным функциям процесса (внешним интерфейсам – external interfaces), входы и выходы которых лежат за границами организации.

Очевидно, что организация процесса разработчика (эти соображения касаются в основном разработчиков заказных систем) зависит от организации процесса заказчика, поскольку между процессами устанавливаются взаимосвязи. Учитывая, что процесс зависит от культуры организации, можно сделать вывод, что культуры заказчика и разработчика испытывают взаимное влияние, и, более того, некоторая совместимость культур является предпосылкой успешной работы.

История создания организации и проблемы, с которыми она сталкивается, как правило, взаимосвязаны. Эдвард Йордон описывает культуру организации, которая складывается при систематическом выполнении «безнадежных проектов» (проектов со сжатым более чем на 50% сроком разработки или уменьшенным числом разработчиков) [84]. В том случае, если организация постоянно выполняет «безнадежные проекты», то она сама изменяется, при этом изменения затрагивают все уровни организации: охватывают средства и технологии, процессы и методологии планирования, используемые организацией-разработчиком ПО. Следовательно, при организации процессов необходимо учитывать, над какими проектами организация собирается работать, какой подход намеревается использовать. В том случае, если только часть проектов организации являются «безнадежными», то потребуется, как минимум, два варианта процессов: один – для обычных проектов, другой - для «безнадежных».

3.2.2. Учет влияния производственной и маркетинговой стратегий

Специфика бизнес-моделей компаний-разработчиков ПО связана с особенностями рынка экономики знаний. М. Кусумано выполнил анализ бизнес-моделей и стратегий компаний-разработчиков ПО, он показал, что успех ИТ-компании зависит не от технологической, а от бизнес-составляющей. [219]. На основе анализа статистики крупнейших ИТ-компаний, он обнаружил тенденцию перехода компаний от продуктовой к сервисной модели в долгосрочной перспективе, проявляющуюся в увеличении доли услуг в обороте ИТ-компании. Выявленная тенденция имеет большое значение для нашей работы, поскольку динамическое изменение бизнес-модели предполагает соответствующую адаптацию ее процессов, и, следовательно, предполагает выбор проектных методологий.

Не смотря на приоритет бизнес-решений перед технологическими решениями, для достижения устойчивых экономических результатов работы компании в долгосрочной перспективе необходимо эффективно управлять проектами и процессами разработки. Согласование методов и моделей управления проектами разработки, методологий, используемых в проектах, в частности, с бизнес-стратегией компании приобретает большое значение.

В для анализа связи между стратегией компании и используемыми моделями разработки мы выделяем два типа бизнес-моделей компаний, разрабатывающих ПО:

- *ориентированные на рынок (продукт),*
- *ориентированные на процесс.*

В основу этого деления положена не декларируемая миссия компании, а приоритеты, которыми руководствуется менеджмент в принятии решений. В зависимости от этого приоритет имеют или процессы маркетинга и продаж или проектирования и разработки. То есть в первом случае существует большая вероятность, что менеджмент компании будет формироваться из маркетинговых служб, а генеральный директор будет больше внимания уделять процессу продаж. Во втором случае менеджмент будет формироваться из технических специалистов,

и процессы разработки будут иметь высокий приоритет. Можно отметить, что процессу проектирования в высокотехнологичных отраслях соответствует процесс разработки ПО.

Следует отметить, что восприятие важности процессов может быть различно на разных уровнях управления [см. например SPICE Part 7, 6.1 Management responsibility and leadership]. Менеджмент среднего уровня традиционно больше значения придает завершению отдельных проектов, и не заботится об улучшении процессов в целом. С другой стороны, руководители иногда недооценивают влияние мероприятий по улучшению процессов на текущую деятельность.

Маркетинговая стратегия компании оказывает существенное влияние на требования к организации процессов разработки, и, следовательно, влияет на выбор инструментальных средств. Маркетинговая стратегия включает в себя стратегию в области цен, выбор каналов распределения, а так же требования к качественных характеристикам производимой продукции.

Прежде чем приступить к анализу влияния маркетинговой стратегии на организацию процессов разработки, рассмотрим возможные маркетинговые стратегии производителей программного обеспечения. Майкл Портер [269] выделяет три возможные маркетинговые стратегии: ценовое лидерство, дифференциация и сфокусированность. На наш взгляд, не смотря на существующую критику [236][7], эта классификация отражает маркетинговые стратегии, реализуемые компаниями. Кроме перечисленных стратегий можно говорить о специфической для разработки программного обеспечения стратегии технологического лидерства, направленной признание своих решений в качестве индустриального стандарта де-факто.[8]

Особенность стратегии ценового лидерства для разработчиков ПО, в отличие от производственных и сервисных организаций, состоит в том, что для его

---

[7] Новые теории не отвергают классификацию Портера, а лишь опровергают необходимость следовать только одной стратегии. Инновации позволяют достигать ценового лидерства и дифференциации одновременно, то есть обеспечивают технологическое лидерство компании.

[8] Яркий пример технологического лидерства проект Java компании Sun Microsystems.

достижения не требуется уменьшение затрат на разработку (сокращение функциональности и других качественных характеристик, ухудшение обслуживания). Причиной служит отсутствие (или незначительная величина) переменных затрат (стоимость производства копий). Исключение составляют затраты, связанные с сервисным обслуживанием и поддержкой пользователей.

Таким образом, стратегия ценового лидерства в меньшей степени предполагает оптимизацию производственных затрат (а следовательно, и оптимизацию процессов разработки), чем дифференциация или сфокусированность, поскольку ценовое лидерство достигается ни за счет снижения общих затрат на разработку, а благодаря снижению стоимости отдельной копии за счет роста числа продаж, то есть за счет рыночного лидерства. Эта стратегия реализуется производителями таких видов ПО, как операционные системы, среды программирования, офисные приложения.

Производители информационных системы для бизнеса добиваются рыночного лидерства (большей доли рынка) не за счет ценовой конкуренции. На рынке систем управления продается не программный продукт, а уверенность в росте эффективности бизнеса клиента, предлагаются идеи и опыт управления бизнесом. Процесс внедрения таких систем сопоставим со временем разработки менее сложной системы, удовлетворяющей требованиям отдельного заказчика.

Поэтому проблема оптимизации процесса разработки в большей степени актуальна для компаний, реализующих стратегии дифференциации и сфокусированности. Именно компании, занимающиеся разработкой на заказ, в первую очередь стремятся сертифицировать свою систему качества, формализовать процессы.

С другой стороны, именно компании, реализующие стратегию ценового лидерства, чаще всего являются монополистами, и, следовательно, получают монопольные прибыли. Именно эти компании имеют финансовую возможность инвестировать в модели и стандарты, разработку инструментов поддержки управления разработкой ПО. Поскольку сами эти компании не испытывают большой потребности в этих стандартах и инструментах, они разрабатывают

методологии, модели стандарты и инструменты не для себя, а для других компаний [171], использующие их инструменты разработки.

Основную угрозу ценовым лидерам составляет бесплатное ПО, что так же влияет на стратегию защиты. Изучение влияния наличия бесплатного программного обеспечения с открытым кодом (open source software) лежит за границами данного исследования. Но особенности организации процесса разработки бесплатного ПО представляют огромный интерес и должны стать результатом отдельного исследования.

На рынке программного обеспечения информационных систем и систем управления базами данных наблюдается влияние эффекта сноба – многочисленные примеры приобретения дорогостоящих информационных систем за их известность, без учета реальных потребностей бизнеса покупателя, и, иногда, без анализа действительных инвестиционных возможностей. Его влияние подкрепляется тем, что рынок информационных систем так же является рынком «лимонов», поскольку качества информационной системы не известны заказчику до ее внедрения. Но надо отметить, что информация о приобретении (внедрении) дорогой системы и в самом деле может улучшить имидж компании у контрагентов и конкурентов, привести к росту рыночной стоимости компании.

Ценовая информация заменяет детальное функциональное описание. Например, СУБД Oracle воспринимается всеми участниками рынка как самая надежная и производительная, SAP/R3 имеет репутацию самой полнофункциональной, гибкой и настраиваемой. Высокая стоимость подкрепляет эту репутацию. В случае появления нового продукта на рынке, его стоимость будет воспринята участниками рынка как информация о функциональны качествах продукта и характеристиках производительности и надежности. Большинство разработчиков информационных систем и систем управления базами данных реализуют стратегию дифференциации по Портеру. Действие эффекта сноба предопределяет отсутствие жесткой необходимости в снижении производственных затрат у производителей.

Разработчики заказного ПО составляют абсолютное большинство софтверных компаний, реализующих стратегию сфокусированности. Успех этих

компаний зависит от отношений с заказчиками, учета индивидуальных потребностей заказчиков.

Организация процессов разработки зависит от выбранной маркетинговой стратегии, которая определяет требования к разрабатываемому продукту, его функциональным характеристикам.

Приведенные на Рис. 2 признаки классификации программного обеспечения оказывают значительное влияние на организацию процессов разработки. В зависимости от формы продаж, назначения ПО, требования к его характеристикам, используемой технологии будет построен процесс разработки. Например, авторские права и назначение, влияет на процесс разработки через поток доходов от продаж, рекламы или реализации связанных продуктов.

Современная тенденция, наиболее заметно проявляющаяся в производстве, состоит в том, что производители решают, что должно быть произведено, а затем находят место размещения производства, используя критерий минимизации затрат. Следует предположить, что эта тенденция проявится во многих областях, в том числе и в разработке программного обеспечения. При этом компании, осуществляющее функции маркетинга: исследование потребностей рынка, продажи и т.д. будут передавать разработку тем компаниям, которые обеспечат самые низкие затраты с гарантированным качеством при определенном уровне риска. Произойдет специализация компаний, выполняющих определенные функции, возможно также дробление функции маркетинга, уже сейчас многие компании используют аналитические отчеты консалтинговых компаний, передавая им, таким образом, функцию первичного анализа рынка.

Большая специализация требует лучшей организации процесса разработки, поскольку необходимо синхронизировать процессы компании-разработчика с процессами нескольких компаний-заказчиков, при этом требования, предъявляемые к качеству и срокам, оказываются более жесткими, чем при разработке заказного ПО для конечного пользователя или разработке тиражируемого ПО

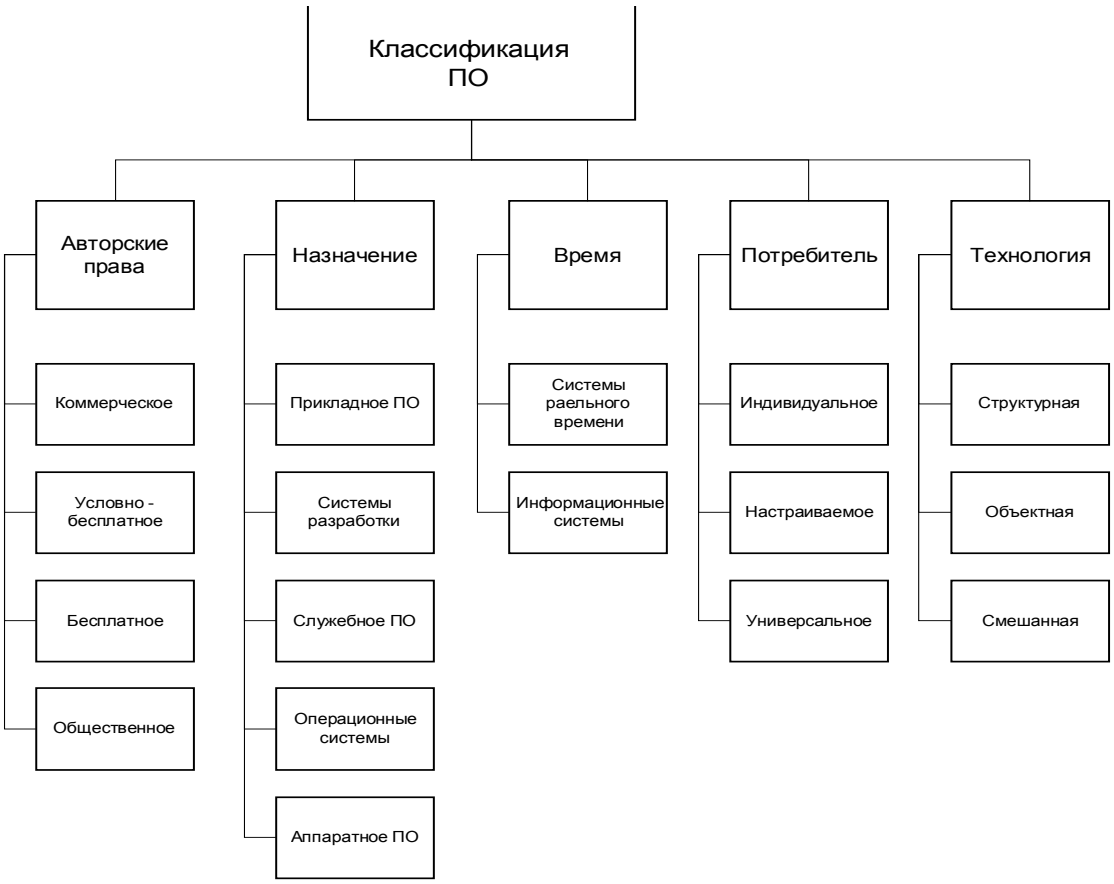


Рис. 2. Классификация программного обеспечения

Одной из основных задач, которую решают компании-разработчики *заказного* ПО, является обеспечение стабильного потока доходов, поскольку получение заказов происходит нерегулярно. Существует два распространенных способа сглаживания колебаний потока доходов и обеспечения стабильного потока доходов: сдача ПО в аренду вместо продажи (получение платы за использование – pay-per-use), получение доходов от текущего обслуживания.

Очевидно, что во втором случае возникает противоречие между интересами разработчика и заказчика, поскольку большое количество дефектов, необходимость изменений (обусловленных изначальной отсутствием необходимых функций или негибкостью ПО) приводят к росту доходов от текущего обслуживания. Очевидно, что идеальное ПО вообще не требует текущего обслуживания. С другой стороны, невысокое качество может привести к отказу от использования ПО, к потере рынка. Следовательно, определяя требования к качеству ПО, разработчик должен учитывать свои конкурентные преимущества и маркетинговую стратегию (см. п. 3.2.1).

Вопрос характера связи между качеством продукта (услуги) и процессами производства является важнейшим для управления. Одним из основных положений, которое лежит в основе моделей и стандартов на процессы и систему качества, является наличие прямой связи между организацией процессов разработки (их полнотой, повторяемостью) и способностью поставщика обеспечить качество продукта. Б. Боэм утверждает, что связь качества ПО с процессом его разработки аналогична зависимости между качеством технических средств и процессом их создания.

Методологии, модели и стандарты, разрабатываются крупными компаниями (Oracle, Microsoft, Hewlett Packard, Sun Microsystems) и государственными организациями (SEI, DoD), обуславливает проблемы с использованием эти стандартов для средних и малых компаний. Не существует успешных адаптаций ISO 9001 или CMM для компании со штатом менее двадцати-тридцати специалистов. Основной проблемой является неконкретный характер этих стандартов, они описывают, что нужно делать, но не дают конкретных рекомендаций. Исключением можно считать модель разработки приложений Microsoft (MSF, Microsoft Solutions Framework), которая содержит большое количество примеров (cases).

Эта проблема особенно актуальна для российских разработчиков ПО, поскольку российские компании в этом секторе в сто раз меньше американских. Затраты на внедрение стандартов и методологий являются фиксированными и не могут быть профинансированы при недостаточном масштабе деятельности[9].

---

[9] Абсолютное большинство российских компаний – разработчиков ПО имеют штат менее 150 человек и оборот менее 3 млн. долл./ год. Среднее число работающих в московских компаниях составляет 45 человек, в питерских — 30 человек, в региональных — 35.

В Москве доля компаний-разработчиков ПО, имеющих оборот более 3 млн долл., составляет 31,3%, оборот до 150 тыс. долл. имеют 18,8% московских компаний. Соответственно для Санкт-Петербурга — это 13,6 и 40,9% компаний, для регионов — 15,4 и 46,2%. (Market-Visio/EDC) http://crnep.ru/news.asp?ID=1518

3.2.3. Влияние технических характеристик проекта на состав команды проекта и распределение задач

При распределении работы между участниками проекта необходимо учитывать специфику технологии разработки и специфику разрабатываемого продукта. Проведенное нами обследование компаний-разработчиков программного обеспечения показывает, что можно выделить три крайних подхода в распределении работы между участниками проекта:

- **Функциональное разделение**. При этом подходе весь проект разбивается на крупные функциональные блоки, каждый из которых передается одному разработчику или команде разработчиков. Бизнес-анализ, проектирование разработка тестирование и внедрение (обучение пользователей) осуществляется теми же специалистами. Этот подход оправдан в том случае, если каждая функция имеет сложную бизнес-логику, требующую специальных знаний (например, модуль «кредиты» или «депозиты» в банковской интегрированной системе). Кроме того, количество средств разработки, используемых на различных этапах жизненного цикла для создания разных уровней приложения, должно быть минимально. В противном случае от разработчика потребуется много специальных знаний, связанных с использование средств проектирования и разработки. В том случае, если менеджмент придерживается функционального подхода, он может стать препятствием для освоения новых средств разработки. Если в системе уже существует «ядро»[10] - то его разработка может быть выделенной функцией. Именно такое разделение существует в компании CSBI при разработке ИС Банкир, где в качестве средства разработки используется СУБД Progress.
- **Уровневое разделение.** Современные бизнес-приложения строятся на основе многоуровневой архитектуры. Чаще всего встречается трехуровневая архитектура, предполагающая наличие уровня презентации (пользовательский интерфейс), бизнес-логика и база данных. При уровневом разделении вся работа по кодированию распределяется на основе спроектированных уровней и

[10] «Ядро» информационной системы реализует функции работы с данными. Все остальные бизнес-функции используют сервисы ядра для извлечения и сохранения данных. Ядро обеспечивает независимость бизнес-логики от структуры данных. Следует отметить, что подобные системы являются устаревшими. Высокая сложность разработки подобных систем, а так же возможность интеграции информационных систем различных компаний с использованием Internet привело к появлению самостоятельных сервисов, реализующих определенный бизнес-процесс или функцию, - происходит постепенный переход от систем ввода данных и обработки транзакций (OLTP, Online Transaction Processing) к распределенным системам поддержки бизнес-процессов, реализованных с помощью адаптивный сервисов (Adaptive Web Services).

используемых средств разработки. Использование этого подхода требует предварительного завершения стадии анализа и проектирования для всего приложения. Использование уровневого подхода оправдано, если для создания каждого уровня необходимо знание отдельных средств разработки, языков программирования[11]. Например, именно так были распределены работы при разработке системы для страховых брокеров, в которой использовался SQL Server 7.0 и Visual Basic. Именно этот подход к распределению ролей позволил разработчиком сфокусироваться на освоении новых версий средств разработки и решении ограниченного набора технических задач.

- **Поэтапное разделение.** В некоторых компаниях[12] существует тенденция вернуть (восстановить) разделение должностей сотрудников на «принимающих решения» и «исполняющих» их и не несущих ответственности. Практическая реализация этого подхода проявляется в разделении проектирования и кодирования. В процессе нисходящего проектирования сверху-вниз создается архитектура приложения, на основе которой создается каркас приложения. Этот подход особенно эффективен при использовании CASE средств, обеспечивающих генерацию кода. Наиболее распространен этот подход для создания объектно-ориентированных приложений. На основе концептуальной модели и сценариев использования проектируется логическая модель приложения, проектируется структура классов, определяются обязанности классов. После генерации кода на выбранном языке программирования из спроектированной модели необходимо только реализовать соответствующие методы классов. При поэтапном подходе можно без труда выделить роли бизнес-аналитика, архитектора инженера и кодировщика.

Очевидно, что все рассмотренные подходы имеют выраженные достоинства и недостатки и редко реализуются в чистом виде. Как правило, имеет место смешанный подход, позволяющий избежать недостатков чистых подходов, но использовать основные преимущества. Рекомендуется проявлять определенную гибкость в выборе подхода и избегать необдуманного копирования подходов одного проекта для другого. Например, успешно применяя функциональный подход для проекта, разрабатываемого с помощью Progress 4GL, следует

---

[11] Например, интерфейс может разрабатываться с использованием VB 6.0, бизнес-логика – C++, а уровень работы с данными (поддержание целостности, извлечение данных) – SQL Server Transact SQL.

[12] Например, в компании Digital Design физическое проектирование является задачей бизнес-аналитика, а в компании Siemens Business Services (Петербургское отделение разработки ПО) физическое проектирование осуществляется системными архитекторами.

использовать элементы уровневого подхода для проекта .Net. Следующая последовательность шагов реализует гибкий подход к созданию ПО[13]:

- Бизнес-аналитик определяет функциональные требования к ПО;
- Архитектор определяет состав компонент, с помощью которых будет реализована требуемая функциональность. На этом этапе определяются повторно используемые компоненты. Для крупного проекта архитектор распределяет компоненты между инженерами;
- Инженер проектирует компоненты и передает проект кодировщикам;
- Кодировщик программно реализует компоненты приложения.

Таким образом, выбор между функциональным и уровневым подходом осуществляется при определении состава компонент их распределении между инженерами (разработчиками).

С точки зрения управляемости проекта наилучшим является поэтапное разделение, которое обеспечивает документирование результатов каждого этапа. Если при функциональном разделении менеджер проекта вынужден довольствоваться лишь списком функций (описанием назначения разрабатываемых компонент), то поэтапное разделение позволяет отслеживать разработку отдельных компонент и их составляющих (бизнес-классов, экранных форм, хранимых процедур, отчетов). Следовательно, поэтапное разделение обеспечивает эффективное использование метрик для планирования проекта, управления рисками и качеством.

Нужно отметить, что именно преимущество поэтапного проектирование сверху-вниз с точки зрения отслежваемости проекта (а так же использования количественных методов) и возможность использования сотрудников с разным уровнем квалификации на различных этапах являются основной причиной использования этого подхода. Основную сложность составляют высокие требования к квалификации бизнес-аналитика и архитектора, которые должны понимать все особенности функционирования системы. Для сложных систем

[13] Эта последовательность отражает процесс разработки программного обеспечения (Software Development Process, SPD) в Siemens Business Services (С-Петербург).

функции аналитика и архитектора должны выполняться несколькими специалистами, что предъявляет дополнительные требования к организации их совместной работы (например, управление версиями в процессе анализа и проектирования, совместная разработка диаграмм).

3.3. Выбор с использованием метода верификации принципов

Выбранная разработчиком методология, в основе которой лежат эмпирические принципы, определяет используемые технологии, средства разработки. Принцип (от лат. principium основа, начало) – это основное, исходное положение какой-либо теории, учения и т.п.; руководящая идея, основное правило деятельности. Принципы являются концентрированным обобщением разностороннего опыта и подходов на уровне структуры деятельности (framework). Их содержание и смысл: в краткой форме дать практическое руководство для конкретных действий по проекту и/или быть основой для разработки стандартов или алгоритмов работы.

Принципы представляют собой предположения, как правило, неявно выраженные, относительно которых не существует единогласного мнения в индустрии. Наилучший результат даст использование методологии, основанной на принципы, адекватных для компании и проекта. Следовательно, необходимо не только определить, какие *принципы* лежат в основе применяемой методологии, каким образом эти принципы реализованы в компонентах методологии, но и постоянно верифицировать адекватность этих принципов для компании в текущий момент. Непрерывная проверка адекватности принципов определяет развитие методов управления. Многие укоренившиеся принципы, лежащие в основе разработки ПО, меняются благодаря новыми технологиями, заменяются более совершенными методами или более высоким уровнем автоматизации. [162, стр. 3].

Под принципами будем понимать предположения относительно фундаментальных законов разработки ПО. Современные принципы касаются процессов разработки, использования инструментальных средств, организации команды проекта и т.д.

Необходимость явной формулировки принципов содержится в CMMI. Цель GP 2.1 “Установление организационной политики” (Establish an Organizational Goal) предусматривает определение целей организации, ожиданий и руководящих принципов.

Сторонники методологии Agile Software Development сформулировали 12 принципов, которым они обязуются следовать [253]. В этом случае принципы носят характер декларации о том, каким образом будет вестись работа над проектом. По мнению сторонников манифеста именно такие подходы к проектам обеспечивают успех, но авторы не проводили специальных исследований, опираются, в основном, на свой собственный опыт и опыт коллег.

В 1994 Дэвис (Davis) сформулировал 15 принципов программной инженерии в статье «Fifteen principles of software engineering», а затем расширил количество принципов до 201 в книге Принципы программной инженерии [223].

Полный анализ современных принципов, используемых в управлении ПО, представляет значительную сложность, поскольку нет единого источника – свода этих принципов. Они приводятся в книгах и статьях экспертов индустрии. Авторство этих принципов, как правило, не указывается, а сами принципы часто приводятся без доказательств. Часть принципов, лежащих в основе разных методологий, входят в противоречие друг с другом.

В Табл. 8. представлены результаты анализа наиболее существенных на момент проведения исследования принципов разработки ПО. Для каждого принципа приведены наиболее типичные модели и стандарты, в которых он содержится в том или ином виде, определены ограничения области действия принципа, а так же модели и стандарты, опирающиеся на противоположный подход.

Табл. 8. Принципы разработки ПО

Принцип	Реализация	Необходимые условия	Противоположный подход
Приоритет архитектуры (упреждающая разработка архитектуры).	Во всех методологиях	Нет	Нет. Часто на практике имеет место приоритет требований.
Ориентация на выпуск в срок.	MSF	Должна существовать возможность поставки продукта с	TQM, управление качеством. Ориентация на выполнение требований

		ограниченной функциональностью	
Основа успешного выполнения проекта – зрелые процессы разработки.	CMM, SPICE, ISO 9000	Значительная техническая и управленческая сложность проекта	Управление «критическими» проектами
Определить проблему до того, как записывать требования.	Классическая водопадная модель. MSF. (Понимание бизнес-перспективы)	Требования должны быть понятны самому заказчику	XP, SCRUM. Определение требований по мере нахождения решения
Минимизация семантического разрыва между требованиями и реализацией.	Объектно-ориентированное проектирование и разработка	Структура ПО повторяет структуру предметной области – возможно для заказных бизнес-систем, но не для системного ПО или систем класса ERP	Структурное Программирование. Объектная разработка системного ПО, настройка систем класса workflow.
Устанавливать приоритеты требований.	MSF. Ориентация на выпуск в срок.	Требования должны отличаться по приоритетности, не все требования должны быть обязательными	Для систем с высоким уровнем рисков (рисков для жизни и существенных финансовых рисков) целесообразно
Разрабатывать приложение за несколько итераций.	MSF, RUP (разработка бизнес-приложений)	Наличие интегрированной среды, поддерживающей круговую разработку и соответствующей организационной структуры проекта. Соответсвующие возможности, предусмотренные контраком.	Водопадная модель (чаще всего: разработка ПО для оборудования, военных технологий, космоса).
Ранняя демонстрация и поставка продукта клиенту (связанные сценарии использования).	Прототипы (Rapid Application Development	Верно в том случае, если клиент обладает возможностью проверки ПО (например, бизнес-приложения).	Водопадная модель
Промежуточные версии должны поставляться клиенту как можно чаще.	Итеративная разработка (RUP, MSF)		Водопадная модель
Изменения в требования не должны вноситься в проект на поздних этапах разработки.	Водопадная модель	Требования должны быть определены. Требования не должны меняться в ходе разработки	Agile Software Process
Необходимо вовлекать клиента в разработку продукта.	eXtreme Programming, RUP	Клиент должен обладать необходимыми людскими ресурсами	Водопадная модель
Разрабатывать, планируя повторное использование.	ООП, OOP	Достаточно временных и финансовых ресурсов – дополнительные затраты.	Модульное проектирование и разработка
Сразу полностью определять все	Водопадная модель	eXtreme Programming,	Фокусироваться на

требования.		RUP	основных требованиях
Разработчики не должны тестировать собственный код.	Классический подход	Команда разработки должна состоять более чем из двух человека.	XP, PSP

3.4. Выбор моделей и стандартов для создания системы менеджмента качества

Задачи создания системы качества (СМК) находится на уровне управления компанией в целом, лежит за границами отдельных проектов [24 стр. 8-9]. Надо отметить, что рассматриваемые нами инструментальные средства (стандарты качества, модели зрелости и т.д.) предполагают наличие непосредственной связи между качеством ПО и процессами разработки. Эта зависимость обусловлена следующими факторами:

Формализация процессов разработки гарантирует реализацию функций обеспечения качества в проектах (планирование, включая планирование качества, контроль за проектированием и разработкой, анализ результатов тестирования, сохранение результатов);

Выполнение функций обеспечения качества способствует формированию методического обеспечения и информационной базы качества (определение измеряемых показателей, их влияния на характеристики качества);

Систематическое выполнение функций обеспечения качества приводит к повышению эффективность выполнения этих функций за счет освоения специальных средств, методик, накопления информации.

СМК компании-разработчика ПО может создаваться и поддерживаться с использованием одного или нескольких стандартов. Для принятия решения, каким стандартом и (или) моделью зрелости руководствоваться, сравним стандарт ISO 9000, SPICE и CMM. Результаты сравнения представлены в Табл. 9.

Табл. 9. Сравнительный анализ стандартов ISO 9000, SPICE и CMM

Отличительные черты	ISO 9000	SPICE	CMM
Пользуется известностью	Во всех странах и индустриях. В основном, в Европе	США, ИТ компании	США, ИТ компании в сфере разработок ПО для нужд обороны
Основная идея	Проверка	Проверка и улучшения	Непрерывные последовательные улучшения

Основное назначение	Определяет требования к системе качества	Оценить и улучшить процессы разработки	Оценить способность организации к разработке и улучшить процессы
Что оценивается?	Организация	Процесс	Организация
Как оценивается?	Удовлетворяет требованиям (да/нет) не основе анализа документации и интервью сотрудников сторонней организацией.	Уровень процесса на основе рейтингов его практик, полученных для экземпляров процессов.	Уровень зрелости (от 2 до 5) Анализ документации и интервью
Структура	Простая структура. Многоуровневый список требований к реализации функций – 20 ключевых элементов. Абстрактная модель	Сложная структура. Стандарт содержит девять частей различной структуры и назначения. Детальная модель	Основу модели составляет структурированное описание требований к организации, находящейся на определенном уровне 1-5 Детальная модель
Аудит	Внешний независимый	Внутренняя оценка	Внешний независимый

Сертификация по модели ISO 9000 более известна в мире и имеет широкое применение. Соответствующий сертификат будет служить достаточным основанием для того, чтобы начать переговоры о возможном сотрудничестве с европейскими заказчиками. Стандарты ISO 9000 признаны в качестве национальных более чем в 100 странах мира, и в том числе в России. Доступны официальные переводы стандарта и нет проблем с подготовкой специалистов. Внедрив требования модели ISO 9001:2000, организация на 90% выполнит требования к процессам второго уровня зрелости и создаст необходимые предпосылки для внедрения требований третьего и более высокого уровня зрелости процессов по модели SEI SW-CMM.

Цель применения ISO 9000 “получить определенную уверенность в том, что поставляемая продукция соответствует установленным требованиям, если поставщик предъявляет доказательства определенных возможностей в области проектирования, разработки монтажа и обслуживания» - особенно актуальна в для разработки программного обеспечения.

Сравнение CMM и SPICE утратило актуальность после выхода интегрированной модели зрелости CMMI, которая объединила подходы обоих

стандартов. Поэтому для крупных компаний, заинтересованных в получении контрактов в США, оптимальным будет выбор модели CMMI.

3.5. Выбор системы метрик в управлении разработкой ПО

3.5.1. Выбор метрик для стратегического управления

Метрики могут использоваться для оценки проекта, или применяться для оценки процессов с целью оценки возможности процессов и их улучшения.

Система метрик, как система сбалансированных показателей (BSC, Balanced Scorecard) [248], используемая для представления стратегические цели компании в измеримых показателях, должна отражать обязательства компании в области качества (содержащиеся в политике качества). Некоторые метрики (например, метрики качества процессов), могут использоваться в составе показателей BSC (например, среди показателей внутренних бизнес-процессов, Internal Business process).

Соотношение между метриками процессов и проектов определяется масштабом проектов: для крупных проектов, работа над которыми ведется в течение нескольких лет, создаются свои процессы. В этом случае уровень проекта лежит выше уровня процесса, и допускается оценивать процессы отдельного проекта. Если работа над проектом ведется отдельным подразделением в компании, то качество процессов в подразделениях может отличаться.

Затраты на сбор информации достаточно высоки, поэтому любая метрика должна решать определенную проблему управления. Нецелесообразно вводить большое количество метрик. Существует возможность собрать количественные данные после завершения проекта, а не в процессе его реализации, используя документацию проекта, базу данных ошибок и современные средства управления версиями.

Имея дело с уникальными проектами, компании затрудняются в использовании числовых данных, полученных для одного проекта для оценки другого. Кроме того, большую сложность представляет использование метрик других компаний (отраслевых) для оценки собственных проектов, обусловленная, в первую очередь, доступностью данных в структурированном виде. Поэтому на

практике вместо метрик для оценки времени выполнения часто используются экспертные оценки.

Так же следует отметить, что эффективное использование метрик и других способов для оценивания возможно только при отсутствии существенного влияния таких субъективных факторов, как личностные политики и политические игры[14]. Кроме того, необходимо избегать «ложной точности» исходных данных, принимая во внимание фактический уровень точности используемых метрик.

3.5.2. Выбор системы метрик для управления качеством, рисками и улучшения процессов

Метрики дают информацию, необходимую для управления рисками, они позволяют прогнозировать количество ошибок, определять необходимое время тестирования. Построение связанной информационной модели, характеризующей существенные данные о процессах разработки, выполненных проектах, а так же характеристиках ПО, является задачей, которую необходимо решить для перехода к устойчивому (зрелому) процессу разработки.

Критериями выбора метрик являются:

Эффективность метрики — дает ли метрика прямой ответ на существующий вопрос или является косвенной мерой?

С*пецифичность* — в каких случаях данная метрика может быть использована? Зависит ли она от методологии, технологии программирования и проектирования, отдельного проекта?

- *Трудоемкость* — затраты на реализацию метрики в сравнении с выигрышем от ее использования. Зависит от сложности получения и обработки данных.

Размер проекта — будет ли оправдано применение (создание) метрики для небольшого проекта?

В ходе анализа деятельности ИТ-компаний мы определили следующие предпосылки успешного использования количественных методов управления процессами:

---

[14] Умышленное искажение результатов оценивания проекта с использованием сложных моделей и метрик, таких как Functional Point и COCOMO встречается достаточно часто. Это явление получило поэтическое название «туманы и миражи».

Разработка программного обеспечения является стратегическим направление деятельности компании. В этом случае целесообразны инвестиции в систему качества: формализацию процессов, способы сбора метрик и метрическую базу;

Компания осуществляет сходные проекты в течение длительного промежутка времени – формируется достаточная количественная база для планирования проектов;

Менеджмент компании осознает преимущества использования количественных методов управления и рассматривает инвестиции в создание программ метрик, приобретение инструментов и формирование метрической базы как капитальные вложения.

Метрики позволяют прогнозировать значения одних показателей, используя другие показатели, являясь, таким образом, инструментом планирования. Кроме того, метрики могут использоваться для улучшения процессов разработки, повышения качества и снижения рисков.

В Центре обеспечения качества программного обеспечения NASA (Software Assurance Technology Center, SATC) была разработана программа метрик (metrics program) обеспечивающая оценку рисков проекта, качества продукции и эффективности процессов [273]. Программа SATC рекомендует отдельно отслеживать качество требований, качество программного обеспечения и других продуктов (документации), качество тестирования и качество выполнения процессов.

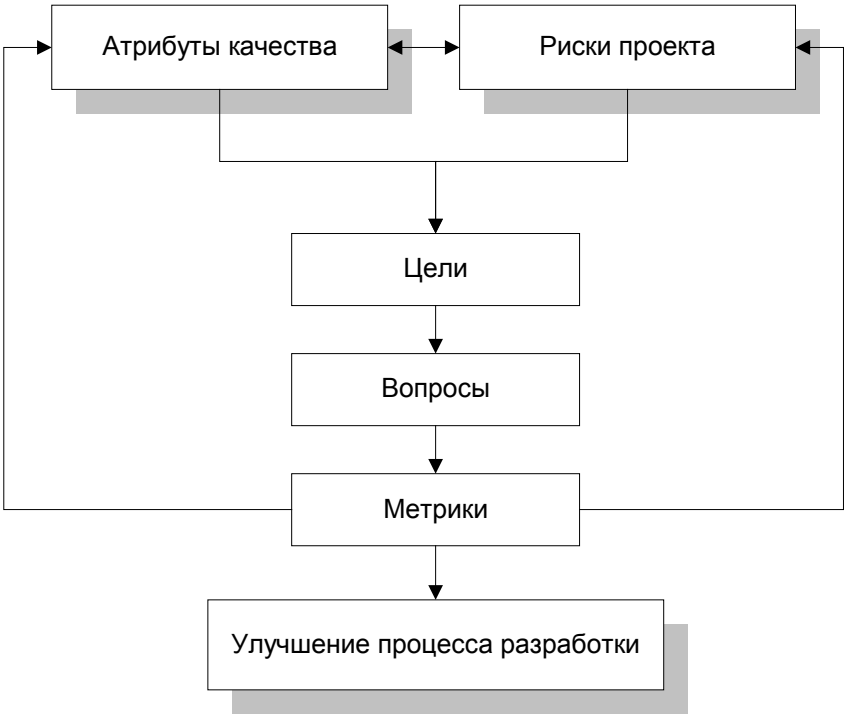


Рис. 3. Модель программы использования метрик SATC [273, стр. 2]

Качество требований включает в себя:

- их полноту, однозначность и непротиворечивость;
- изменчивость (количество и скорость изменений);
- отслеживаемость снизу-вверх и сверху-вниз.

Метрики оценки полноты и однозначности требований используют подсчет фраз, содержащих предположительные конструкции (иногда, может быть, как обычно и т. д.) и фразы типа «будет уточнено позднее», «требуется уточнить» и т. д. Существуют автоматические средства анализа требований в текстовом виде. Использование этих метрик оправдано только в том случае, если существует определенный стиль написания документации. Изменчивость требований оценивается как процентное отношение количества измененных требований к общему количеству требований за определенный период. Отслеживаемость требований предполагает возможность перехода от функциональных требований к проектным решениям. Метрики отслежваемости используют подсчет требований, которые не могут быть отнесены к требованиям более высокого уровня, и количество требований, которые не находят отображения в проектной документации (программном коде).

Программа SATC переносит задачу реализации требований заказчика из сферы разработки ПО, в сферу разработки требований (управления требованиями). Для анализа различных качественных характеристик используются одни и те же метрики. Например, сложность ПО (отдельного модуля, процедуры) (оцениваемая в SATC как цикломатическая сложность) и размер, влияют на возможное число ошибок в коде, время, необходимое на обнаружение и исправление ошибок, и возможность повторного использования. Метрики размера используют количество строк кода (с учетом или без учета комментариев и пустых строк), количество программных операторов.

Управление риском на этапе разработки осуществляется следующим способом. Между объемом и размером существует корреляция, которая, по мнению SATC, может быть аппроксимирована как линейная зависимость. Рекомендуется оценить сложность на основе размера и сравнить со сложность, полученной с помощью метрики сложности. Если полученные результаты будут существенно отличаться, то следует провести дополнительный анализ кода.

Большую практическую ценность представляет прогнозирование ошибок с использованием таблиц контрольных проверок, которые формируются на основе данных причинно-следственного анализа существующей базы данных ошибок. Под ошибкой понимается неправильность, погрешность или неумышленное, невольное искажение объекта или процесса [133 стр. 152]. При этом подразумевается, что известно правильное или неискаженное эталонное состояние объекта, к которому относится ошибка. Неполнота спецификаций приводит к необходимости включать в состав ошибок ситуации, когда нарушаются некоторые правила здравого смысла, формально не предусмотренные эталоном. В случае возникновения такой ситуации целесообразно не только фиксировать ошибку в программе в базе данных ошибок, но и отслеживать неполноту спецификаций (эталонов).

Документированность ПО оценивается с помощью подсчета строк комментариев. Метрика документированности показывает отношение строк комментариев к общему количеству строк кода. Для модулей (процедур) с

высоким значением метрики, характеризующей отношение сложности к размеру, документированность особенно важна.

Целями тестирования являются обнаружение и исправление ошибок, выявление модулей (процедур), которые потенциально содержат большое количество ошибок, обеспечение выпуска в срок программного обеспечения, содержащего допустимое количество ошибок (good enough software). Качество (эффективность) тестирования характеризуется корректностью выпускаемого ПО, количеством ошибок, обнаруживаемых на различных этапах тестирования в модулях/процедурах и полнотой выпускаемого ПО (его соответствием спецификации).

Для оценки качества процессов используется отношение числа дефектов к объему. Для сопоставления качества процессов в проектах, использующих различные средства разработки, объем измеряется в MAELOC (Million assembly-equivalent lines of code – миллионах приведенных строк ассемблера).

Если требованием к процедуре тестирования задан процент ошибок, который должен быть обнаружен на определенном этапе (выпуск бета-версии, выпуск релиза), то мерой риска служит оценка времени завершения тестирования. Если ожидаемое время значительно превышает срок завершения проекта, то можно говорить о высоком риске.

В дополнение к предлагаемым SATC метрикам, на наш взгляд, следует использовать персональные метрики, позволяющие отслеживать качество работы сотрудников (программистов, тестировщиков, разработчиков документации). Отслеживая в течение в двух лет количество ошибок в программных продуктах (кодах, документации), разрабатываемых в компании CSBI, было обнаружено, что оно в незначительной степени зависит от сложности разрабатываемого ПО. Наибольшее влияние на количество ошибок в разработанном коде имеет авторство этого кода. Зная, кому принадлежит данная процедура, можно заранее прогнозировать количество ошибок с высокой вероятностью (по нашим наблюдениям 80-90%). Это наблюдение обуславливает эффективность использования собственного процесса разработки (PSP).

Кроме того, разработчики, вносящие большое количество ошибок, как правило, не могут разрабатывать сложные модули, – существует ограничение сложности. Для управления процессом разработки большое значение имеет не только абсолютное значение этой метрики в данный момент, но и динамика ее изменения, которая показывает улучшение качества работы отдельного сотрудника. Эти метрики следует использовать при периодической аттестации. Нужно отметить, что персональные метрики не должны использоваться для перекладывания ответственности за проблемы проекта Существование структурного и объектного подходов к проектированию и разработке обусловило наличие структурных и объектных метрик. Структурные метрики могут использоваться для измерения объектно-ориентированного ПО, но не наоборот.

3.5.3. Выбор системы метрик в зависимости от языка разработки

При выборе метрик одним из важнейших критериев является интеропрерабельность – независимость от языка разработки. Все объектные метрики являются интероперабельными, но не все объектно-ориентированные языки являются полностью объектными (например, Visual Basic не поддерживает наследование, большинство языков не поддерживают множественное наследование), что сказывается на результатах использования метрик.

Преимуществом объектных метрик является возможность их использования на этапе дизайна (проектирования) системы. Метрики позволяют объективно оценить преимущества и недостатки дизайна (архитектуры системы), выбрать наилучший вариант, кроме того, метрики могут использоваться для прогнозирования (оценивания) трудоемкости разработки, тестирования поддержки. В работе приведен лишь краткий обзор объектных метрик, их действительное количество гораздо больше, например, существует группа метрик, характеризующих категории классов (группы сильно связанных классов).

Надо отметить, что большинство метрик характеризуют одновременно несколько аспектов ПО, например количество строк кода характеризует трудоемкость разработки, а так же является мерой структурированности

программы, если говорить о числе строк кода в отдельном модуле, кроме того, сложность так же можно оценивать с помощью подсчета строк кода.

С другой стороны, характеристики ПО често оцениваются различными способами, например размер ПО (software size) может оцениваться с точки зрения функциональности, исполнения контрактных требований (иногда нефункциональных), размера проекта (потраченных усилий, времени разработки, количества разработчиков), размера системы (включая аппаратные средства и сеть), объема информации (внешних и внутренних потоков данных), и, наконец, физического размера (количество строк кода, размер скомпилированного кода, объем необходимой памяти).

Выбор метрик для оценки трудоемкости разработки ПО.

Прежде чем использовать метод функциональных точек, необходимо определить назначение ПО, разработать спецификации. Метод функциональных точек включает несколько этапов: определение элементов, оценка сложности элементов, получение интегральной оценки ПО в функциональных точках. Для каждого элемента определяется уровень сложности, каждому уровню сложности соответствует определенный вес в функциональных точках.

3.5.4. Выбор модели численной оценки проекта разработки ПО

Процесс принятия решения включает определение проблемы, анализ проблемы, разработка взаимоисключающих решений, поиск оптимального решения, преобразование решения в эффективное решение. Модели чаще всего используется на фазе анализа проблемы и поиске оптимального решения. В то же время модели практически бесполезны для определения проблемы. В лучшем случае модель поможет выявить отклонения в характеристиках продукта, процесса или параметрах проекта.

Использование математических моделей в управлении предполагает предварительное определение конкретных управленческих задач, которые решаются с помощью модели. Особенности использования моделей в управлении процессами разработки программного обеспечения, обусловленные сложностью

получения числовых оценок (метрик), а так же недостаточностью информации и сложность алгоритмов. Можно выделить различное применение моделей:

Систематическое применение модели для решения определенной задачи (например, планирования ресурсов);

Однократное (единовременное) применение модели для решения определенной задачи управления (например, оценки производительности процесса).

- Наиболее общая классификация моделей по назначению приведена ниже:
- Модели оценивания (предсказание, прогнозирование)
- Сроков разработки
- Стоимости (трудоемкости)
- Рисков (количества ошибок, ожидаемых отклонений)
- Модели отслеживания
- Состояния проекта (% выполнения)
- производительности процессов
- Оптимизационные модели
- Распределение ресурсов
- Оптимизация расписания

В [265] приводятся результаты опроса менеджеров софтверных компаний, показывающего, для каких целей используются числовые оценки и математические модели. Было выделено двенадцать различных задач:

- Оценка возможности выполнения проекта (оценка концепции);
- Оценка дизайна, сравнение возможных вариантов реализации;
- Принятие решения об участии в тендере,
- Подготовка проектного предложения;
- Оценка предложения;
- Согласование договора;
- Планирование проекта (составление расписания);
- Отслеживание проекта;
- Кадровое обеспечение проекта;
- Оптимизация использования ресурсов в ходе проекта;
- Прогнозирование (оценка) завершения проекта;
- Перепланирование (изменение проекта)

Существует тенденция роста использования моделей на более поздних этапах разработки, чаще всего модели используются для планирования проекта, обеспечения ресурсами (кадрами) и прогнозирования времени выполнения. Модели используются относительно редко для оценки проекта до заключения контракта, контрактных переговоров или оптимизации использования ресурсов в ходе проекта. На наш взгляд, существующая статистика объясняется разработанностью моделей, обеспечивающих решение данных задач. Нужно отметить, что наиболее часто используемые модели наименее специфичны для разработки программного обеспечения, например, сетевое планирование или распределение ресурсов. Модели, предполагающие вероятностные оценки, использующие приблизительную информацию, используются реже.

Известно, что неудачное завершение многих ИТ проектов было обусловлено неэтичным поведением ИТ специалистов: менеджеров проектов, технических специалистов. Можно выделить следующие элементы неэтичного поведения: искажение информации о ходе проекта (затратах, сроках, технических характеристиках, существующей производительности, количестве ошибок), фальсификация результатов технического или бизнес-анализа (рекомендация незрелых технологий, дорогостоящих избыточных решений). Очевидно, что результатом такого поведения является искажение исходной информации. Проблема искажения информации рассмотрена в п. 1.4.4.1.

Построение эффективного процесса требует возможности количественной оценки его параметров. Математическое моделирование процессов позволяет проанализировать возможные варианты улучшения процесса. Наиболее сложным, и вместе с тем, наиболее ценным с точки зрения практического использования, является имитационное динамическое моделирование. Необходимо учесть, что использование динамического моделирования процесса предполагает формализацию процесса, определение его логики. Не смотря на видимую простоту описания бизнес-процессов, в действительности моделирование логики процесса представляет значительную сложность. Обычно моделируется лишь основной поток, при этом не уделяется внимание альтернативным потокам.

Планирование проекта на предконтрактной стадии включает в себя решение следующих задач:

- Оценка возможности выполнения проекта (оценка концепции);
- Оценка дизайна, сравнение возможных вариантов реализации;
- Принятие решения об участии в тендере,
- Подготовка проектного предложения;
- Согласование договора.

Определение стоимости проекта и оценка времени работы. Эта задача особенно актуальна для проектов с фиксированной суммой контракта (fixed cost contract). От решения этой задачи зависит как получение контракта, поскольку в условиях конкуренции цена должна быть близка к издержкам (best bid), так и прибыльность компании, поскольку сумма контракта должна превышать затраты на разработку. Наиболее известной моделью этого вида является COCOMO II.

Успех проекта во многом зависит от того, насколько эффективно будет решена задача определения наилучшего плана приемки работ и оплаты, особенно в тех случаях, когда финансирование проекта осуществляется из текущих выплат заказчика. Разработаны модели, позволяющие решать эту задачу для строительных компаний, например модель Рассела [276]. Нам не известны модели, адаптированные для разработки программного обеспечения.

Оценка рисков – возможности финансовых убытков в случае превышения бюджета и возможность нарушения сроков.

В ходе реализации проекта решаются следующие задачи:

- Оптимизация использования ресурсов. Эта задача является наиболее сложной для много проектных задач. Кроме того, для работ, составляющих процесс разработки программного обеспечения, невозможно использовать одну функцию зависимости между длительностью работы и количеством ресурсов. Необходимо выделять дискретные значения;
- Определение статуса проекта, например, сетевая интерактивная модель контроля за ходом выполнения проекта Голенко – Гранберга [234];
- Определение состава работ (WBS, work breakdown structure), связей между ними;

- Анализ выполненного проекта является обязательным этапом. Практика рецензирования завершенных проектов позволяет повысить качество управления этими проектами. Следует отметить, что незначительное число компаний эффективно рецензируют выполненные проекты [171, стр. 500]. В качестве основных причин указываются изменение целей проекта, занятость сотрудников в других проектах.

3.5.5. Выбор модели численных оценок в управлении процессами

Построение эффективного процесса требует наличия возможности получения количественных оценок процесса. Исходным условием, делающим возможным оценку процесса, является его повторяемость (если речь не идет об оценке степени повторяемости / зрелости процесса). Можно обозначить два подхода к получению количественных оценок:

- Сбор (измерение) характеристик экземпляров процессов;
- Получение оценок с помощью моделей.

Очевидно, что первый способ применим только для реально существующих (организованных) процессов, в то время как использование моделирования позволяет проанализировать различные варианты построения процесса, распределения ресурсов без реального воплощения, исключить неэффективные решения. Классификация моделей, используемых в управлении, приведена в п. 0 В данном разделе мы рассматриваем особенности сущности процесса и возможность использования различных видов моделирования: описательного и математического, аналитического и имитационного.

Необходимым условиями для построения адекватной модели является понимание сущности моделируемого явления, а так же точная формулировка задачи моделирования. Анализируя подходы к моделированию процессов разработки ПО необходимо учитывать следующие особенности процессов:

Процесс характеризуется различной степенью повторяемости. Могут иметь место различные вариации. В крайнем случае, может отсутствовать определенная последовательность действий, некоторые из условий формализации процесса (условия входа, выхода, логика выполнения функции, владелец процесса и т.д.)

могут быть неопределенны. Следует учитывать это обстоятельство, как при построении моделей, так и при использовании их результатов.

Процесс представляет собой сложную динамическую структуру. Одновременно в компании протекают множество процессов имеющих различный характер и длительность цикла;

Временные и качественные характеристики выполнения отдельных функций нелинейно зависят от других параметров (выделенных ресурсов, графика работы, распределения задач). Например, установление более длительного рабочего дня в течение ограниченного промежутка времени (от одной недели до двух-трех месяцев) может привести к реальному ускорению проекта, и даже повышению качества продукта, обусловленному большей концентрированностью разработчиков над задачами проекта. Но в длительном периоде такая практика может дать обратный эффект, из-за утомления участников проекта. Другим примером является использование одного сотрудника в нескольких проектах. Можно предположить, что при работе над одним проектом достигается наилучшая производительность. Действительно, в этом отсутствуют потери времени на переключение между задачами. Но если сотрудник занимается одним проектом достаточно долго, то на определенном этапе происходит потеря интереса. В этом случае целесообразно переключить сотрудника на другой проект, в результате чего может быть достигнуто существенное увеличение производительности как при работе над новым проектом, так и при возвращении к старому проекту. На самом деле влияние психологических факторов столь сложно, что следует моделировать влияние каждого фактора в отдельности, с широким использованием качественной информации об их природе и закономерностях.

Классификация процессов может использоваться при разработке моделей. При построении имитационных моделей следует учитывать, что все процессы можно разделить на две категории в зависимости от характера инициация процесса:

Периодические процессы – начало и/или окончание процесса совпадают с определенной календарной датой (например, ежегодное планирование или еженедельное антивирусное обслуживание);

Процессы, инициируемые событием – начало процесса инициируется внешним событием или внутренним событием (например, подписание контракта на разработку инициирует процесс проектирования)

Большинство бизнес-процессов имеют циклический характер, например ресурсы на входе преобразуются в готовую продукцию в течение определенного промежутка времени. Одной из важнейших характеристик процесса является продолжительность его цикла – время между началом и окончанием.

По количеству одновременно выполняемых экземпляров процесса можно выделить следующие виды процессов:[15]

- Дискретные процессы – одновременно выполняется один экземпляр процесса, возможна ситуация, когда данный процесс не выполняется;
- Непрерывные процессы – одновременно выполняется один экземпляр процесса. Процесс выполняется постоянно;
- Параллельные процессы – выполняется несколько экземпляров процессов.

Процессы различаются также по внутренней структуре. В одних процессах функции выполняются в прямой последовательности, другие содержат параллельные участки, циклы, условные ветвления. Структура процесса разработки ПО зависит от выбранной модели жизненного цикла, организационный структуры компании, масштаба проектов организации.

В зависимости от назначения можно выделить следующие направления моделирования:

- *Моделирование логики процесса.* Наилучшим образом эту задачу решают структурные и объектные модели: IDEF0, EPC, Activity Diagram;
- *Моделирование поведения процесса*, различных закономерностей. Для этих целей можно использовать аналитические математические модели. Следует учесть, что во многих случаях эти модели служат лишь для демонстрации характера зависимости, но не могут использоваться для получения реальных числовых оценок в условиях практики из-за недоступности исходной информации;

[15] Предложенная терминология отличается от общепринятой, используемой для описания производственных процессов.

- *Имитационное моделирование* представляет собой наиболее мощное средство получения числовых характеристик процесса. Для построения имитационной модели необходимо понимание логики процесса и поведения процесса, следовательно, имитационная модель включает в себя структурную модель процесса и совокупность аналитических моделей, связанных между собой. Математической основой этих моделей являются нелинейные дифференциальные уравнения. Динамические модели отражают нелинейные и нестационарные зависимости между характеристиками проекта;
- *Имитационное динамическое моделирование* является одним из способов оценки времени разработки приложения. Разработано множество имитационных моделей, которые позволяют исследовать нелинейные зависимости между различными факторами, влияющими на динамику проектных процессов. Например, с помощью динамического моделирования можно оценить эффект от сверхурочной работы в течение определенного промежутка времени, или возможность ускорения проекта [240].

Ф. Брукс отмечал, что зависимость между количеством разработчиков и временем разработки носит нелинейный характер [25]. Очевидно, что все зависимости между ключевыми переменными, характеризующими выполнение проекта, и, следовательно, производительность процессов, являются нелинейными, и, кроме того, изменяются со временем. Вследствие эффекта обратной связи, возникающего в результате принятия решений, изменение одной переменной может повлиять не только на другие переменные, но со временем может так же воздействовать на собственное значение.

3.5.6. Учет влияния использования численных методов оценки на мотивацию сотрудников

Использование метрик не только позволяет получить численные оценки характеристик продукта, процесса и проекта, но и в определенной степени оказывает влияние на эти характеристики. Использование метрик оказывает влияние на мотивацию сотрудников, а, следовательно, и на результаты труда.

Следующие рассуждения описывают результаты собственных наблюдений, а так же мнения менеджеров и программистов, полученных в результате неформальных опросов.

В том случае, если метрики не используются, у сотрудников (в первую очередь программистов) может возникнуть ошибочное мнение, что менеджмент не интересуется результатами работы, или не обладает реальными возможностями учета результатов работы. Следовательно, хорошая работа не будет замечена и вознаграждена.

Можно сделать вывод, что использование метрик оправдано, если они оказывают положительное влияние на мотивацию. С другой стороны, использование метрик может быть воспринято как демонстрация отсутствия доверия к сотрудникам. В этом случае их использование отрицательно скажется на мотивации, и результаты будут тем хуже, чем менее эффективная метрика используется. Например, в случае использования метрики SLOC, существует возможность написания неиспользуемых частей кода, которая, тем не менее, будет учитываться данной метрикой. Таким образом, использование метрик не только влияет на производительность, но влияет на качество разрабатываемого ПО, и в предельном случае, может оказать влияние на архитектуру. Боэм приводит примеры влияния метрик (установленных критериев успеха) на характеристики качества, каждая характеристика достигала своего наивысшего значения того, когда она являлась критерием успеха группы, разрабатывающей программу [24, стр. 14].

Можно говорить об «открытом» и «скрытом» использовании метрик. В первом случае сотрудники знают о сборе данных, представляют себе механизм и алгоритм получения данных, их обработки, а так же цели использования полученных данных. Во втором случае сотрудники не имеют представления о том, что осуществляется сбор данных, оцениваются определенные характеристики. Рекомендуется не только собирать данные открыто, но и предоставлять собранные данные всем сотрудникам организации, что бы они могли участвовать в улучшении процессов. С другой стороны необходимо учитывать требование

конфиденциальности, например, подчиненным не должны быть доступны данные, характеризующие работу руководителей.

Это требование содержится в более жестком виде в CMM (Level 4. Quantitative Process Management, Commitments1): использование метрик для оценки работы отдельного сотрудника уменьшит достоверность и ухудшит полезность предоставляемых данных, поэтому данные о производительности отдельных сотрудников должны быть защищены, а доступ к этим данным должен строго контролироваться. [266, стр. L4-3]. В управлении используются данные о производительности процесса, именно эти характеристики контролируются и улучшаются.

Влияние на мотивацию оказывает не только сам факт сбора количественной информации. Положительное влияние так же оказывает повышение уверенности сотрудников в результатах, которые они получают, за счет использования эффективных процессов (прогнозирование рисков, тестирование). Метрики позволяют более точно прогнозировать трудозатраты, использование метрик улучшает реалистичность планов.

Реинжиниринг процессов представляет собой изменение, а любые изменения в организации сталкиваются со значительным сопротивлением. Большое количество работ посвящено проблеме преодоления сопротивления изменениям (change management). Известно, что одним из способов преодоления сопротивления является демонстрация положительных результатов от изменений как можно раньше. Метрики позволяют повысить мотивацию к улучшению процессов, демонстрируя результаты с помощью числовых характеристик [SPICE Part 7, 6.3 Process improvement goals and motivation].

Метрики не должны усиливать конкуренцию внутри рабочих групп, следовательно, необходимы метрики, которые оценивали бы работу группы, а не отдельных сотрудников. Таким образом, существует компромисс между необходимостью оценки результатов отдельного сотрудника и команды. Возможным выходом может быть внутригрупповая оценка отдельных сотрудников, и оценка команд на уровне компании.

Метрики необходимы для обеспечения управления на основе поставленных целей и самоконтроля. Что бы сотрудники могли контролировать эффективность и результаты собственной деятельности, они должны знать не только свои цели, но и должны так же уметь оценить (измерить) свою собственную эффективность и результаты своей деятельности, взяв за основу стоящие перед ними цели.

3.6. Информационное и техническое обеспечение выбора

3.6.1. Формирование библиотеки инструментальных средств

Выбор и адаптация инструментальных средств предполагают создание библиотеки инструментальных средств. Библиотека является инструментом управления знаниями в компании.

Для формирования библиотеки инструментальных средств необходимо:

- разработать процедуры обновления;
- разработать классификацию материалов (на основе типовой классификации);
- определить оптимальный объем включаемых материалов;
- обеспечить техническую поддержку библиотеки, включая надежное хранение, обеспечение доступа, поиска, хранение версий;

Мы рекомендует следующий подход к индексации элементов методологии. Для каждого элемента хранения в библиотеке инструментальных средств (модели, стандарта, техники и т.д.) необходимо определить:

- Категорию. Необходимо отнести элемент к одной из выделенных категорий, например стандарт, метрика, концепция и др.;
- Назначение.

Основные задачи, которые позволяет решить использование данного элемента. Такими задачами может быть:

- Уменьшение риска;
- Снижение количества дефектов;
- Сокращение сроков разработки;
- Обеспечение эффективного использования ресурсов;
- Область действия;
- Процесс (используя принятую модель процессов);
- Роли (используя модель команды проекта и организационную структуру);

- Стадии (этапы ЖЦ ПО) используя модели жизненного цикла;
- Технологию разработки (ООП, Web, etc);
- Объем (количество разработчиков и сроки проекта, при котором оправдано использование данного элемента).

Связи с другими элементами. Связи могут быть следующих видов.

- Используется в другом элементе;
- Использует другие элементы;
- Используется одновременно с другими элементами;
- Связь с процедурами сертификации;
- Используется для сертификации по ISO 9000;
- Используется для сертификации на соответствие заданному уровню CMM.

3.6.2. Обеспечение повторного использования

Преимущество стандартизации процессов в отдельной компании состоит не в установлении жестких правил, а *в возможности повторного использования процессов* в проектах при сходстве условий. Известно, что повторное использование обеспечивает снижение затрат. Соотношение 20/80, характеризующее долю времени, затрачиваемую на кодирование, говорит о целесообразности повторного использования не только кода, но и документации. Кроме того, значительный эффект может быть достигнут от повторного использования сложившихся коллективов (команд) разработчиков. Производительность труда, качество продукта не являются постоянной величиной для коллектива разработчиков и отдельного разработчика, то есть производительность отдельного сотрудника может измениться, если он перейдет в другую команду.

Только повторное использование процессов (повторяемость процесса) позволяет улучшать качество разрабатываемого ПО. Нужно отметить, что при отсутствии повторяемости, качество программного обеспечения может быть достаточно высоким, но менеджмент не имеет возможности влиять на качество.

3.6.3. Адаптация инструментальных средств

Современные методологии управления разработкой ПО и внедрением информационных систем, а также стандарты и модели зрелости допускают

возможность адаптации. Использование тех или иных инструментальных средств не дает гарантии рыночного успеха и даже не обеспечивает достижения заданных значений характеристик процессов разработки. Инструментальные средства, доступные всем участникам рынка не могут обеспечивать рыночного преимущества если их использовать без необходимой адаптации.

Рекомендации по адаптации инструментальных средств часто являются составной частью методологий и стандартов. Например, в CMMI указывается, необходимость адаптации модели с учетом целей бизнеса и особенностей организации. Определены практики адаптации модели CMMI для проектов – так называемые “Tailoring guidelines”. Согласно CMMI адаптация включает в себя выбор стандартной модели процессов и модели жизненного цикла и их модификацию. Допускается использование стандартных (типовых) моделей.

Адаптация инструментальных средств предполагает наличие помимо обязательной части, рекомендательных частей, которые и являются объектами адаптации.

3.6.4. Структура документации и структура стандартов компании-разработчика

Концепции, модели, стандарты, инструменты находят практическое применение в работе компаний-разработчиков программного обеспечения. Документация является ядром системы менеджмента качества, служит для использования методов, моделей и стандартов в практике ежедневной работы компании. Рис. 4. представлена структура документации компании-разработчика программного обеспечения. Данная структура была разработана в рамках консалтингового проекта для компании CSBI EE (Computer Systems for Business International). Структура определяет место стандартов в составе документации, используемой компанией.

Рис. 4. Структура документации компании – разработчика ПО

Классификация внутрифирменных стандартов, выпаленная в рамках того же проекта, на основе объекта стандартизации представлена на Рис. 5.

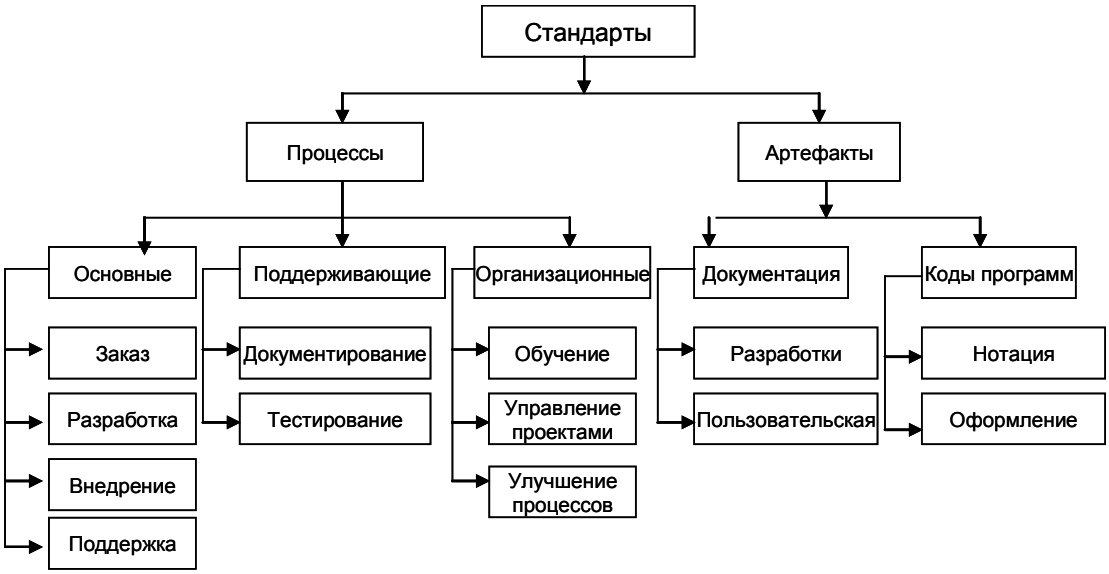


Рис. 5. Классификация внутрифирменных стандартов

3.6.5. Методика формализации процессов разработки

Формализация процесса является необходимым условием для перехода к процессному управлению. Формализация включает в себя выделение процессов, а так же структуризацию самих процессов, определение логики процессов.

Формализация (структурирование) процесса включает в себя определение:

- целей процесса;
- владельца процесса;
- состава и последовательности работ;
- условий входа и выхода (как для граничных работа, так и для работ внутри процесса);
- входов и выходов (используемых материалов, производимой продукции, документации);
- участников процесса (ответственность, роли и деятельности для каждого участника в разрезе работ).

Наличие формального описания процесса, а так же его фиксация во внутрифирменном стандарте, имеет значение даже в том случае, если процесс уже является повторяемым. Как правило, если функцию выполняет один и тот же сотрудник, то он автоматически вырабатывает для себя определенные правила, последовательность выполнения действий. Следовательно, если несколько функций, образующих процесс, выполняются тем же составом, то процесс становиться повторяющимся, что может создать иллюзию стабильной работы. Но в этом случае изменение кадрового состава, или изменение условий работы, вероятно, приведет к незапланированному изменению процесса.

Нужно отметить, что большинство сотрудников обычно знают, над каким проектом они работают. В то же время сотрудники иногда не знают, какому процессу принадлежит выполняемая функция.

Менеджеры компании стоят перед выбором между скорейшим завершением текущих проектов и улучшением процессов разработки, в результате чего лучше будут выполняться проекты в будущем. Такой выбор обусловлен необходимостью отвлечения ресурсов компании на деятельность по улучшению процессов. Особенность этой задачи требует отвлечения наиболее дефицитных ресурсов – квалифицированных сотрудников. Совершенствование процессов разработки представляет собой инвестиции, которые не всегда находят адекватное отражение в финансовой отчетности.

Возможным способом отражения (учета) инвестиций в совершенствование процессов является сертификация компании по одному или нескольким стандартам качества (ISO 9000, CMM).

Даже наличие принятых стандартов процессов гарантируют повторяемости процесса., существует проблема «фиксации» процесса при разработке ПО. Очевидно, что участие большого количества специалистов, выполняющих различные функции процесса, использование специализированных программных средств косвенным образом способствует «фиксации» процесса. Обеспечение повторяемости процессов является одной из основных обязанностей Отдела по обеспечению качества (Quality Assurance).

Степень формализованности (структурированности, повторяемости) процесса разработки программного обеспечения зависит от целей его создания, конечного назначения и численности группы разработчиков [24, стр. 49]. В общем случае, чем сложнее и больше разрабатываемая система, тем более подробно необходимо планировать проект, тем сложнее координировать и контролировать работу. Проблема формализации процесса разработки актуальна для многих компаний-производителей ПО. Поэтому представляется целесообразным разработать соответствующую методику, которая бы позволила формализовать процессы разработки программного обеспечения, непрерывно и целенаправленно улучшать их.

Существуют методики формализации и улучшения процессов, например, достаточно полная методика представлена в [229]. В основе такой методики должны лежать универсальные концепции управления. С другой стороны, существует потребность в доступном руководстве, адаптированном для разработки ПО, которое бы давало рекомендации по переходу к более зрелым (повторяемым) процессам, с учетом специфики этого вида деятельности.

Если отсутствует какое-либо формализованное описание процессов, а так же не используется какой-либо повторяющийся процесс, то использование следующей методики позволит получить описание процессов, которым должна следовать компания.

При существенном разнообразии выполняемых проектов компания не может использовать один стандартный набор процессов. Следовательно, необходимо наличие библиотеки процессов. На первый взгляд кажется, что следует структурировать и улучшать процессы разработки самого важного и крупного

проекта в компании. Не сложно предложить модель процессов и формализовать процессы крупного проекта, но достаточно сложно внедрить эту модель. Поэтому, что бы уменьшить риски, следует начать с небольшого нового проекта.

Существует и другое мнение, состоящее в том, что любые изменения должны быть радикальными, всеобщими и проводиться быстро, что бы преодолеть сопротивление [188]. Этот подход может быть справедлив для сервисных организаций, но в разработке ПО изменение процесса требует более длительного обучения, подготовки инфраструктуры, апробации методов.

До начала проекта формализации необходимо определить язык моделирования, а так же инструментальные средства, которые будут использоваться (ARIS, Visio, Rational Rose, BPWin, etc). Выбор языка моделирования и инструментов представляет собой отдельную задачу. Следует учитывать квалификацию персонала, существующий опыт в использовании моделирования при проектировании и разработки ПО, поскольку здесь могут быть использованы те же средства.

Для создания референтных моделей в работе использовался ARIS 5.0. Формализация процессов разработки включает следующую последовательность шагов:

- *На нулевом* шаге уточняется стратегия бизнеса, маркетинговая стратегия, организационная структура компании, анализируется культура компании (модель культуры компании). Уточняются инвестиционные возможности компании. Если информация уже доступна, то она документируется и распространяется между участниками проекта формализации, в противном случае потребуется проведение дополнительных мероприятий для уточнения необходимой информации (Примечание. Этот шаг соответствует модели SPICE ORG.1). В результате уточняются требования к качеству разрабатываемого ПО, определяется технология, которая будет использоваться.
- *На первом* шаге делается выбор базовой (референтной) модели процессов – в зависимости от вида разрабатываемого ПО (информационные системы для бизнеса, системы реального времени, системы навигации и т. д.). Выбор модели,

адаптированной для конкретного вида ПО (информационных проектов), позволяет существенно упростить дальнейшую формализацию.

- *На третьем* шаге осуществляется сравнение референтной модели с полученными моделями функций, артефактов и ролей организации. Уточняются названия процессов, функций, ролей, фаз и этапов, видов документов. В результате получается идеальная модель процессов организации. Полученная модель повторно анализируется с точки зрения достижения целей процесса. Возможно сравнение модели с различными стандартами (ISO 9000, SPICE, CMMI).
- *На четвертом* шаге происходит уточнение модели. Формируются различные представления, позволяющие отразить различные аспекты использования. Уточняются условия входа и выхода, Формализованное описание процессов включает в себя: список процессов с указанием владельцев процессов, модель каждого процесса, отражающую его логику (например eEPC или IDEF0/IDEF3), описание каждой функции (включающее описание используемых методов, инструментов).
- *На пятом* шаге полученная модель повторно анализируется с точки зрения эффективности управления проектами.
- *На шестом* шаге разрабатывается программа внедрения формализованных процессов, включающая в себя программу обучения, план закупки программного и технического обеспечения, разрабатывается процедура контроля за использованием формализованных процессов.
- *На седьмом* шаге осуществляется внедрение формальных процессов. Учитывая, что только часть процессов привязаны непосредственно к проектам (основные и вспомогательные процессы жизненного цикла), а другая часть существует независимо (организационные процессы), внедрение формальных процессов распадается на две задачи: использование формализованных процессов в конкретном проекте и использование формализованных процессов в управлении. Рекомендуется использовать формальные процессы в новом небольшом проекте, чтобы минимизировать риски организации.
- *На заключительном* шаге осуществляется сравнение формального описания с выполняемыми процессами. Исправляется формальное описание и

предпринимаются шаги по приведению реальных процессов в соответствие с описанием.

3.6.6. Методика оптимизации процессов разработки ПО

Для управления процессами необходимо планировать улучшение процессов в соответствие с бизнес-целями, оценивать сами процессы и эффективность мероприятий по их улучшению.

Выделяются три уровня усилий по улучшению процессов:

- Непрерывное улучшение процессов, направленное на уменьшение разброса показателей качества, непрерывное улучшение работ и процессов;
- Перепроектирования бизнес-процессов, как правило, состоящее в выявлении и удалении бесполезных работ, в результате чего уменьшается стоимость процесса и сокращается длительность цикла;
- Реинжиниринг бизнес-процессов, который позволяет значительно увеличивать эффективность процессов, производительность и качество за счет использования новых технологий.

Подобное разделение имеет значение, поскольку для каждого уровня используются разные методы, ресурсы. Непрерывное улучшение может осуществляться отделом качества, на основании предложений, вносимым самими сотрудниками. Реинжиниринг требует поддержки высшего менеджмента, его результат зависит как от используемых технологий реинжиниринга, так и от лидерства.

Следует отметить, что в сервисных организация, для которых разрабатывалась эта классификация, не существует «библиотек» процессов, то есть существует только одна действующая версия каждого процесса. Соответственно, задача улучшения в этом случае сводиться к изменению только одной связанной модели процессов. В компаниях-разработчиках ПО может существовать «библиотека» процессов – набор связанных моделей процессов жизненного цикла, которые применяются в зависимости от используемой технологии, масштаба проекта, его специфики (информационный портал, система

управления предприятием, системное ПО, т.д.), степени определенности требований. Учитывая, что существуют три группы процессов: процессы жизненного цикла, вспомогательные процессы и организационные процессы, следует различать действия по улучшению процессов для каждой группы.

Улучшению процесса должно предшествовать определение целей, которая должна быть достигнута в результате изменения. Чаще всего встречаются следующие цели улучшения процессов:

- улучшение производительности процесса (использование лучших практик разработки);
- повышение эффективности процессов – соответствия процессов целям организации;
- повышение надежности процесса (снижение различных рисков, а так же построение повторяемого процесса);
- снижение затрат за счет оптимизации использования ресурсов;
- повышение точности планирования за счет использования метрик (например, предсказание количества дефектов в системе и определение стоимости использования системы с таким количеством дефектов (определение времени отладки / тестирования).

В большинстве компаний существует значительное сопротивление изменениям [168]. Формализация процессов, их стандартизация (повышение уровня зрелости) также являются изменением. Необходимо учитывать объективно существующие противодействие изменениям, используя средства управления изменениями, максимально вовлекая сотрудников в проект по формализации, информируя их о целях, которые необходимо достигнуть и средствах, которые будут использованы.

Можно выделить следующие возможности улучшения процесса:

- сокращение длительности процесса (снижение стоимости) за счет уменьшение числа организационных уровней и функциональных подразделений, участвующих в процессе. Что бы наглядно представить процесс в организационной структуре можно использовать Диаграммы потоков событий (process chain diagram);
- уменьшение рисков процесса за счет добавления дополнительных функций проверок;

- улучшение производительности сотрудников за счет улучшения коммуникации, устранения конфликтов. При этом улучшения могут наступить не в том процессе, который был изменен, а в том, который использует результаты этого процесса.

Таким образом, можно стремиться к улучшению результатов процесса или получению заданных характеристик процесса, которые должны влиять на результат или имеют самостоятельную ценность для потребителя. Мероприятия по улучшению процессов осуществляются менеджментом компании (в рамках программы реинжиниринга) и (или) собственниками процессов в рамках непрерывного улучшения процессов.

Предлагаемая методика предназначена для непрерывного улучшения. Она построена на основе предпосылки, что невозможно построить эффективный процесс эмпирическим путем, поскольку невозможно учесть все факторы, влияющие на характеристики процесса.

- *На начальном* шаге необходимо формализовать процессы разработки. В результате должно быть получено описание процессов, включающее логику процесса, условия входа и выхода, владельца процесса. Описание методики формализации процесса приведено в п. Глава 3.
- *На первом* шаге формальная модель сравнивается с существующим положением дел в компании. Выполняются ли в действительности процессы в соответствие с разработанными требованиями? Определяются все несоответствия. Улучшение процессов включает в себя улучшение идеальной модели (стандарта) и улучшение реальных процессов (установление соответствия реального процесса стандарту).
- *На втором* шаге устанавливаются цели улучшения процессов. Такими целями может быть достижение заданного уровня CMMI, достижение определенных экономических результатов деятельности, качественных характеристик процессов, проектов, продукта. В зависимости от целей выбирается способ улучшения: изменение структуры одного или нескольких процессов, создание новых процессов, функций, улучшение структуры (содержания) документов, использование новых инструментов, технологий.

- *На третьем* шаге выбирается способ проектирования изменений (улучшений). При построении референтных моделей использовался способ, который можно эффективно использовать для улучшения процессов, а именно – сравнение существующих моделей с выбранной моделью (стандартом), например, проверка выполнения требований CMMI. Следует так же использовать предложения, сделанные сотрудниками ранее, для этого необходимо накапливать такие предложения. Можно воспользоваться готовой методикой улучшения процессов, например SPICE.
- *На четвертом* шаге определяется штат сотрудников, которых затрагивают предлагаемые изменения. Цели и задачи изменений доводятся до сведения этих сотрудников. Обсуждаются возможные варианты. Уточняются детали предлагаемых изменений и определяются цели, которые должны быть достигнуты.
- *На пятом* шаге согласованные изменения вносятся в модели процессов. Планируются действия по реализации изменений. Пересматриваются связанные с изменениями стандарты, инструкции, приобретаются необходимые программные и технические средства. Проводится необходимое обучение.
- *На шестом* шаге проекты выполняются с использованием новых процессов. Происходит сбор данных о характеристиках измененных процессов.
- *На седьмом* шаге полученные результаты сравниваются с ожидаемыми результатами изменения процесса. Если результаты достигнуты, то стандарты фиксируются, в противном случае можно вернуться к исходному варианту или перейти к шагу четыре.

3.7. Референтные модели компании-разработчика программного обеспечения

3.7.1. Формализация процессов разработки ПО

Для компаний-разработчиков ПО проблема формализации бизнес-процессов (процессов разработки) отличается от той проблемы, с которой сталкиваются производственные компании, поскольку повторяемость процессов в производственных компаниях, как правило, наблюдается и в том случае, когда отсутствует их формальное описание. В разработке ПО *отсутствие формального*

описание в абсолютном большинстве случаев свидетельствует об отсутствие их повторяемости (незрелости процессов, в терминологии СММ).

В связи с этим существует необходимость в использовании стандартных подходов, которые бы позволили бы уменьшить затраты на создание эффективных процессов разработки ПО.

Но технология разработки программного обеспечения все еще находится в состоянии быстрого эволюционного развития, что в некоторых случаях затрудняет использование каких-либо устойчивых моделей. Использование референтных моделей процессов, разработанных с учетом типа ПО и используемых технологий позволяет ускорить переход к повторяемым процессам.

Следует учитывать риск чрезмерной формализации, жесткий процесс может стать препятствием для инноваций, ослабить конкурентоспособность компании в условиях быстрых изменений. Поэтому необходимо постоянно сверять существующий процесс на его соответствие целям организации, а так же предусмотреть возможность изменения процесса на основе предложений сотрудников – создать механизм приспособления к изменяющимся условиям.

Однозначное выделение (определение) процессов в функционирующей организации представляет техническую сложность, обусловленную наличием связей между процессами, отсутствием регулярной повторяемости процесса, существованием «висящих» функций – функций, которые не принадлежат ни одному процессу или функций, принадлежащих нескольким процессам одновременно.

Формальное описание необходимо для улучшения процессов (внедрения информационных систем, разработки формальных инструкций, перепроектирования процессов). Формализация является условием установления повторяемости процесса, то есть формализация всегда на начальном этапе предполагает не описание существующего, а описание «идеального» процесса. Именно это соображение делает особенно ценными референтные модели процессов разработки, которые облегчают разработку процессов для данной компании. В п. 3.7. приводятся разработанные нами референтные модели

организации-разработчика заказного ПО, выполненные с использованием ARIS 5.0 и нотации eEPC (Event Driven Process Chain).

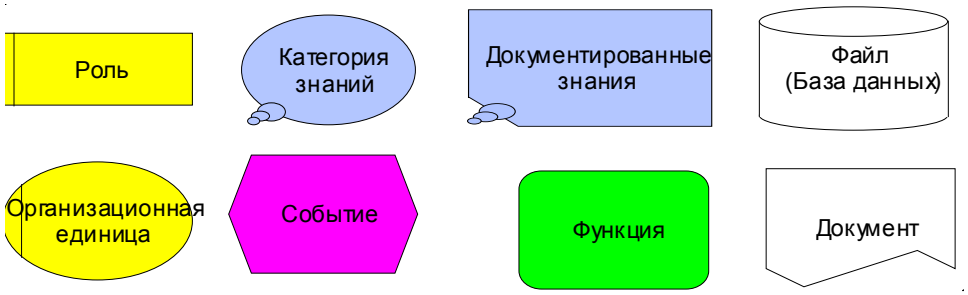


Рис. 6. Нотация eEPC (использованные элементы)

3.7.2. Модель структуры знаний и карта знаний

Модель стриктуры знаний компании является наиболее общей, в сравнении с моделями процессов, моделью организации. На ней могут быть представлены как неструктурированные и недокументированные знания (как правило, наиболее общие), так и знания, содержащиеся в документах, базах данных и хранилищах данных. Модель структуры знаний позволяет выявить знания, существенные для работы компании, определить, где содержаться эти знания. Не смотря на то, что модели знаний на прямую не связаны с процессом, их использование при формализации и улучшении процесса оправдано, поскольку в процессах формируются и используются знания, содержащиеся в документах, база данных, и недокументированные знания.

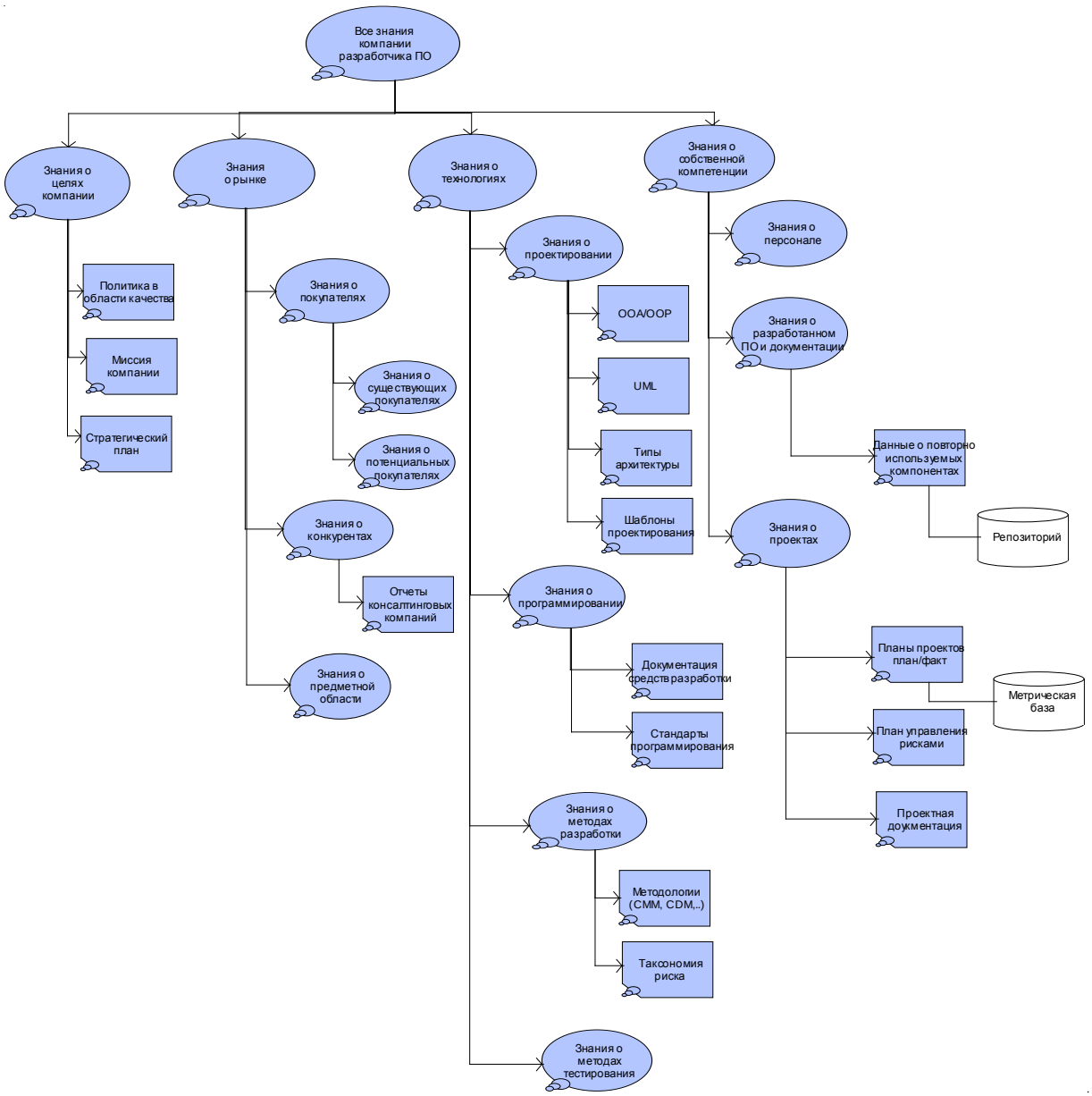


Рис. 7. Модель структуры знаний компании

Процесс обучения должен быть направлен на расширение знаний, а одна из задач процесса создания инфраструктуры состоит в обеспечение доступа к знаниям в любой форме. Таким образом, модели знаний позволяют выделить некоторые артефакты (документы), которые будут использоваться при проектировании процессов, а так же определяют требования к процессам.

Карта знаний показывает роли, которые играют участники проекта, и знания, которыми они обладают (или которые им необходимы для выполнения работы). Следует отметить, что построение моделей является итеративным процессом, и в

ходе моделирования процессов разработки уточняется и дополняется структура знаний и карта знаний.

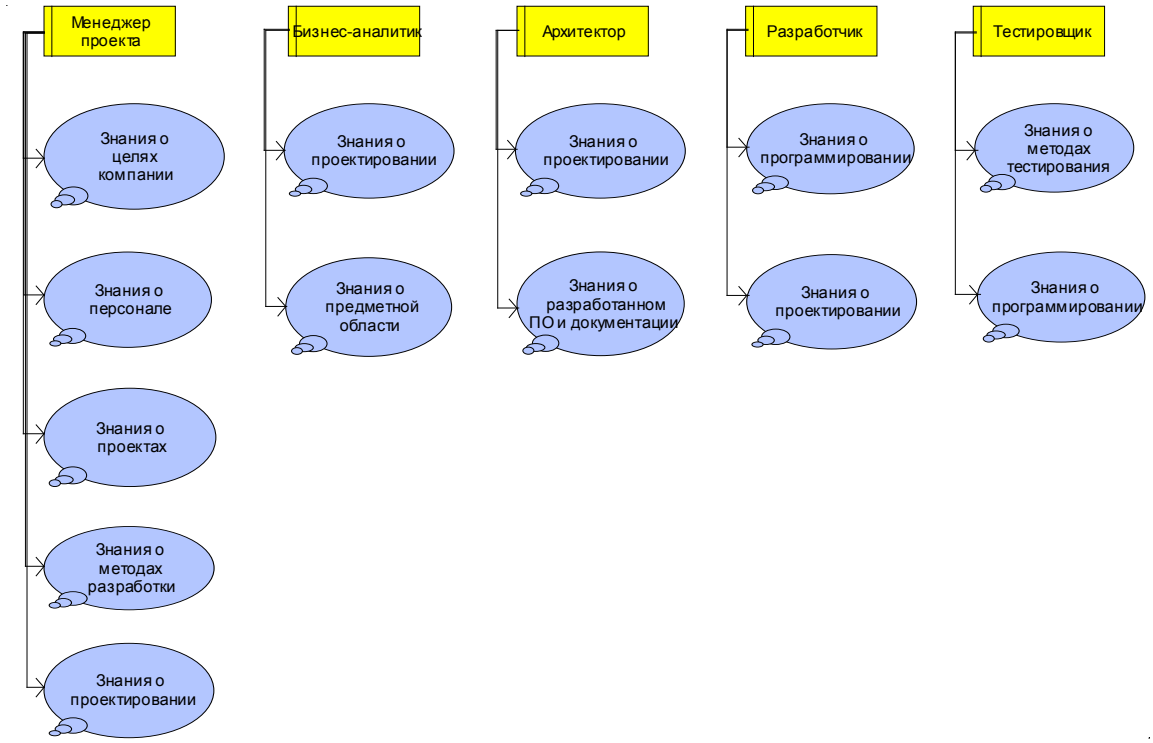


Рис. 8 Карта знаний

Примечание. Для обозначения роли используется символ должность «position», поскольку обозначение для роли в нотации eEPC методологии ARIS отсутствует. Альтернативой было использования символа Actor нотации UML, поддерживаемой ARIS.

3.7.3. Модель организационной структуры

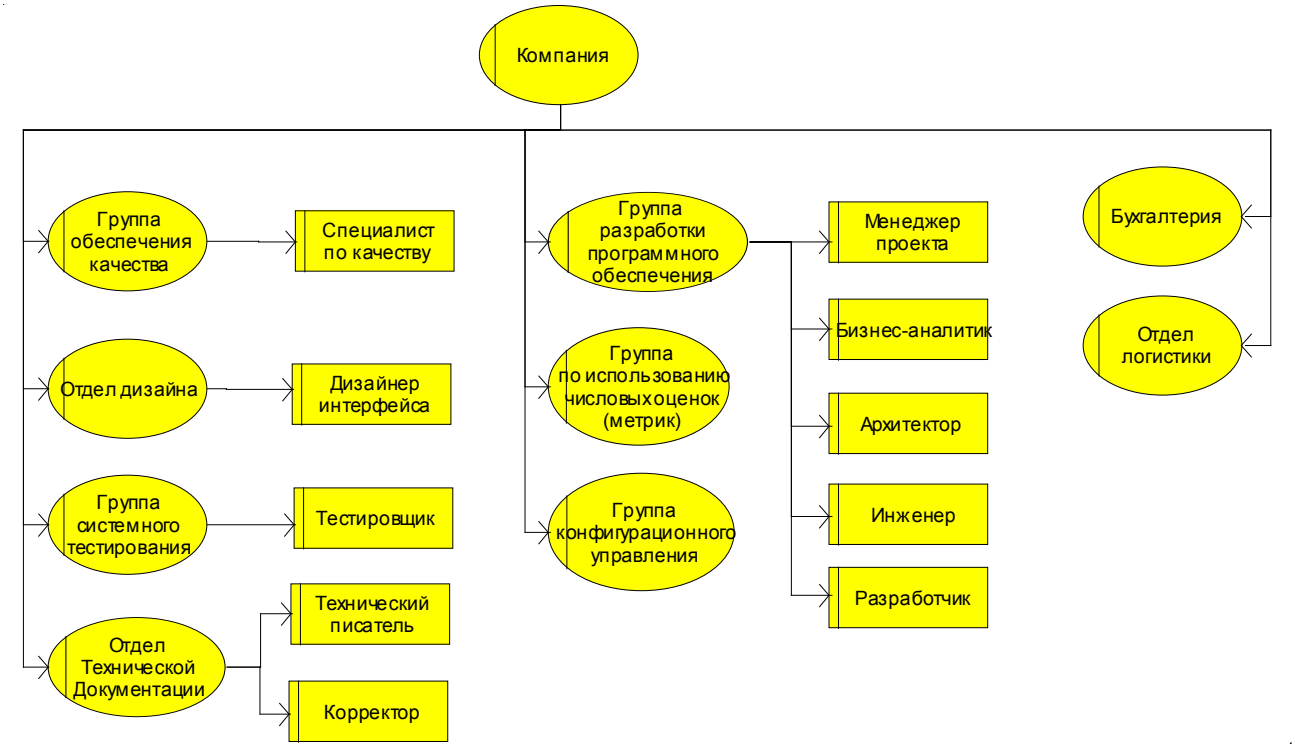


Рис. 9. Модель организационной структуры

Большинство компаний-разработчиков ПО имеют матричную или проектную организационную структуру. Предлагаемая структура организации содержит как проектные группы, так и функциональные отделы. При этом участники проектных групп не входят в существующие отделы, а сотрудники отделов участвуют в выполнении проектов.

Модель организационной структуры используется для построения модели процессов разработки, поскольку она выделяет отделы и исполнителей, ответственных за выполнение функций.

3.7.4. Модели процессов разработки жизненного цикла

Представленные в работе референтные модели могут быть детализированы организацией до уровня отдельных бизнес-процедур, должностных инструкций. Представленные модели процессов реализуются на тех фазах, на которых они являются основными (если используется MSF). В общем случае, модели процессов будут отличаться для различных этапов ЖЦ ПО

3.7.4.1. Дерево функций (структура процессов)

Предлагаемая модель процессов отличается от всех моделей, рассмотренных в работе. В отличие от ISO 12207 предлагается не три, а две группы процессов. На наш взгляд процессы обучения, усовершенствования и создания инфраструктуры являются достаточно общими, они не специфичны для разработки ПО, поэтому мы не включили их в набор референтных моделей.

Не смотря на то, что предлагаемые модели процессов в первую очередь предназначена для разработчиков заказных информационных систем для бизнеса, процесс конвертирования данных не выделен в отдельный процесс, как предлагается в CDM. На наш взгляд, конвертирование данных является частью процессов анализа требования, проектирования, разработки и тестирования. Можно рассматривать перенос данных как одно из функциональных требований.

Поддержка и сопровождение не является в полной мере процессом разработки ПО, включает в себя консультации пользователей, отслеживание изменений в требованиях государственных организациях. Безусловно, поддержка пользователей (текущее обслуживание) необходимо в конечном итоге рассматривать как единый процесс, так как для него существует общие входы (договор на поддержку), выходы (сервис для клиентов). В рамках этого процесса следует выделять подпроцессы - исправление ошибок, доработки и консультации.

Кроме представленных на карте процессов можно так же выделить процесс управления изменениями.

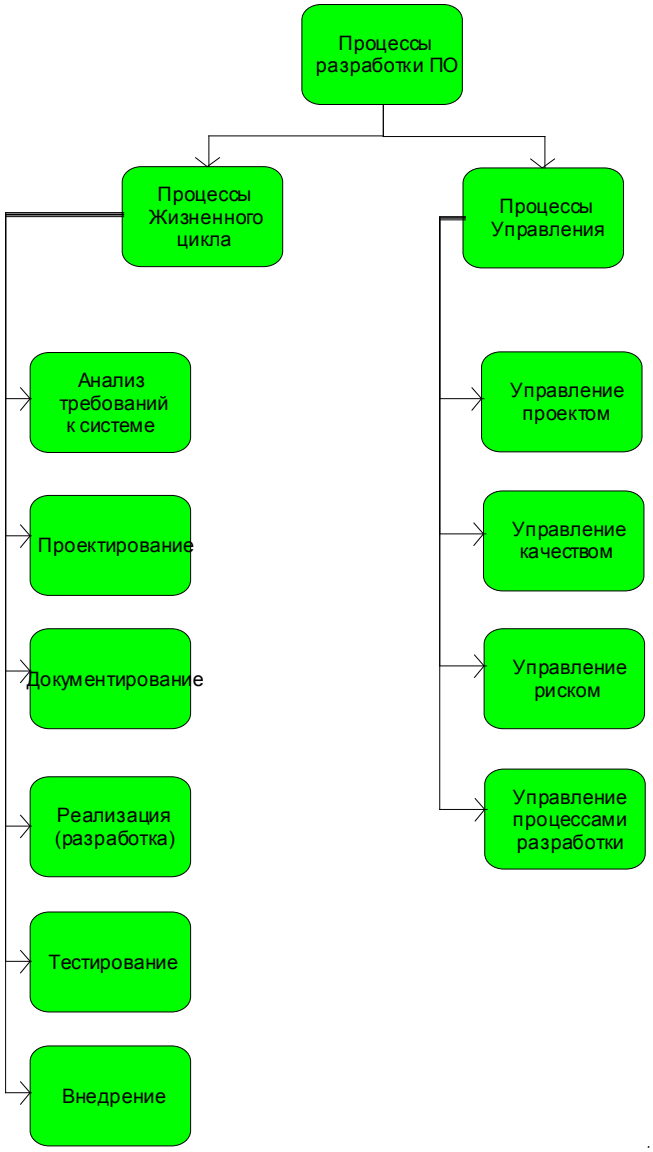


Рис. 10 Дерево функций (структура процессов)

3.7.4.2. Процесс анализа

Процесс анализа включает в себя определение требований заказчика (для заказного ПО) или определите требований рынка. Процесс анализа совпадает со стадией концептуального дизайна процесса проектирования MSF. Целью процесса является учет требований пользователей и бизнеса. В результате процесса получается описание задачи и ее решение в терминах сценариев. Кроме функциональных требований определяются требования к переносу данных, надежности, техническому обеспечению. Полный набор требований содержится в стандарте на техническое задание. Определяются контрактные условия. Входом процесса является договор на разработку с заказчиком или заявка на разработку от

отдела маркетинга, которые являются выходами процесса продаж или маркетингового анализа.

В ходе процесса требования формализуются и согласовываются с заказчиком (на предмет полноты) и с группами разработки и тестирования (на предмет понятности, исполнимости и согласованности). Здесь могут быть использованы метрики качества требований. Все требования, для которых были обнаружены проблемы, просматриваются бизнес-аналитиком (или группой, ответственной за анализ), в документацию вносятся изменения.

Предлагаемая модель требует согласования функциональных и технических требований с группами разработки, системного тестирования для оценки их понятности, выполнимости, согласованности.

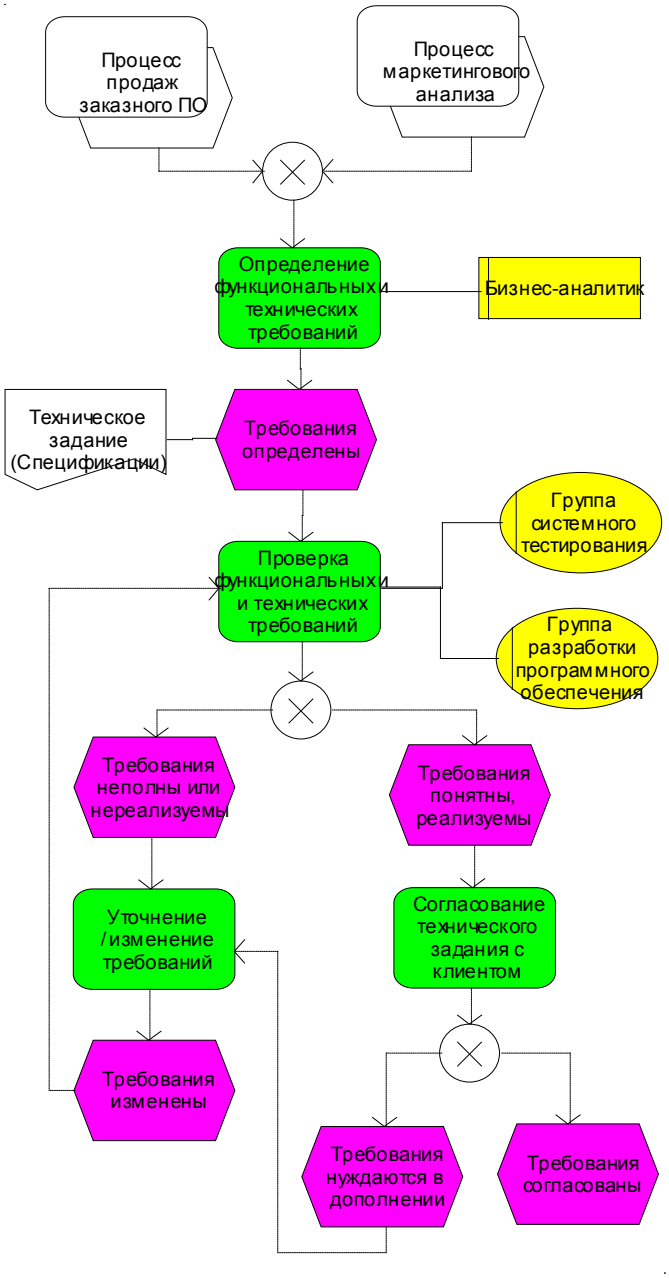


Рис. 11. Процесс анализа

3.7.4.3. Процесс проектирования

Процесс проектирования необходим для получения согласованного плана разработки. С одной стороны технический проект представляет собой техническое решение, а с другой – является продолжением планирования.

Процесс проектирования включает в себя логическое проектирование, состоящее в учете требований проектной группы и представленное как набор сервисов (абстрактная модель решения) и физическое проектирование, в

результате которого определяется технология разработки (спецификации интерфейса, физическая модель БД).

В зависимости от выбранной модели ЖЦ процесс проектирования может быть инициирован завершением процесса анализа, предоставлением согласованных требований к ПО (для каскадной модели) или начаться одновременно с процессом анализа (для MSF).

Предлагаемая модель процесса проектирования предполагает выделение дизайна интерфейса в отдельный процесс, который предшествует разработке компонент.

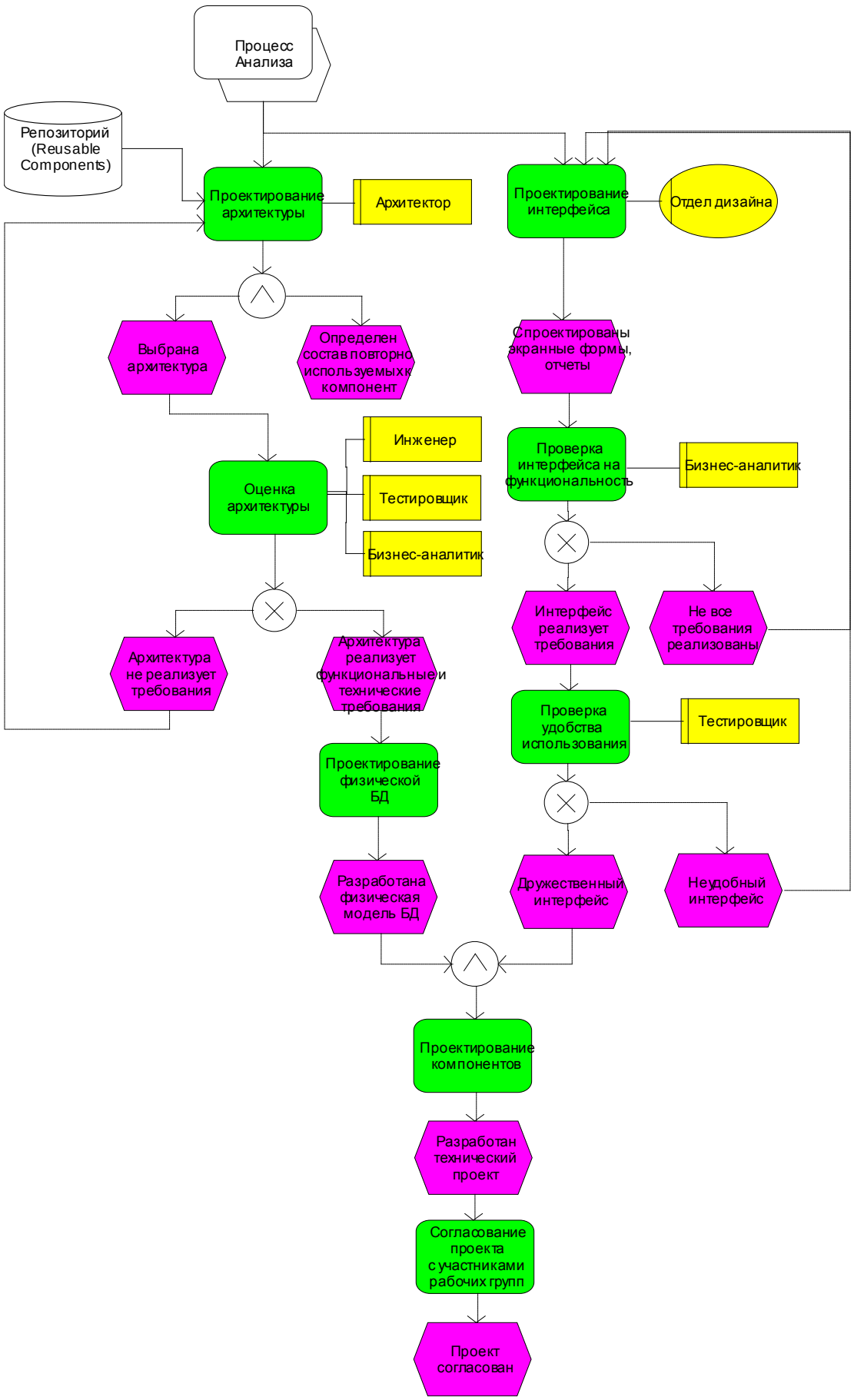


Рис. 12. Процесс проектирования

3.7.4.4. Процесс разработки

Процесс разработки имеет на входе технический проект, а на выходе – готовую к установке версию ПО (на промежуточных циклах собирается версия для системного тестирования, на последней итерации – дистрибутив для установки у заказчика). Процесс разработки включает в себя создание приложения, реализующего интерфейс и объекты бизнес-логики, создание БД, хранимых процедур и триггеров.

Можно выделить несколько различных уровней детализации процесса разработки (как и любого другого процесса). На самом высоком уровне определяется логика процесса, детализированная до функций, выполняемых группой разработки. На следующем уровне детализируются функции, выполняемые отдельным разработчиком в течение одного цикла (от получения требований до сборки версии для системного тестирования). Дальнейшая детализация приводит к рассмотрению функции создания (модификации) программного объекта (класса, хранимой процедуры) как отдельного процесса. При построении этого процесса использовался PSP (см. п. 0).Выделяются такие функции как блокировка (разблокировка) объекта в средстве управления версиями, просмотр собственного кода, тестирование компонент с использованием списка ошибок. Конечный уровень детализации процесса определяется желаемым уровнем регламентации работы, детализацией инструкций.

Существуют следующие аспекты, которые определяют требования к эффективному процессу разработки:

Необходимо использовать средства управления версиями вне зависимости от того, сколько человек работает над проектом. Средства управления версиями отличаются своей сложностью (поддерживаемыми функциями). К функциями управления версиями, выполняемым в процессе разработки, относятся: помещение приложения в средство управления версиями, создание локальной копии приложения, блокировка / разблокировка объекта для внесения в него изменений, помещение измененной версии в средство управления версиями. Средство

управления версиями позволяет исключить одновременное изменение одного объекта двумя разработчиками, позволяет проводить тестирование сделанных изменений в локальной версии, обеспечивает хранение всех версий объекта. Наиболее продвинутые средства управления версиями поддерживают несколько сборок (baseline), позволяют иметь несколько версий приложения одновременно (например, версию, поставляемую пользователю, версию для системного тестирования и новую версию). Visual Source Safe - наиболее часто используемое небольшими компаниями-разработчиками средство управления версиями, не поддерживает одновременную сборку нескольких версий в явном виде. Следует строить процесс разработки в части управления версиями с учетом возможностей используемого средства управления версиями. Отдельную проблему представляет собой управление версиями баз данных (как для самих данных, так и для хранимых процедур и триггеров). Эта проблема особенно важна для тех СУБД, где данные и процедуры представляют единое целое – хранятся в одном файле (например, MS SQL Server).

В ходе тестирования собственного кода программист может изменить данные таким образом, что они будут непригодны для дальнейшего использования (например, будет разрушена целостность данных или удалены справочники). Следовательно, необходимо предусмотреть функцию восстановления данных, или использовать копии базы данных в тестировании. Во втором случае так же необходимо поддерживать актуальность тестовых БД.

Необходимо обеспечить интеграцию процесса разработки с другими процессами (тестирования, планирования, документирования и т.д.) Эта интеграция обеспечивается за счет общих событий (например, сборка дистрибутива – процесс разработки инициирует тестирование дистрибутива – процесс тестирования), за счет общих информационных ресурсов (например, новые типы ошибок, выявленные разработчиками, попадают в БД ошибок и используются в процессе тестирования) или за счет общих функций, когда результаты функции приводят к событиям, принадлежащим разным процессам. Этот аспект имеет наибольшее значение для процесса разработки, поскольку он находится в центре совокупности процессов компании.

Необходимо обеспечить несколько уровней планирования работы. На самом высоком уровне используется план, получаемый в результате процесса управления проектами: План ЖЦ ПО и План разрабатываемой функциональности. На основе этого плана создается план разработки компонент. Этот план создается совместно группой разработки, группой тестирования и менеджером проекта. На основе плана разработки компонент группа разработки формирует внутренние планы – планы выпуска внутренних версий (не обязательно формальные) в которых определяется, что будет разработано в течение недели. Эти планы составляются самими разработчиками и являются их обязательствами.

Предлагаемый процесс разработки учитывает специфику разработки информационных систем для бизнеса. Заполнение БД данными вынесено в отдельный подпроцесс. Описан порядок создания структуры каталогов, создание шаблона приложения из модели, разработанной в процессе проектирования. Установлен порядок включения повторно-используемых компонент.

Предлагаемый собственный процесс разработки имеет следующие преимущества: контролируется блокировка / разблокировка модуля проекта, просмотр кода предшествует его тестированию, сведения об ошибках документируются, при просмотре и тестировании используется список типовых ошибок, оценка кода с помощью метрик осуществляется так же до тестирования. На наш взгляд следует просматривать и оценивать код до тестирования, поскольку тестирование занимает значительную часть времени разработчиков, и если после завершения тестирования код не будет удовлетворять требованиям к понятности, сложности, то его изменение приведет к необходимости повторения тестирования, и, следовательно, к потере результатов предыдущего тестирования. На практике это приводит к тому, что если просмотр кода и использование метрик качества и показывают необходимость перепроектирования кода, этого не происходит, поскольку не тестирование уже потрачено достаточно много времени.

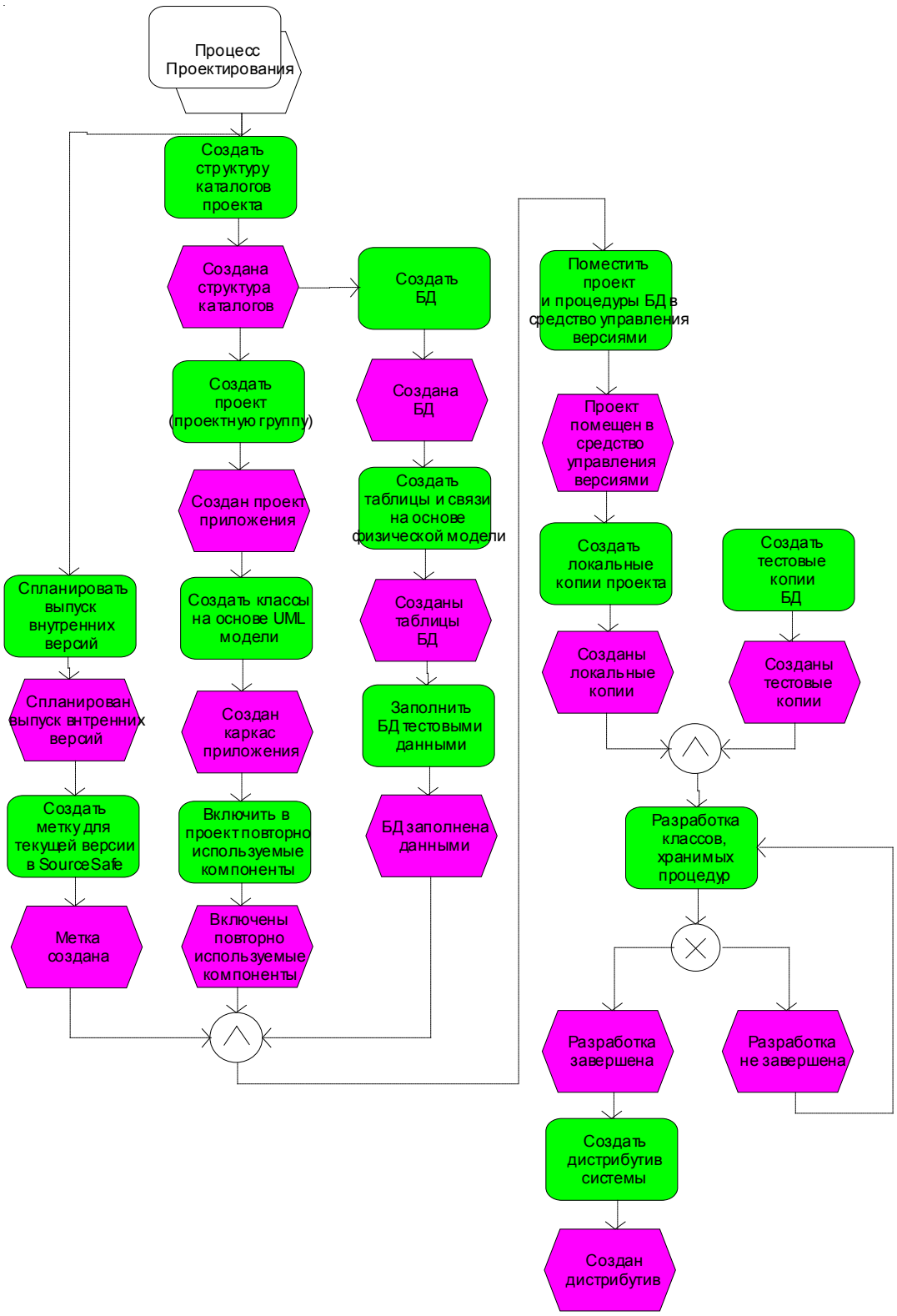


Рис. 13. Процесс разработки (общий)

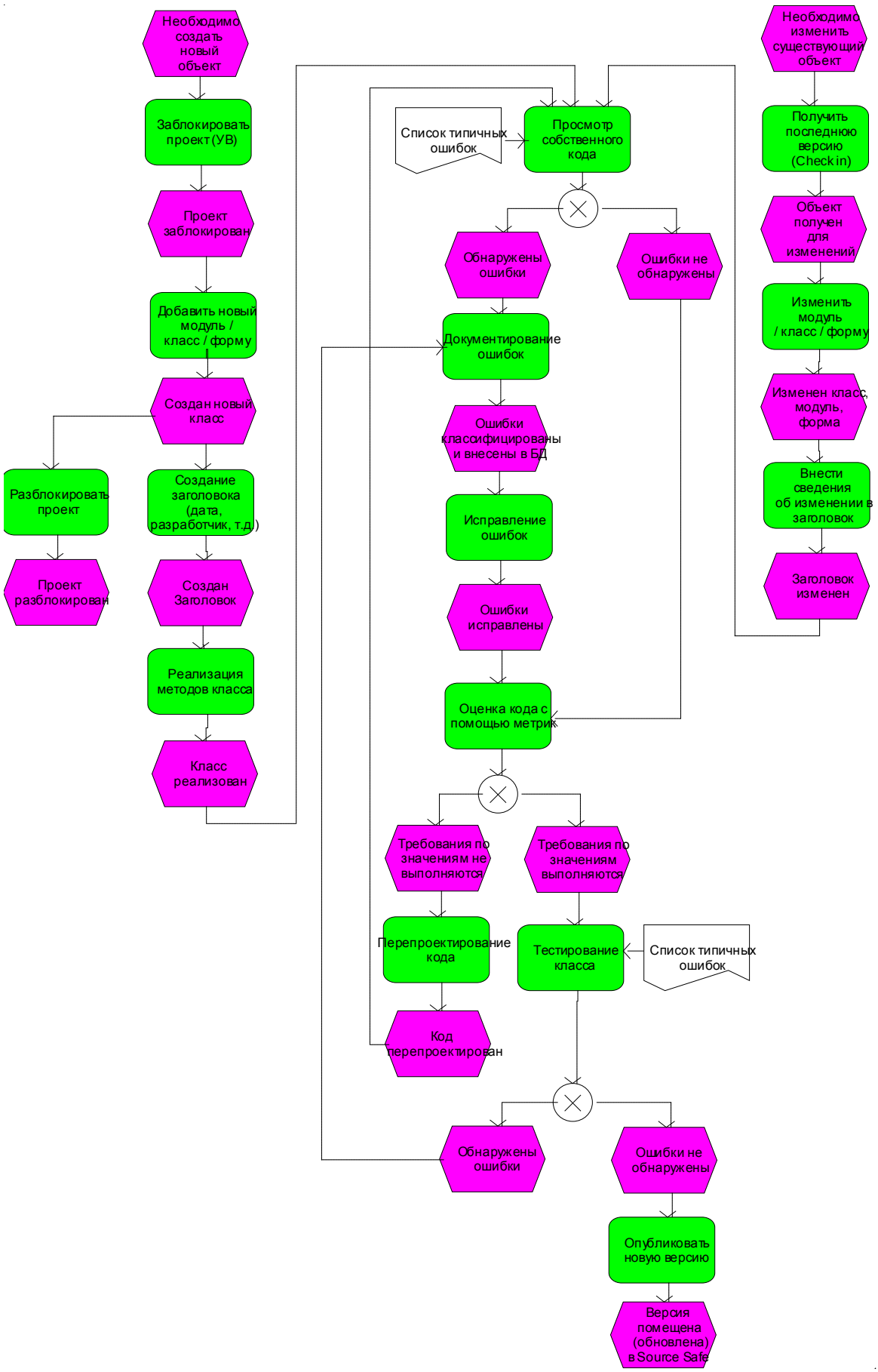


Рис. 14. Собственный процесс разработки

3.7.4.5. Процесс тестирования

Тестирование частично включается в процесс разработки. Процесс тестирования выделяется в отдельный процесс, поскольку он имеет свои задачи и выполняется отдельной группой (группой тестирования). Независимость группы тестирования от группы разработки и менеджера проекта является одним из условий обеспечения качества. Тестированию подлежат все разрабатываемые продукты: документация, программное обеспечение. Тестирование ПО включает в себя как тестирование качественных характеристик ПО (см. п. 2.2. , 2.1.), так и тестирование дистрибутива системы на различных платформах (на которых будет использоваться система), тестировании документации.

Процесс тестирования начинается раньше, чем процесс проектирования. После того, как в процессе анализа создаются функциональные и технические требования, и еще до того, как эти требования будут полностью согласованы с заказчиком и группой разработки, создается первая версия плана тестирования. Это необходимо для того, что бы учесть затраты времени на тестирование при составлении плана ЖЦ ПО, согласования даты выпуска с заказчиком. Кроме того, в некоторых случаях может потребоваться приобретение дополнительных инструментальных средств тестирования или разработка специализированного ПО. Группа тестирования может заказывать создание ПО для тестирования группе разработки или осуществлять разработку самостоятельно.

В тестировании используются метрики для предсказания времени тестирования, ожидаемой продолжительности тестирования, оценки рисков (например, метрики полноты спецификаций).

Оптимизация тестирования и отладки приложений представляется одной из наиболее актуальных задач. В отличие от этапа проектирования и разработки, длительность которых определяется требованиями к конечному продукту и производительностью, длительность отладки может быть установлена произвольно. То есть, может быть задано время отладки или условие (условия), при котором отладка заканчивается. При этом существующие модели строятся на

основе предположения, что между временем отладки и количеством необнаруженных ошибок существует обратная зависимость.

Необходимо различать *первичные* и *вторичные* проявления ошибки. Первичные ошибки обнаруживаются в тексте программ и подлежат корректировке. Вторичные ошибки представляют собой искажение выходных результатов исполнения программы, именно с вторичными ошибками сталкивается конечный пользователь на этапе эксплуатации программы. Следовательно, именно прогнозируемое количество вторичных ошибок должно быть критерием при принятии решения о продолжении или прекращении отладки.

При этом ожидаемое количество вторичных ошибок определяется ожидаемым количеством первичных ошибок и структурой приложения. Например, если в процедуре, используемой для формирования десяти различных отчетов содержится одна ошибка, то пользователь столкнется с десятью ошибками.

Для вторичных ошибок значимыми являются:

- место возникновения (бизнес-функция, отчетность, и т.д.)
- влияние на работу пользователя (отказ, снижение работоспособности, искажение данных);
- частота проявления ошибки (связана с необходимой частотой выполнения операции, которая приводит к ее проявлению).

Таким образом, следует не только учитывать количество обнаруженных первичных ошибок, но прогнозировать вторичные ошибки, уделяя особое внимание наиболее часто используемым функциям, которые необходимы для бизнес-процесса клиента.

Кроме того, следует различать ошибки и недоработки. Недоработки не приводят к существенному ухудшению работы программы, и, в худшем случае, снижают эстетическое восприятие. Примером недоработки является отсутствие выравнивания полей формы, недостаточный размер поля в отчете для символьной

строки. Время, необходимое для обнаружения и исправления всех недоработок, может быть сопоставимо со временем разработки приложения[16].

Процесс управления рисками тесно интегрирован с процессом тестирования. Именно в ходе тестирования требований, проектных решений и реализации оцениваются возможные риски и вырабатываются меры по их предотвращению.

Существуют два уровня детализации процесса тестирования: формализация процесса в рамках жизненного цикла ПО, при этом делается акцент на тестировании различных продуктов, документации, дизайна интерфейса, проекта, исполнимого кода и формализации отдельной операции тестирования, акцент делается на использовании метрик для прогнозирования значения и фиксации полученных результатов для использования в других проектах и улучшения процессов разработки.

Большую долю времени в процессе тестирования занимает подготовка сценариев и наборов тестовых данных. Следует отметить, что тестовые данные необходимы также группе разработки для проведения внутреннего тестирования. Поэтому необходимо синхронизировать процесс разработки и тестирования – тестовые данные должны быть подготовлены до начала разработки компонент. Тестовые данные могут подготавливаться как группой разработки, так и группой тестирования, в зависимости от степени загрузки сотрудников, учитывая многопроектный характер работ.

Наибольшую сложность в моделировании процессе тестирования представляет его циклический характер, обусловленный тем, что при исправлении ошибок, найденных в результате тестирования, могут быть внесены новые ошибки, которые так же необходимо обнаружить, и исправить. В крайнем случае, процесс может быть несходящимся, это произойдет в том случае, если при исправлении ошибок будет вноситься больше ошибок, чем было исправлено. Очевидно, что моделируя отдельно процесс разработки и тестирования, достаточно сложно отобразить существующее между ними взаимодействие.

---

[16] Реальным подтверждением тому стал проект разработки системы V4 (Анализ продаж и планирование маркетинговых компаний) VB 6.0, работоспособная версия которого была выпущена за полгода, а на совершенствование графического интерфейса и отчетности ушло более года

Можно выделить несколько этапов процесса тестирования, подготовка тестирования: составление плана тестирования, разработка сценариев, подготовка тестовых данных, тестирование требований, тестирование проекта, тестирование ПО и документации: выполнение сценариев, тестирование дистрибутива. В каждом случае при обнаружении ошибок в документации или ПО, а также при обнаружении несоответствия характеристик заданным требованиям группа тестирования документирует обнаруженные ошибки и несоответствия, присваивает им уровень важности. Кроме того, обновляется БД типов ошибок, используемая при составлении сценариев тестирования.

Исправление ошибок может занять много времени, которое может быть неучтено в плане проекта. Использование методов прогнозирования времени тестирования для корректировки планов является перспективным направлением улучшения процессов.

Кроме того, можно выделять в самостоятельные процессы процесс тестирования кода, документации, дистрибутива (полноты), производительности. Предлагаемая модель процесса не раскрывает циклический характер процесса тестирования, в модели сознательно опущено тестирование документации. Модель охватывает полностью ЖЦ ПО: от подготовки требований, до выпуска готовой версии. Предусматривается оценка рисков на основе анализа требований, разработка плана тестирования. Следует отметить, что план тестирования может повлиять на план разработки (например, в первую очередь будут разработаны те компоненты, тестирование которые займет больше времени).

Основная итерация в рамках процесса тестирования осуществляется в соответствии с планом выпуска внутренних версий. В зависимости от масштаба приложения такая версия может разрабатываться с периодичностью от одного дня до одной недели. Внутренняя версия собрана, когда все входящие в нее компоненты протестированы разработчиком.

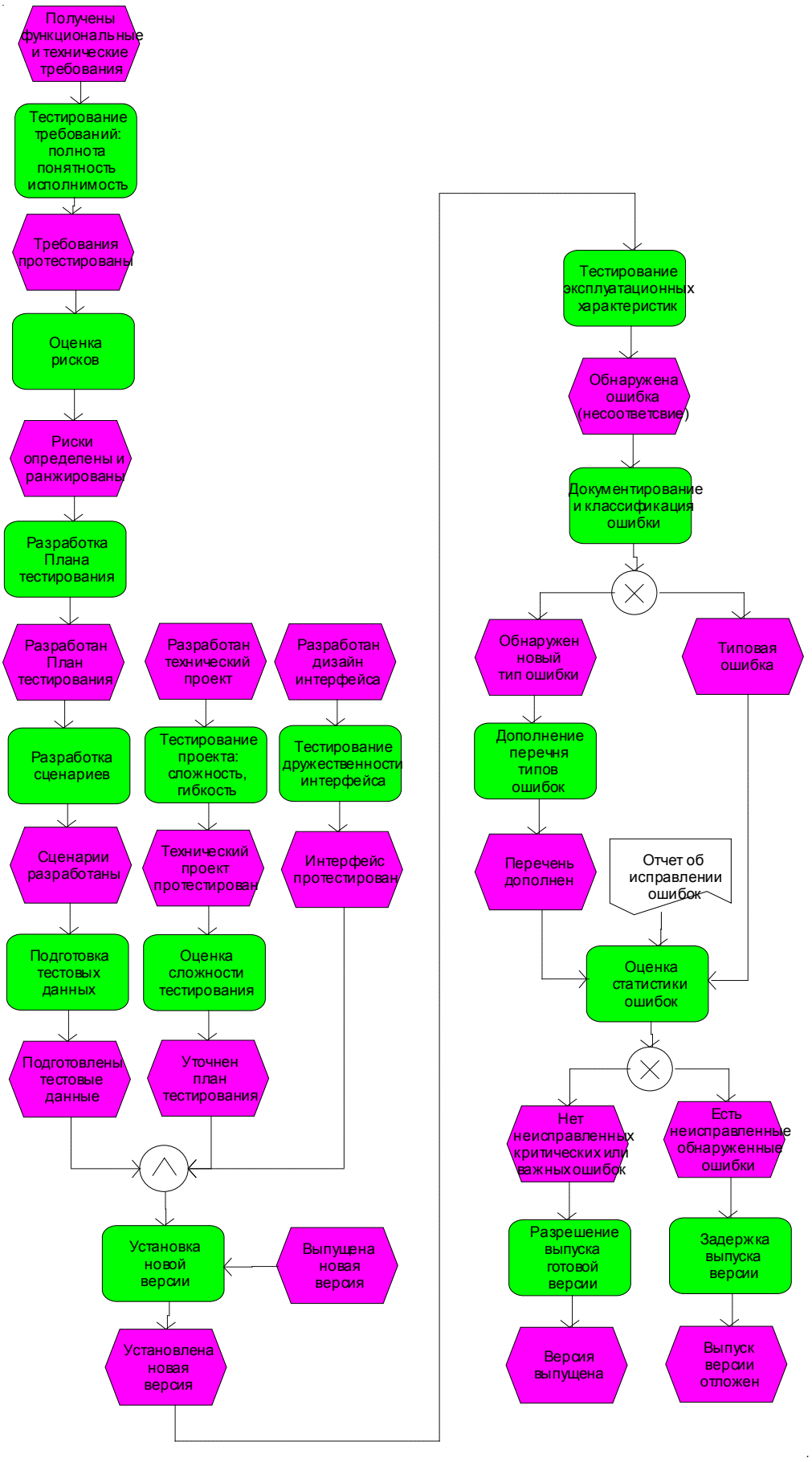


Рис. 15. Процесс тестирования

3.7.4.6. Процесс внедрения

Процесс внедрения начинается задолго до завершения тестирования. Внедрение требует создание плана развертывания. Внедрение, на наш взгляд, представляет собой один из наиболее рискованных процессов, поскольку здесь результат во многом зависит от клиента, его способности предоставить необходимые технические ресурсы, специалистов. Требуется согласование этапов внедрения не только с процессом разработки и тестирования, но и с событиями, происходящими у заказчика.

Необходимо управлять процессом внедрения, то есть процесс управления проектом должен быть тесно интегрирован с процессом внедрения. Следует прогнозировать риски внедрения на ранних этапах разработки, строить сам план ЖЦ ПО с учетом этих рисков (см. п. 1.4.4.2, 3.5.5), учитывая возможность поэтапного перехода на новую систему, необходимость обучения, требования к переносу данных, а так же необходимость изменения бизнес-процессов заказчика.

Необходимо проанализировать, каким образом изменятся процессы заказчика после внедрения разрабатываемого ПО, согласовать эти изменения до эксплуатации, убедиться, что заказчиком подготовлены новые версии инструкций и процедур.

Представленная модель процесса внедрения предусматривает установку бета-версии ПО до начала обучения пользователей, что бы обучение происходило на системе, максимально похожей на ту, которой будет пользоваться заказчик. Поэтому инфраструктура должна быть развернута как можно раньше. Кроме того, пользовательская документация (руководство пользователя, файл контекстной помощи) должна быть разработана до начала обучения. Процесс внедрения имеет несколько выходов: обученных пользователей, установленную систему и перенесенные данные.

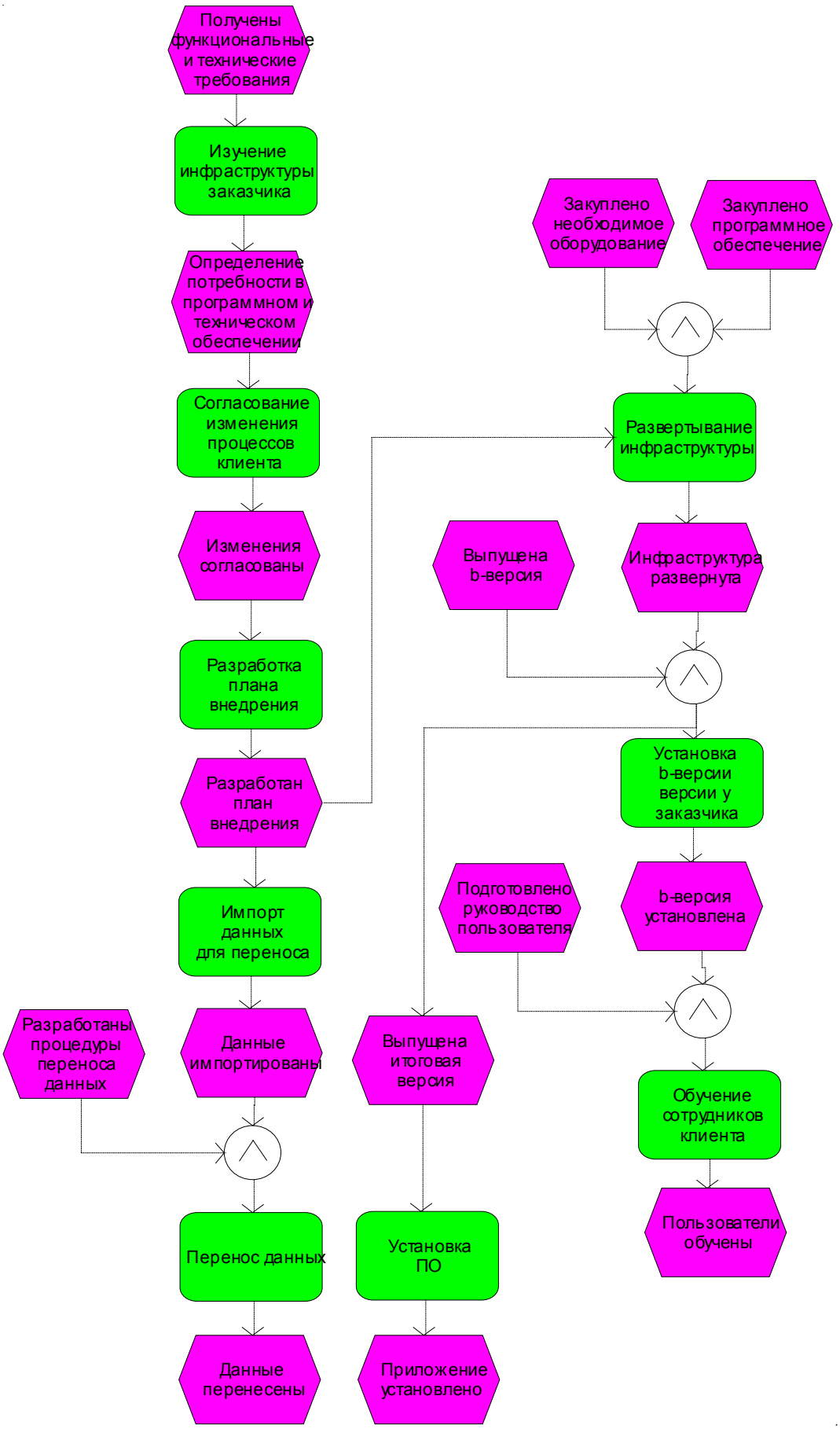


Рис. 16. Процесс внедрения

3.7.4.7. Процесс документирования

Процесс документирования вынесен в отдельный процесс, поскольку в средних и крупных организациях документированием занимается выделенное подразделение. Составление документации требует специализированных навыков, инструментов и нуждается в организованном процессе. Документирование должно быть синхронизировано с процессом проектирования. Требования к разрабатываемой документации содержаться в техническом задании. Документация может подготавливаться для распространения в бумажном и (или) электронном виде. Специфическим видом документации является электронная справка.

Следует разделять две возможные трактовки процесса документирования: документирование как подготовка отдельного документа к публикации в бумажном или электронном виде, включающая в себя набор, корректуру, верстку, или документирование как процесс создания полного комплекса документации к ПО: технического задания, технического проекта, описания программы, руководства пользователя в бумажном и электронном виде.

В представленной системе моделей разработка технического задания (спецификации) осуществляется в ходе процесса анализа, технический проект создается в ходе процесса проектирования, описание программы может быть получено с помощью метода обратного проектирования с использованием CASE средств, необходимо только сопровождать код комментариями. Таким образом, документирование касается в первую очередь подготовки пользовательской документации.

Разработка руководства пользователя может начаться до завершения разработки приложения и завершиться одновременно с выпуском бета-версии.

Необходимо завершить подготовку файла справки до сборки бета-версии.

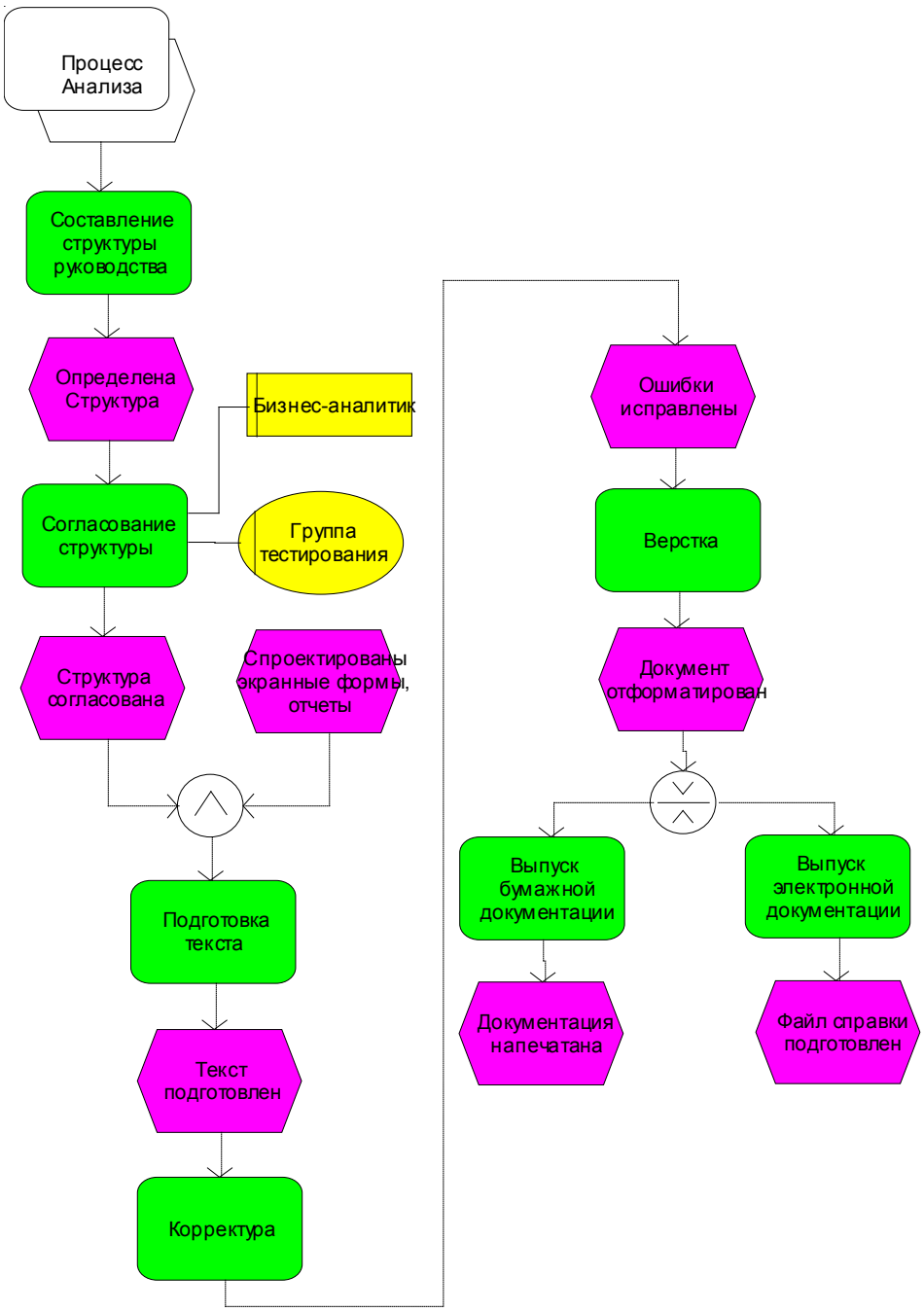


Рис. 17. Процесс документирования

3.7.5. Модели организационных (управленческих) процессов

3.7.5.1. Управление проектом

Процесс управления проектом разработки ПО инициируется одновременно с процессом анализа в тот момент, когда заказчик подписывает договор на разработку, или отдел маркетинга принимает решение о разработке нового продукта (новых функций). Определяется название проекта, устанавливаются цели проекта, контрольные точки, границы проекта. Управление проектом включает в

себя планирование проекта, определение ресурсов, отслеживание выполнения проекта (включая сроки, результаты и бюджет). Кроме того, в процессе управления проектом должны учитываться все изменения, в требованиях.

Одной из проблем управления проектами в разработке ПО является двойственность структуры работ плана. С одной стороны работы следует планировать работы в разрезе отдельных разрабатываемых процедур, классов, модулей, компонент, поскольку при оценки трудоемкости наиболее точные оценки могут быть получены в разрезе разрабатываемых компонент, с другой – необходимо планировать в разрезе функциональности, доставляемой пользователю, поскольку именно функциональность поставляется в конечном итоге заказчику. В зависимости от выбранной архитектуры и проекта, структура компонент и структура функций ПО будет отличаться между собой. Следует отметить, что в некоторых проектах (компаниях) существует тенденция свести к минимуму расхождения между составом компонент и функциональностью, такой подход лежит в основе RUP.

Планирование разработки в разрезе компонент невозможно до проектирования архитектуры, а планирование в разрезе функций возможно после проведения анализа и создания технического задания. С самого первого шага существует возможность планирования в разрезе фаз и этапов жизненного цикла.

Производительность аналитика, программиста, тестировщика зависит от следующих факторов:

Внутренних факторов, характеризующих разработчика: квалификации, профессиональных навыков;

Организации процесса разработки (используемых инструментов, качества получаемых материалов, методического обеспечения);

Организации проекта - характера загрузки (например, производительность при работе над одним проектом при полной нагрузке будет выше, чем при одновременной работе над двумя проектами в режиме разделения времени);

Организации работы: мотивации.

При планировании проектов следует учитывать, что определенная часть времени программистов, тестировщиков и других технических специалистов

должна быть зарезервирована для повышения квалификации: изучения новых технологий, продуктов, причем в сфере разработки ПО время обучения может оказаться выше, чем в других видах деятельности, особенно при использовании новых технологий.

В том случае, если состав функций существенно отличается от состава компонент, а сам проект является достаточно масштабным, представляется целесообразным иметь три взаимосвязанных плана проекта: План жизненного цикла ПО, построенный на основе выбранной модели *ЖЦ ПО, План функциональности и План Компонент.*

План ЖЦ ПО содержит основные стадии и фазы проекта, контрольные точки, даты встреч с покупателем, даты поставки версий. Именно этот план используется в переговорном процессе и согласовывается с покупателем.

План функциональности содержит перечень реализуемых ПО функций: бизнес-транзакций, отчетов, возможности настройки, поиска и т.д. Этот план используется для отслеживания разрабатываемой функциональности. Между планом ЖЦ ПО и планом функциональности устанавливаются связи через выпускаемые версии. После того, как разработана архитектура, создается План разработки компонент. *План разработки компонент* создается, чтобы распределять работы между разработчиками, на основе этого плана строиться план компонентного тестирования.

План тестирования разрабатывается группой тестирования и передается менеджеру проекта для интеграции в план жизненного цикла.

Второй проблемой является согласование плана проекта и бюджета с заказчиком. Бюджет определяется менеджером проекта исходя из планируемой трудоемкости разработки требуемой функциональности, стоимости работ и нормы прибыли, но, с другой стороны, бюджет представляет собой затраты заказчика и ограничен его инвестиционными возможностями по этому проекту. Проект бюджета должен быть согласован до начала процесса проектирования. В управлении проектами существуют несколько видов контрактов: контракт с твердой ценой – затраты определяются в начале проекта, контракт с возмещением затрат (с возможностью установления максимальной гарантированной выплаты).

Тип контракта определяет распределение рисков между заказчиком и исполнителем.

Следует предусмотреть несколько этапов согласования бюджета, а так же процедуру корректировки бюджета в случае пересмотра планов, требований или необходимости привлечения дополнительных ресурсов.

Основными задачами управление проекта является выбор (разработка) совокупности согласованных моделей процессов разработки (планирование), а так же обеспечение следованию этим процессам (организация, контроль). Безусловно, в ежедневной работе менеджера проекта использование референтных моделей процессов в формате eEPC является неоправданно трудоемким. В действительности следует использовать шаблоны планов проекта, построенные на основе моделей процессов[17]. Референтные модели предназначены для изменения процессов, а не для организации ежедневной работы. Безусловно, можно использовать референтные модели при создании планов (Плана ЖЦ ПО).

Предлагаемая модель процесса управления проектом отражает некоторые функции в свернутом виде, они могут быть представлены в виде соответствующей eEPC диаграммы. Например, отслеживание планов и бюджетов включает в себя отслеживание проделанной работы, контроль за израсходованными средствами, составление отчета о состоянии проекта(Project Status Report) с определенной периодичность. Функция отслеживания проекта тесно связана с управлением рисками, обычно отчет о проекте включает в себя анализ основных текущих рисков проекта и мероприятий по предотвращению этих рисков

Следует отметить, что использование данных предыдущих проектов для оценки существующей практики управления рисками возможно лишь при наличии документального подтверждения – описания результатов анализа рисков, протоколов обмана информацией о рисках, распоряжениях, направленных на уменьшение / предотвращение рисков. Поэтому можно рекомендовать всем

[17] MS Project 2000 имеет два шаблона для планирования разработки ПО: Software Development и MSF Application Development

разработчикам ПО, а особенно компаниями, занимающимся разработкой заказного ПО, разработать формы документов, содержащих информацию о рисках.

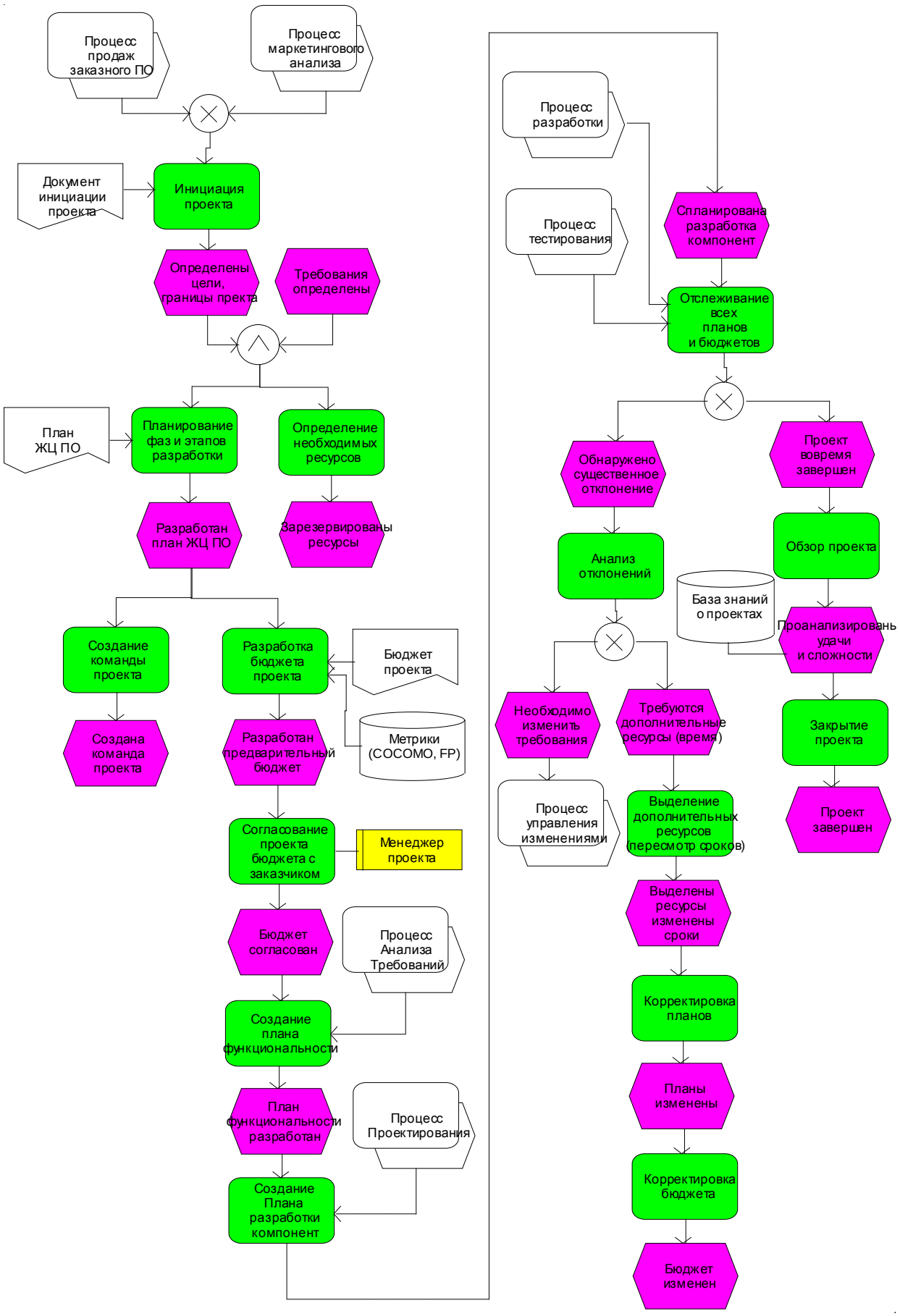


Рис. 18. Процесс управления проектом

На наш взгляд, основным способом управления рисками является совершенствование процессов разработки ПО. Совершенствование процесса управления изменениями, создание однозначной процедуры приема/отклонения изменений позволяет уменьшить риск изменения требований. Каждое требование должно тестироваться и подвергаться количественному анализу до того, как оно будет принято к исполнению в текущей версии. Анализ требований должен осуществляться с точки зрения соответствия целям приложения, реальным возможностям их реализации (наличие необходимых ресурсов), соотношение между полезностью изменения и затратами

3.7.5.2. Управление качеством

Обеспечение качества преследует цель обеспечения необходимой прозрачности процессов разработки, а управление качеством призвано обеспечить количественное понимание проектов разработки и достижение установленных целей качества. СММ раскрывает проблему управления качеством на двух уровнях: Уровень 2. Повторяемый. Обеспечение качества и Уровень 4. Оптимизирующий. Управление качеством.

В управление качеством включает в себя следующие процессы: разработка программы метрик, оценка процессов, оценка программных продуктов, выработка мероприятий по устранению отклонений, оценка степени удовлетворенности заказчика характеристиками ПО, уровнем обслуживания, оценка соответствия существующих процессам требованиям к качеству разрабатываемой продукции.

Представленная модель процессов управления качеством включает все перечисленные процессы. Модель опирается на ЖЦ ПО, процесс привязан к проекту разработки. Начало процесса управления качеством совпадает с определение целей проекта, для каждого проекта разрабатывается план обеспечения качества. С учетом требований к качеству ПО, учетом рисков разрабатывается программа метрик (см. п. 3.5.1).

В ходе выполнения проекта группа обеспечения качества проверяет соответствие реализуемых процессов стандартам, выявляет причины отклонений. Группа обеспечения качества проводит независимую проверку разрабатываемого

ПО, документации, и информирует о результатах проверок, как вышестоящий менеджмент, так и группу разработки. Кроме того, группа обеспечения качества оценивает удовлетворенность заказчика разрабатываемым ПО и документацией, предоставляемым сервисом. По результатом этой оценки уточняются требования к качеству, как для результатов конкретного проекта, так и для компании, дополняется политика качества.

Поскольку между некоторыми характеристиками качества существует обратная зависимость (например, эффективность программы и мобильность), необходимо устанавливать целевые параметры качества для проекта и их относительные приоритеты.

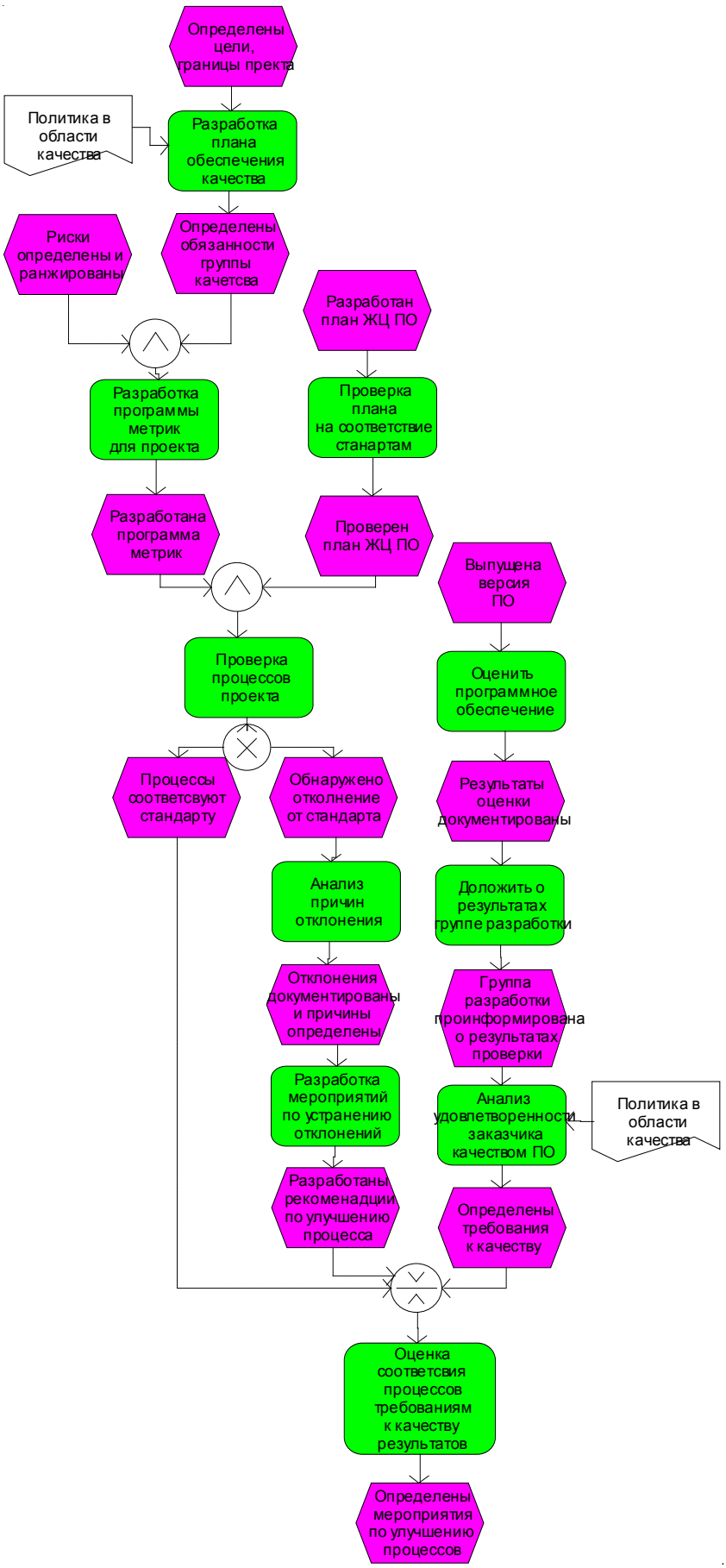


Рис. 19. Процесс управления качеством

ЗАКЛЮЧЕНИЕ

В ходе проведенного исследования в соответствии с целью и задачами исследования были получены следующие основные новые научные результаты:

Подтвердилась гипотеза, что специфика управления разработкой ПО требует адаптированных подходов к управлению, и, следовательно, специализированных инструментальных средств. Теория управления разработкой ПО опирается на концепции общей теории управления: концепцию глобального управления качеством (TQM), теорию управления проектами, теорию управления бизнес-процессами и концепцию реинжиниринга. Современная теория управления разработкой программным обеспечением развивается одновременно в двух основных направлениях:

Во-первых, в сторону специализации концепций, сформированных в рамках международной теории управления, их адаптации и расширения в соответствие со спецификой этого вида деятельности (SPICE, Prince, CMMI). В этом направлении развиваются методологии, направленные на создание ИТ-компании, обеспечивающей заданное качество выполнения проектов,

Во-вторых, в сторону теоретического обобщения практического опыта разработки программного обеспечения, (Agile Software Development, eXtreme Programming).

Классификация существующих методов, моделей и стандартов управления разработкой ПО раскрывает состав современных инструментальных средств управления разработкой ПО, иллюстрирует характер взаимосвязей между ними. В центре современной теории управления разработкой ПО две группы методологий: методологии совершенствования деятельности компании (SPICE, CMMI) и проектные методологии. Существует два класса проектных методологий: методологии разработки ПО (RUP, MSF, Crystal, XP) и методологии внедрения информационных систем (Accelerated SAP, Oracle CDM, On Target). Проектные методологии опираются на стандарты процессов жизненного цикла. Методологии совершенствования деятельности компании опираются на положения концепции TQM. Прогнозируемые (предикативные) методологии фокусируются на детальном

планировании будущего. Адаптивные методологии нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. Существует два класса проектных методологий, в зависимости от подхода: класс, основанный на подходах TQM, процессном управлении и непрерывных улучшениях и класс, основанный на взаимодействии, итерациях и командной работе.

В основе проектных методологий и методик совершенствования процессов лежат определенные принципы. Для выбора методологий на основе метода верификации принципов необходимо определить, на каких принципах основывается та или иная методология, и определить набор принципов, адекватных для проекта и компании.

Риски объективно присущи процессу разработки ПО. Классификации (таксономии) рисков являются эффективным инструментом выявления рисков. Для управления двумя категориями рисков (неизбежными рисками и добровольными рисками) необходимо использовать различные подходы. Управление рисками включает в себя как создание инфраструктуры управления рисками, так и непосредственное управление рисками в ходе реализации проектов. Различается проактивное и реактивное управление рисками, а также стратегическое и оперативное управлении рисками.

Использование количественных оценок для планирования и контроля выполнения ИТ-проектов является показателем зрелости процессов разработки. Метрики служат для получения количественных оценок. Выбор метрик зависит от технологии разработки, масштабов проекта, доступных технических средств и накопленного опыта.

Практически все методологии, модели и стандарты предлагают модель процессов ИТ-компании. Процессы делятся на две группы: процессы проектов и организационные процессы. К проектным процессам относятся процессы разработки и процессы управления проектом. Процессы разработки основываются на определенной модели жизненного цикла разработки ПО. Модели процессов разработки прогнозируемых методологий основываются на каскадной модели ЖЦ ПО, а модели процессов адаптивных методологий используют итеративную модель ЖЦ ПО. Модели и стандарты не содержат представления логики

процессов (динамики процессов). Это делает актуальным разработку референтных моделий процессов ИТ-компании. Референтные модели уменьшают трудозатраты при формализации процессов, а также служат для передачи лучших практик и обмена опытом.

Выбор и адаптация инструментальных средств управления разработкой ПО должна производится с учетом специфики корпоративной культуры, маркетинговой стратегии, особенностей проекта и разрабатываемого продукта. Существует два основных вида маркетинговых стратегий ИТ компаний: разработка готовых ИТ-продуктов (продуктовая стратегия) и разработка заказных систем (сервисная стратегия). Основной фокус для разработчиков тиражируемого ПО – это продукт, а для разработчиков заказных систем – процесс. Фокус определяет организационную структуру, модели процессов, структуру метрик.

Формализация процессов является первым необходимым шагом перехода к процессному правлению. В ходе формализации целесообразно использовать модели процессов, содержащихся в проектных методологиях и моделях зрелости, а также использовать референтные модели. Помимо референтный моделей практическую ценность имеют классификация документации компании-разработчика ПО и классификация внутрифирменных стандартов.

ОСНОВНЫЕ ТЕРМИНЫ, ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИ

BSC (Balanced Scorecard) – Система сбалансированных показателей, используемая для управления компанией.

COCOMO (COnstructive COst Model) – Метрика оценки стоимости и времени проекта, разработана Барри Боэмом (Barry Boehm)

CMM (Capability Maturity Model) — модель зрелости процессов разработки

DoD (Department of Defense) – Министерство обороны США

FP (Functional Point) Функциональная точка – единица измерения объема программного обеспечения.

IEEE (Institute of Electrical and Electronics Engineers) - .Институт Электронной и Электротехнической Инженерии

MAELOC (Million assembly-equivalent lines of code) – Метрика объема миллионов строк кода в эквивалентных командах языка ассемблера

MTFB (Mean Time Between Failures) - Среднее время между провалами – Характеристика качества программного обеспечения

SATC (Software Assurance Technology Center) – Центр управления качеством программного обеспечения NASA

SEI (Software Engineering Institute) – Институт разработки программного обеспечения DoD

SLOC (Software lines of code) - Метрика. Основанная на подсчете количества строк кода. См. MAELOC

SPICE (Software Process Improvement and Capability determination) – стандарт «Улучшение процессов и оценка их возможности»

TQM (Total Quality Management) — концепция управления качеством (тотальное управление качеством)

МЭК – Международная Электротехническая Комиссия

ИСО (ISO) – международная организация по стандартизации. International organization for standartization

Возможность процесса разработки ПО (software process capability) - описывает ожидаемый результат, который можно достигнуть, следуя процессу

разработки.

Зрелость процесса разработки ПО (software process maturity) – это степень определенности, управляемости, измеряемости и эффективности процесса разработки ПО.

Программа — алгоритм, записанный в форме, воспринимаемой ЭВМ [ГОСТ 19.004-80].

Программное обеспечение (ПО) — интеллектуальный продукт, состоящий из программ, процедур и правил и любой другой связанной с ними документации, относящихся к функционированию системы обработки данных [ИСО 2382-1].

Процесс — комплекс взаимосвязанных действий, которые трансформируют входы в выходы [ИСО/МЭК 12207].

Процесс разработки ПО — процесс или набор процессов, которые используются организацией или проектом для планирования, управления, осуществления, проверки, руководства и совершенствования связанных действий ПО [ИСО 9000-3-91].

Разработка — все виды деятельности, выполняемые для создания продукции программного обеспечения [ИСО 9000-3-91].

Система качества — совокупность организационной структуры, методик, процессов и ресурсов, необходимых для осуществления общего руководства качеством.

Стандарт – это "документ, установленный с согласия и одобренный уполномоченной организацией, который определяет правила руководства или характеристики операций или их результатов для общего пользования с целью поддержания определенного порядка в данной среде". ISO/МЭК 2: 1996)

Управление качеством — методы и виды деятельности оперативного характера, используемых для выполнений требований к качеству [ИСО 9000].

ЛИТЕРАТУРА

1. Аалдерс Роб. ИТ аутсорсинг. Практическое руководство. Альпина Бизнес Букс, 2004 г. 300 стр.
2. Агафонов В.Н. Спецификация программ: понятийные средства и их организация. — Новосибирск: Наука, 1987. - 240 с.
3. Агафонов В.А. Анализ стратегий и разработка комплексных программ.- М.: Наука, 1990.- 216с.
4. Аджиев В. Объектная ориентация: философия и футурология. Открытые системы, N 06 1996 г.
5. Аджиев В. MS: корпоративная культура разработки ПО. Открытые системы, N 01 1998 г.
6. Альпина Паблишер. Управление проектом по созданию интернет-сайта. Альпина Паблишер, 2001 г. 337 стр.
7. Андон Ф.И., Коваль Г.И., Коротун Т.М., Суслов В.Ю.; [Отв. ред. И.В.Сергиенко]. Основы инженерии качества программных систем / НАН Украины. Ин-т прогр. систем. - К.: Изд. дом "Академпериодика", 2002. - 502 с.: ил., табл.
8. Ансофф Игорь. Новая корпоративная стратегия. Серия: Теория и практика менеджмента. Издательство: Питер, 1999 г. 416 стр.
9. Армстронг М., Барон А. Performance Management. Управление эффективностью работы. Performance Management: The New Realities. Серия: Developing Practice. Издательство: Hippo, 2005 г.- 384 стр.
10. Арчибальд Р.Д. Управление высокотехнологичными программами и проектами / Пер. с анг. Мамонтова Е.В.; под ред. Баженова А.Д., Арсеньева О.А. ДМК Пресс; АйТи, 2004.- 463 с
11. Ауэр Кент, Миллер Рой. Экстремальное программирование: постановка процесса. С первых шагов и до победного конца Planning Extreme Programming. Серия: Библиотека программиста. Издательство: Питер, 2003 г. - 368 стр.
12. Ахмед Х.З. Разработка корпоративных Java приложений с помощью J2EE и UML/ Пер. с англ.А.В.Высоцкого.- М.: Вильямс, 2002.- 267 с.
13. Баранов С. Н. Модель зрелости способностей к разработке программных продуктов. \\ Программные продукты и системы № 4, 1998. с. 2-7
14. Бауэр Р. Управление инвестиционным проектом. Опыт ИБМ - М.: ИНФРА-М, 1995. — 208 с.
15. Джон Бентли. Жемчужины программирования. Издательство: Питер, 2002 г. - 272 стр.
16. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. Питер. 2004 г. – 320 с.
17. Богданов Д. В. Путилов В. А. Фильчаков В. В. Стандартизация процессов обеспечения качества программного обеспечения. — Апатиты, КФ ПетрГУ, 1988. — 152 с.
18. Бек К. Экстремальное программирование: планирование.- СПб: Питер, 2003.-

143 с.
19. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. Пер. с англ.- СПб: Питер, 2004.- 317 с.
20. Благодатских В.А., Волнин В.А., Поскакалов К.Ф. Стандартизация разработки программных средств. Учебное пособие. Издательство: Финансы и статистика, 2006 г.288 стр.
21. Брауде Э. Д. Технология разработки программного обеспечения. Питер. 2004 г. – 656 с.
22. Боэм Б. Инженерное проектирование программного обеспечения: Пер с англ. — М.: Радио и Связь, 1985. — 240 с.
23. Большая советская энциклопедия, «Советская энциклопедия» 1969 — 1978 гг.в 30 томах.
24. Боэм Б., Браун Дж., Каспар Х., Характеристики качества программного обеспечения. — М.: "Мир", 1981. — 208 с.
25. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. – Пер. с англ. – СПб.: Символ-Плюс, 1999. – 304 с.
26. Бугорский В.Н. Экономика и проектирование информационных систем. .- СПб.: Роза мира, 1998.- 340 с.
27. Буч Г. Объектно-ориентированное проектирование. — Киев, Москва, “Диалектика”, АО “ИВК” — 516 с.
28. Буч Г. Рамбо. Д. Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. ДМК, 2000. – 432 с.
29. Дм. Волков, А. Гавердовский, И. Косякин "Заметки о Российском Программировании", Открытые Системы, 1997, N. 3 (23), с. 78-80
30. Волкова В. Н. Чабровский В. А. Цель: Прогнозирование, анализ, структуризация. — М.: ИСЭП РАН, 1995 г. — 113 с.
31. Васкевич Д. Стратегии клиент/сервер. Руководство по выживанию для специалистов по реинженирингу корпораций. - 2-е изд. - Пер. с англ. - Киев: Диалектика, 1996 —396 с.
32. Вендров А. М. CASE — технологии. Современные методы и средства проектирования. М.: Фин. и Ст. 1998 г. — 176 с.
33. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник для вузов.- М.: Финансы и статистика, 2002.- 347 с.
34. Вигерс Карл И. Разработка требований к программному обеспечению. Русская Редакция, 2004 г. – 576 с.
35. Вирт Н. "Долой Жирные Программы", Открытые Системы, N. 6 (20), 1996, с. 27-31
36. Власова Л. Российский ИТ-аутсорсинг: предложение поспевает за спросом. Экономика и жизнь.- 2005. - N 18. - С. 35.
37. Воас Джеффри Качество ПО: восемь мифов Журнал "Открытые Системы", # 09-10/1999

38. Гантер Р. Методы управления проектированием программного обеспечения: Пер. с англ. — Мир, 1981. — 329 с.

39. Гацко Г. Тестирование ПО как один из элементов системы качества. Открытые системы № 06 1998 г.

40. Гейн К., Сарсон Т. Структурный системный анализ: средства и методы. -М: “Эйтекс”, 1992 - 274 с.

41. Генс Г. В. Концепция Business Performance Management: начало пути. Альпина Бизнес Букс, 2004 г. 270 стр.

42. Гласс Р. Сопровождение программного обеспечения. Пер. с англ. Г.В.Вышковского; Под ред.Ю.А.Чернышева.- М.: Мир, 1983.- 158 с.: ил.

43. Грей Клиффорд Ф., Ларсон Эрик У. Управление проектами. Практическое руководство. Издательство: Дело и Сервис (ДИС), 2002 г. - 528 с.

44. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений/ Пер. с англ.- М.: ДМК Пресс, 2002.- 698 с.

45. Единая система программной документации. ГОСТ 19.001. М.: ИПК Издательство стандартов , 1998 – 164с.

46. Международный стандарт ИСО 9000-3: 1991 Общее руководство качеством и стандарты по обеспечению качества. Часть 3: Руководящие указания по применению ИСО 9001 при разработке, поставке и обслуживании программного обеспечения.

47. Международный стандарт ИСО 9001 Системы Качества. Модель для обеспечения качества при проектировании и/или разработке, производстве монтаже и обслуживании.

48. Государственный стандарт Российской Федерации. ГОСТ 34.601-90 Автоматизированные Системы Стадии создания Комплекс стандартов на автоматизированные системы. ИПК издательство стандартов, М.: - 1997

49. Государственный стандарт Союза ССР. Оценка качества программных средств. Общие положения. ГОСТ 28195-89. Издательство стандартов, Москва. 1989 г. 39 с.

50. Государственный стандарт Российской Федерации. ГОСТ Р ИСО/МЭК 12207-99. Информационные технологии. Процессы жизненного цикла программных средств. Госстандарт России, Москва. 46 с.

51. Государственный стандарт Российской Федерации. ГОСТ Р ИСО/МЭК 9126-93 Информационные технологии. Оценка программной продукции. Характеристики качества. Руководства по их применению. Издательство стандартов, Москва. 19 с.

52. Государственный стандарт Российской Федерации. ГОСТ Р ИСО/МЭК 9294-93 Информационные технологии. Руководство по управлению документированием программного обеспечения, Издательство стандартов, Москва. 1994 г. 19 с.

53. Государственный стандарт Российской Федерации. ГОСТ Р ИСО/МЭК 12119-2000. Информационные технологии. Пакеты программ. Требования к качеству и тестирование. Издательство стандартов, Москва. 2001 г. 19 с.

54. Рекомендации по стандартизации. Р.50.1.028-2001. Информационные

технологии поддержки жизненного цикла продукта. Методология функционального моделирования. Издательство стандартов, Москва. 2001 г. 53 с.

55. Рекомендации по стандартизации. Р.50.1.031-2001. Информационные технологии поддержки жизненного цикла продукта. Терминологический словарь. Часть 1. Стадии жизненного цикла продукции. Издательство стандартов, Москва. 2001 г. 32 с.

56. Рекомендации по стандартизации. Р.50.1.032-2001. Информационные технологии поддержки жизненного цикла продукта. Терминологический словарь. Часть 2. Применение стандартов серии ГОСТ Р ИСО10303. Издательство стандартов, Москва. 2001 г. 10 с.

57. Рекомендации по стандартизации. Р.50.1.027-2001. Информационные технологии поддержки жизненного цикла продукта. Автоматизированный обмен технической информацией. Основные положения и общие требования. Издательство стандартов, Москва. 39 с.

58. Государственный стандарт Российской Федерации. ГОСТ Р ИСО 9127-94. Системы обработки данных. Документация пользователя и информация на упаковке для потребительских пакетов программ. Издательство стандартов, Москва. 1995 г. 18 с.

59. Государственный стандарт Российской Федерации. ГОСТ Р ИСО 9000-2001. Системы менеджмента качества ОСНОВНЫЕ ПОЛОЖЕНИЯ И СЛОВАРЬ.

60. Межгосударственный стандарт. ГОСТ 34.320-96. Информационные технологии. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы. Межгосударственный совет по стандартизации, метрологии и сертификации. Издательство стандартов Минск, 2001 г. 46 с.

61. Дастин Э., Рэшка Дж., Пол Д. Автоматизированное тестирование программного обеспечения Внедрение, управление и эксплуатация: Пер. с англ./.- М.: Лори, 2003.- 567 с

62. Дашевский А.И. Разработка и стандартизация программных средств и информационных технологий : Методические указания по изучению дисциплины и задания по выполнению контр. работ для студентов заоч. формы обучения. 16 с.

63. Деминг Э. Выход из кризиса. - Пер. с англ. - Тверь: Альба, 1994 г. — 498 с.

64. Домаратский А. Н. Управление улучшением стандартного процесса и качеством программных изделий. \\ Программные продукты и системы № 4, 1998. с. 20-24

65. Друкер П. Ф. Практика менеджмента. Пер. с англ.: - М.: Издательский дом «Вильямс», 2001. – 398 с.

66. Друкер П. Ф. Рынок: Как выйти в лидеры. Практика и принципы. М.: Бук Чембер Интернэшнл, 1992 г. – 350 с.

67. Деверадж С., Кохли Р.Окупаемость ИТ : Измерение отдачи от инвестиций в информационные технологии / М.: Новый издательский дом, 2005.- 178 с.

68. Джалота Панкаж. Управление программным проектом на практике. Software Project Management in Practice. Издательство: Лори, 2005 г. 224 стр.

69. Джерк Н. Разработка приложений для электронной коммерции/ Пер. с англ.

Е.Матвеева.- СПб.: Питер, 2001.- 506 с.
70. Еременко А. Разработка бизнес-приложений в Microsoft Business Solutions - Axapta версии 3.0/ Еременко А., Шашков Р.- М.: Альпина Бизнес Букс, 2005.- 502 с.: ил.+ CD-ROM
71. Ермошкин Н., А. Тарасов. Стратегия информационных технологий предприятия. Как Cisco Systems и ведущие компании мира используют Интернет Решения для бизнеса. Издательство: Московский гуманитарный университет, 2003 г. 360 стр.
72. Жарков С.В. Shareware: профессиональная разработка и продвижение программ.- СПб: БХВ-Петербург, 2002.- 317 с.
73. Желены М. Информационные технологии в бизнесе. СПб: Питер, 2002. – 1120 с.: ил. (Серия «Бизнес-класс»
74. Жуков В. И. Путилов В. А. Фильчиков В. В. Объектно-ориентированная технология разработки программных систем. — Апатиты, КФ Петр ГУ, 1988. — 96 с.
75. Забавин В. От локального - к глобальному: Опыт реализации ИТ-проектов на отечественных предприятиях. //. Бухгалтер и компьютер.- 2003. - N 5. - С. 16 - 17.- (1561-4492)
76. Закис Алексей. RUP и другие методологии разработки ПО. http://www.shmakov.ru/news/text136.html
77. Зиндер Е. З. Соотнесение и использование стандартов организации жизненных циклов систем. Системы Управления Базами Данных · # 3/97 · стр. 41-53
78. Иванов И. В., Баранов В. В., Лысак Г. И., Кирсанов О. В. Высокотехнологичные предприятия в эпоху глобализации. Альпина Паблишер, 2003 г. 416 стр.
79. Иванов М. А., Шустерман Д. М. Организация как ваш инструмент. Российский менталитет и практика бизнеса. Альпина Бизнес Букс, 2004 г. 400 стр.
80. Ивлев В. А., Попова Т. В. ABIS. Информационные системы на основе действий. Издательство: 1С-Паблишинг, 2005 г. - 248 с.
81. Игнатьев М.Б., Фильчаков В.В., Осовецкий Л.Г. Активные методы обеспечения надежности алгоритмов и программ. – СПб: Политехника, 1992 г. - 170 с.
82. Исикава К. Японские методы управления качеством. М., 1988. — 250 с.
83. Йордон Э. Структурное проектирование и конструирование программ./ Пер. с англ. Фролова В.В.,Теплицкого Л.А.;Под ред. Королева Л.Н.- М.: Мир, 1979.- 415 с.
84. Йордон Э. Путь камикадзе. Как разработчику программного обеспечения выжить в безнадежном проекте. Издательство «Лори», 2001 – 256 с.
85. Йордон. Э. Управление сложными интернет-проектами. Издательство Лори, 2003 г. – 344 стр.
86. Йордон Э., Аргила К. Структурные модели в объектно-ориентированном анализе и проектировании М.: Лори, 1999 г. — 266 стр.

87. Калбертсон Роберт, Браун Крис, Кобб Гэри. Быстрое тестирование. Издательство: Вильямс, 2002 г. - 384 стр

88. Кале Вивек Внедрение SAP R/3. Руководство для менеджеров и инженеров. Компания АйТи, 2004 г. 470 стр.

89. Камер Д.Э. Компьютерные сети и Internet: Разработка приложений для Internet: пер. с англ.- 3-е изд.- М.: Вильямс, 2002.- 639 с.: ил.+ CD-ROM.

90. Кантор Марри. Управление программными проектами. Практическое руководство по разработке успешного программного обеспечения. Издательство: Вильямс, 2002 г. - 176 стр.

91. Каплан Роберт, Нортон Дейвид Стратегические карты. Трансформация нематериальных активов в материальные результаты. Олимп-бизнес, 2004 г. – 512 с.

92. Керниган Брайан, Пайк Роб. Практика программирования. Вильямс, 2004 г. 288 с.

93. Кармайкл Энди, Дэн Хейвуд. Быстрая и качественная разработка программного обеспечения. Вильямс, 2003 г. - 400 стр.

94. Керцнер Гарольд. Стратегическое планирование для управления проектами с использованием модели зрелости. Издательства: ДМК пресс, Компания АйТи, 2003 г. 320 стр.

95. Карминский А.М., Карминский С.А., Нестеров В.П., Черников Б.В. Информатизация бизнеса: концепции, технологии, системы/.- 2-е изд., перераб. и доп.- М.: Финансы и статистика, 2004.- 621 с.

96. Кастельс М. Информационная эпоха: экономика, общество и культура. ГУ ВШЭ; Пер. с англ. под ред. О.И.Шкаратана.- М.: ГУ ВШЭ, 2000.- 607 с.: ил.

97. Ковени Майкл, Гэнстер Деннис, Хартлен Брайан, Кинг Дейв. Стратегический разрыв. Технологии воплощения корпоративной стратегии в жизнь. Альпина Бизнес Букс, 2004 г. 240 стр.

98. Колесов А. Кто поддержит российских производителей программного обеспечения? "Известия" от 26.06.99 — с.5

99. Коллинз Д. Структурный анализ в разработке систем. - М: Финансы и статистика, 1986 - 280 с.

100. Колтунова Е. В Концепция единой архитектуры приложений электронного правительства. ВКСС.Connect №4 2005 г.

101. Колтунова Е. В. Управление проектированием и разработкой информационных систем. Тезисы докладов Весенних семинаров молодых ученых – экономистов’ 98. – СПб.: Издательство С. – Петербургского университета, 1999 – стр. 6-7.

102. Колтунова Е. В. Управление затратами с использованием системы потоков работ. Сборник научных трудов выпуск 3. с. 73-78 Моделирование экономических процессов и структур. СПбГИУ, 2005 г.

103. Колтунова Е. В. Выбор информационных систем. Журнал Управление компанией, Июль, 2004 г.

104. Колтунова Е. В. Как оценить стоимость проекта автоматизации?

http://www.nsda.ru/home.asp?artId=2141
105. Колтунова Е. В. Требования к информационной системе и модели жизненного цикла. http://www.nsda.ru/home.asp?artId=2142
106. Колтунова Е. В. Управление информационными проектами: модель зрелости технологических процессов / Менеджмент и экономика в творчестве молодых исследователей тезисы докладов научно практической конференции студентов и спирантов СПбГИЭА 20-21 апреля 1999 года.
107. Колтунова Е. В. Методы и модели управления проектами / Бюллетень семинар молодых экономистов Выпуск 7 1999 г.
108. Колтунова Е. В Классификация методологий, моделей и стандартов управления разработкой программным обеспечением для решения задачи их адекватного выбора. Математическое моделирование в экономике. Сборник научных трудов. с. 267-276 СПбГИЭУ, 2006 г – 320 с.
109. Еникеева Л.А, Колтунова Е.В. Особенности моделирования финансовых потоков инвестиционных проектов на стадии их реализации Сб. науч. тр. – СПбГИЭА, 1999 г.
110. Колтунова Е.В. Выбор информационных систем. Журнал Управление компанией, Июль, 2004 г
111. Качаева Е. В.,. Еникеева Л. А, Колтунова Е. В. Реинжиниринг коммерческих предприятий.: Метод. указ. к лаб. работам для студ. спец. 061400 Коммерция СПбГИЭА.- Спб., 1999. - 25 с.
112. Коберн А. Современные методы описания функциональных требований к системам. М: Лори, 2002 г. – 264 с.
113. Коберн А. Каждому проекту своя методология. Humans and Technology Technical Report, TR 99.04, Oct.1999 7691 Dell Rd, Salt Lake City, UT 84121 USA
114. Коберн А. Быстрая разработка программного обеспечения. М: Лори, 2002 г. – 314 с.
115. Коленсо Майкл. Стратегия кайзен для успешных перемен в организации. Серия: Менеджмент для лидера. Издательство: Инфра-М, 2002 г. 178 стр.
116. Константайн Л. Человеческий фактор в программировании. Символ-Плюс, 2004 г. – 384 с.
117. Константайн Л., Локвуд Л. Разработка программного обеспечения. Питер. 2004 г. – 592 с.
118. Костина Л. П. Метод критического пути в многопроектных разработках с учетом ресурсов. // ЭиММ, 1998 Т. 34 №3
119. Коуд П., Норт Д., Мэйфилд М. Объектные модели. Стратегии, шаблоны и приложения Лори, 1999 г. - 430 стр.
120. Коул Дж., Горэм Т. Принципы тестирования ПО // Открытые системы. № 2 1998 г.
121. Кох Ричард. Стратегия. Как создавать и использовать эффективную стратегию. Серия: Теория и практика менеджмента. Издательство: Питер, 2003 г. 320 стр.
122. Крачтен Филипп. Введение в Rational Unified Process. Вильямс, 2002 г. - 240

стр.
123. Керцнер Г. Стратегическое планирование для управления проектами с использованием модели зрелости / Пер. с англ. /Под ред. А.Д.Баженова.- М.: Финансы и статистика, 2003.- 320 с.
124. Крайер Э. Успешная сертификация на соответствие нормам ИСО серии 9000. Руководство по подготовке и проведению сертификации, дальнейшие шаги. Пер с нем. М.: ИздАт, 1999 – 550 с.
125. Кречетов Н.Технологии оценки качества программных продуктов, ComputerWeek №25 1996 — с.27-32,42
126. Кролл Пер, Крачтен Филипп Rational Unified Process - это легко. Руководство по RUP для практиков. Издательство: КУДИЦ-Образ, 2004 г. 432 стр.
127. Купер Алан. Психбольница в руках пациентов. Издательство: Символ-Плюс, 2005 г. - 336 стр.
128. Кунде Йеспер. Корпоративная религия. Издательство: Стокгольмская школа экономики в Санкт-Петербурге, 2002 г. 272 стр.
129. Кьоу Дж. Объектно-ориентированное проектирование: Учебный курс/ Дж.Кьоу , М.Джеанини.- СПб: Питер, 2005.- 237 с.
130. Леффингуэлл Дин, Уидриг Дон Принципы работы с требованиями к программному обеспечению. Унифицированный подход. Издательство: Вильямс, 2002 г. 448 стр.
131. Липаев В.В. Выбор и оценивание характеристик качества программных средств: Методы и стандарты/ Моск.госуд.технологич.ин-т "Станкин".- М.: Синтег, 2001.- 224 с.
132. Липаев В. В. Качество программного обеспечения//Финансы и статистика, М., 1983.
133. Липаев В. В. Проектирование программных средств. М.: Высш. шк., 1990. – 330 с.
134. Липаев В. В. Тестирование программ. — М.: Радио и связь, 1986. — 296 с.
135. Липаев В. В. Управление разработкой ПС: методы, стандарты, технология. М.: Фин. и Ст. 1993 г. — 160 с.
136. Липаев В. В. Филинов Е. Н. Мобильность программ и данных в открытых информационных системах. М.: РФФИ: Науч. Кн., 1997 г. — 361 с.
137. Лихачева Г.Н. Информационные технологии в экономике: Учебно-методическое пособие, руководство по изучению дисциплины, применение информационных технологий в экономике и управлении, тесты, учебная программа. Лихачева Г.Н. и др.- М.: Московский гос. университет экономики, статистики и информатики, 2004.- 184 с.
138. Лодон Дж., Лодон К. Управление информационными системами. 7-е изд. Питер, 2005 г. – 912 с.
139. Мартин Р.К. Ньюкирк Дж.В., Косс Р.С Быстрая разработка программ. Принципы, примеры, практика/ Мартин Р.К.,.; Пер. с англ.А.П.Сергеева, Т.А.Шамренко.- М.: Вильямс, 2004.- 743 с.

140. Мак-Карти Джим и Мишель. Программируем командный дух.: Символ-Плюс, 2004 г. - 416 с.

141. Макконнелл С. Совершенный код. Практическое руководство по разработке программного обеспечения. Питер. 2005 г. – 896 с.

142. Марка Д. А. МакГоуэн К. Методология структурного анализа и проектирования. — М.: МетаТехнология, 1993, 240 с.

143. Мартин Роберт К., Ньюкирк Джеймс В., Косс Роберт С. Быстрая разработка программ. Принципы, примеры, практика. Вильямс, 2004 г. - 752 стр.

144. Марченко Е. Что такое качество программного обеспечения? http://www.silicontaiga.ru/home.asp?artId=2862

145. Мацяшек Л.А. Анализ требований и проектирование систем: Разработка информационных систем с использованием UML: Пер. с англ.- М.: Вильямс, 2002.- 428 с.

146. Мощелла Д. Бизнес-перспективы информационных технологий. Альпина Бизнес Букс, Москва 2004 г., 252 с.

147. Ньюкомер Э. Веб-сервисы. Для профессионалов. Питер. 2003 г. – 256 с.

148. Одинцов И.О. Профессиональное программирование. Системный подход.- СПб: БХВ-Петербург, 2002.- 510 с.

149. Орлов С.А. Технологии разработки программного обеспечения: Разработка сложных программных систем: Учебное пособие.- СПб: Питер, 2002.- 463 с.: ил.- (Учебник для вузов)

150. Ойхман Е. Г. Попов Э. В. Реинжиниринг бузнеса: реинжиниринг организации и информационные технологии. — М.: Финансы и статистика, 1997. — 336 с.

151. Палмер Стивен Р., Фелсинг Джон М. Практическое руководство по функционально-ориентированной разработке ПО. Издательство: Вильямс, 2002 г. - 304 стр.

152. Питеркин С.В., Оладов Н.А., Исаев Д.В.- Точно вовремя для России: Практика применения ERP-систем/ М.: Альпина Паблишер, 2002.- 365 с.

153. Питерс Т.Дж. Р.Х.Уотерман-мл В поисках совершенства: Уроки самых успешных компаний Америки М.: Вильямс, 2005.- 558 с

154. Полковников А.В., Управление проектами - выбор, внедрение и использование ПО в России, PC Week\RU, 1996. № 34-35

155. Полукаев О. Коваль Д. Моделирование бизнеса и архитектуры информационных систем. СУБД № 4 1995 с. 81-94

156. Поттосин И. "Хорошая программа": попытка точного определения понятия //Программирование. — 1997 № 2. — с. 3-17.

157. Пратт Т. Языки программирования: Разработка и реализация/ Под ред. А.Матросова.- 4-е изд.- СПб: Питер, 2002.- 688 с.

158. Пржиялковский В.В. Сложный анализ данных большого объема: новые перспективы компьютеризации // СУБД. — 1996 г. № 4. с.71-83.

159. Причад Дж. COM и CORBA просто и доступно. М.: Лори, 2001. – 372 с.

160. Рейнвотер Дж. Ханк. Как пасти котов. Наставление для программистов, руководящих другими программистами Издательство: Питер, 2006 г. - 256 стр.

161. Реймана Л.Д. Волокитин А.В., Панкратов. Интернет-технологии в Федеральной целевой программе "Электронная Россия (2002-2010 годы)": Справочное пособие/ М.: Изд-во "Известия"; НТЦ "Фиорд-Инфо", 2003.- 271 с.

162. Ройс У. Управление проектами по созданию программного обеспечения. Унифицированный подход. М.: Лори, 2002. – 434 с.

163. Ротер М. Шук Дж. Учитесь видеть бизнес-процессы. Практика построения карт потоков создания ценности. Альпина Бизнес Букс, CBSD. Центр развития деловых навыков, 2005 г. - 144 стр.

164. Рыбков Н. Шкала качества. 'Инфобизнес' № 44 от 20 ноября 2001

165. Рынок труда в сфере IT http://www.silicontaiga.ru/home.asp?artId=4600 Работа для вас N 60 (1556) от 24 апреля 2006

166. Руководство к Своду знаний по управлению проектами Третье издание (Руководство PMBOK®) Американский национальный стандарт ANSI/PMI 99-001-2004. 401 с.

167. Салливан Эд. Время – деньги. Создание команды разработчиков программного обеспечения. Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2002 – 368 стр.

168. Сенге Питер М., Клейнер Арт, Робертс Шарлота, Росс Ричард Б., Рот Джордж, Смит Брайан Дж. Танец перемен: новые проблемы самообучающихся организаций. Пер с англ. – М.: ЗАО «Олимп-Бизнес», 2004 г. – 624 с.

169. Сигел Дэвид. Шагни в будущее. Стратегия в эпоху электронного бизнеса. Издательство: Олимп-Бизнес, 2001 г. 384 стр.

170. Синтес А. Освой самостоятельно объектно-ориентированное программирование за 21 день : Пер. с англ. - М.: Вильямс, 2002.- 671 с.: ил.

171. Скотт Ф. Уилсон и др. Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD. Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2000. – 608 с.

172. Смит Конни, Уильямс Ллойд. Эффективные решения. Практическое руководство по созданию гибкого и масштабируемого программного обеспечения. Вильямс, 2003 г. - 448 стр.

173. Советов Б. Я. Яковлев С. А. Моделирование систем. — М.: Высш. шк., 1985 г. — 271 с.

174. Соколов Р.В. Разработка диаграмм потоков данных с использованием CASE-технологии : Методич. указ. к практич. занятиям и лаборатор. работе по дисциплине «Проектирование информационных систем в экономике»/ СПб: СПбГИЭА, 2001.- 18 с.

175. Соммервилл Иан Инженерия программного обеспечения. Издательство: Вильямс, 2002 г. - 624 стр.

176. Спинеллис Диомидис. Анализ программного кода на примере проектов Open Source. Вильямс, 2004 г. – 528 с.

177. Спирли Э. Корпоративные хранилища данных. Планирование, разработка,

реализация. Пер. с англ. и ред. В.М.Неумоина.- М.: Вильямс Т.1.- 2001.- 396 с.: ил.
178. Спольски Дж. Джоэл о программировании. Joel on Software. Серия: Профессионально. Издательство: Символ-Плюс, 2006 г.- 352 стр.
179. Стокгольмская Школа Экономики в России. Комплексное исследование. Влияние ИТ-технологий на Российскую экономику. Июнь, 2007 г. 171 стр. Microsoft
180. Стивенс Р. Тестирование и отладка программ на Visual Basic. Пер. с англ. – М.: ДМК Прогресс, 2001. – 384 с., ил. (Серия «Для программистов»)
181. Сухарев А.Г. Подготовка ИТ-специалистов в России. Мир ПК.- 2004. - N 10. - С. 99 - 101
182. Тассел Ван Д. Стиль, разработка, эффективность, отладка и испытание программ: Пер. с англ. — М.: Мир, 1985. — 332 с.
183. ДеМарко Том, Листер Тимоти. Вальсируя с Медведями: управление рисками в проектах по разработке программного обеспечения. Издательство: Компания p.m.Office, 2005 г. - 196 стр.
184. ДеМарко Том, Листер Тимоти. Человеческий фактор: успешные проекты и команды. Издательство: Символ-Плюс, 2005 г. - 256 стр.
185. Томсетт Р. Радикальное управление ИТ-проектами. Лори, 2005 г. 294 стр.
186. Торрес Р. Дж. Практическое руководство по проектированию и разработке пользовательского интерфейса. Вильямс, 2002 г. 400 стр.
187. Уайт Б.А. Управление конфигурацией программных средств : Практическое руководство по Rational ClearCase М.: ДМК Пресс, 2002.- 265 с.
188. Фарсон Р. Менеджмент абсурда. Парадоксы лидерства. Пер. с англ. Летвинский. – К.: «София», 201. – 240 с.
189. Фатрелл Р. Т., Шафер Д. Ф., Шафер Л. И. Управление программными проектами: достижение оптимального качества при минимуме затрат. Вильямс 2003. - 1136 с.
190. Фаулер М. Новые методологии программирования http://www.silicontaiga.ru/home.asp?artId=4889
191. Фаулер М. Архитектура корпоративных программных приложений. Вильямс, 2004 г. – 533 с.
192. Фаулер М. Рефакторинг: улучшение существующего кода / Пер. с англ.С.Маккавеева - СПб: Символ-Плюс, 2004.- 430 с.: ил.
193. Федеральная целевая программа "Электронная Россия (2002-2010 годы) http://www.e-rus.ru/site.shtml?id=15
194. Фролов А.В. Визуальное проектирование приложений C#/ Фролов А.В., Фролов Г.В.- М.: КУДИЦ-ОБРАЗ, 2003.- 511 с.
195. Филипс Джозеф. Менеджмент ИТ-проектов. Лори, 2005 г. 378 стр.
196. Фокс Дж. Программное обеспечение и его разработка/ Пер. с англ. Карпова Л.Е. под ред. Подшивалова Д.Б.- М.: Мир, 1985.- 368 с.
197. Хаммер М., Чампи Дж. Реинжениринг корпорации: Манифест революции в бизнесе. Пер. с англ. /Под ред. и с предисл. В.С.Катькало. - СПб.: Издательство С. - Петербургского университета, 1997. — 332 с

198. Хэзфилд Р., Кирби Л. и др.Искусство программирования на С: Фундаментальные алгоритмы, структуры данных и примеры приложений: Энциклопедия программиста: Пер. с англ./ Киев: ДиаСофт, 2002.- 484 с.: ил.

199. Ховард Р. А. Динамическое программирование и марковские процессы. — М.: Сов. Радио, 1964 г. — 190 с.

200. Хулей Грэм, Сондерс Джон, Пирси Найджел. Маркетинговая стратегия и конкурентное позиционирование. Издательство: Баланс Бизнес Букс, 2005 г. 778 стр.

201. Цимбал Александр, Аншина Марина. Технологии создания распределенных Издательство: Питер, 2003 г. - 576 стр.

202. Чарнецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. Для профессионалов. Питер. 2005 год – 736 стр.

203. Шапиро В. Д. и др. Управление проектами (Зарубежный опыт). — СПб.: "ДваТри", 1993 г. — 443 с.

204. Шапиро В. Д. и др. Управление проектами. — СПб.: "ДваТри", 1996 г. — 610 с.

205. Шейн Э. Организационная культура и лидерство. Построение. Эволюция. Совершенствование Издательство: Питер, 2002 г., 336 стр

206. Шелест В.Д. Программирование.- СПб: БХВ-Петербург, 2002.- 584 с.

207. Alexander C., Ishikawa S., Silverstein M., Jacobson M., Fiksdahl-King I., Angel S. A Pattern Language. Oxford University Press, New York, 1977

208. Boehm B., "A Spiral Model of Software Development and Enhancement", Proc. Int'l Workshop Software Process and Software Environments, ACM Press, 1985

209. Boehm B. COCOMO II Model Definition Manual, Center for Software Engineering, 1998 - 37p.

210. G.Booch "The End of Objects and the Last Programmer", - htpp://www.rational.com/pst/tech_papers

211. Booch G. "Objects Solutions", - Addison-Wesley, 1996

212. Booch G., Rumbaugh, Jacobson I. The Unified Modeling Language User Guide. – Addison-Wesley, 1999 – 482 p.

213. Cassidy Anita. A Practical Guide to Information Systems Strategic Planning CRC Press LLC, 1998 NW -

214. Capability Maturity Model® Integration (CMMI SM), Version 1.1 Continuous Representation CMU/SEI-2002-TR-011 ESC-TR-2002-011 CMMI Product Team Copyright 2002 by Carnegie Mellon University. 725 p.

215. Cockburn A. Characterizing People as Non-Linear, First-Order Components in Software Development. http://alistair.cockburn.us/crystal/articles/cpanfocisd/characterizingpeopleasnonlinear.html

216. Cockburn A. Humans and Technology Humans and Technology Technical Report, TR 99.04, Oct.1999 7691 Dell Rd, Salt Lake City, UT 84121 USA

217. Crosby P. B. Quality Is Still Free: Making Quality Certain In Uncertain Times. McGraw-Hill: 1979. — 225 p.

218. CobiT 4.1. IT Governance Institute, 2007 – 213 p.
219. Cusumano Michael A. The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad Free Press, 2004 – 352 p.
220. Curran T. Gerhard K. SAP R/3 Business blueprint. —Prentice Hall, 1998. — 287 p.
221. Curtis W. et al., "On Building Software Process Models under the Lamppost", Proc. Int'l Conf. Software Eng., IEEE CS Press, 1987.
222. Davenport, T.H. & Short, J.E. "The New Industrial Engineering: Information Technology and Business Process Redesign," Sloan Management Review, 1990, Summer pp. 11-27
223. Davis A. "Fifteen Principles of Software Engineering", - IEEE Software, Vol. 11, N.6, 1994, pp.94-101
224. DeMarco T. Controlling Software Projects. — Englewoods Clifs. Prentice Hall, 1985. — 284 p.
225. Dennis A. Systems Analysis Design/ Dennis A., Wixom B.- 2rd edition: Wiley, 2003.- 537 p.
226. Dijkstra E., Programming Considered as a Human Activity, Proceeding of the 1965 IFIP Congress, Amsterdam, The Netherlands: North Holland Publishing, pp. 101
227. Dijkstra E., "Go To Statement Considered Harmful", Comm. ACM, Mar. 1968
228. Introduction of Experience ® Pro http://www.sttf.fi/eng/products/experience/indexExperience.htm
229. Framework for Managing Process Improvement, Department of Defense: 1994. — 329 p.
230. Gilb T., "Evolutionary Development," ACM Software Eng. Notes, Apr. 1981, p. 17
231. Gilb T., "Evolutionary Delivery versus the 'Waterfall Model'," ACM Software Requirements Eng. Notes, July 1985
232. Gerland L. Moeller, Lester D. Dogman Operations planning with VERT // Operations Research Vol. 29 № 4 1981 p. 676-679
233. Goldberg M. S., Weber A. Ch. Evaluation of the Risk Analysis and Cost Management (RACM) Model. — Virginia. Institute for defense analyses, 1998 г. — 231 с.
234. Golenko-Ginsburg D. Gonik A. On-line control for network construction projects. // JORS, 1997 P.175-183
235. Gordon S., Gordon J Information Systems: Management Approach. Wiley, 2004.- 429 p.
236. Hamel G, Prahalad C. K. Competing for the future. Boston: Harvard Business Scholl Press, 1994
237. Handy Charles Gods of Management: The Changing Work of Organizations Oxford University Press, 1996 г 272 стр.
238. Hohmann L. "Journey of the Software Professional", - Prentice Hall, 1996
239. Hon-Shilang Lau, Amy Hing-Ling Lau. An Improved Approach for Estimating the

Mean and Standard Deviation of a Subjective Probability Distribution // Journal of Forecasting, Vol. 16, 1997 — 83 - 95 p.

240. Howick S. Eden C/ The impact of disruption and delay when compressing large projects: Going for incentives? Journal of Operations Research Society. (2001) 52, 24-36

241. Humphrey W. S. The Personal Process in Software Engineering, A personal Commitment to Software Quality. — SEI, 1994. —29 p.

242. Watts S. Humphrey “The Personal Process in Software Engineering”, Copyright (c) 1994 Institute of Electrical and Electronics Engineers. Presented at the Third International Conference on the Software Process, Reston, Virginia, October 10-11, 1994, pp 69-77

243. IEEE Std 1002, Taxonomy of Software Engineering Standards (1986) http://ieeexplore.ieee.org

244. ISO/IEC 9126 : Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use – 1991.

245. ISO/IEC TR 15504 SPICE Consolidated product. Software Process Assessment. Version 1.0 Technical Report.

246. ISO/IEC 19501:2005 Information technology -- Open Distributed Processing -- Unified Modeling Language (UML) Version 1.4.2

247. Juran J. M. Quality Planning and Analysis: From Product Development Through Use. McGraw-Hill Series in Industrial Engineering and Management Science, 1993. — 634 p.

248. Kaplan R. S. Norton D. P. The Balanced scorecard Harvard Business School Press Boston, Massachusetts, 1996 – 322 p.

249. Kitchenham, Barbara, Pfleeger, Shari Lawrence, Software Quality: The Elusive Target, IEEE Software, Vol. 13, No. 1, January 1996, pp. 12-21.

250. Krafzig D. Banke K. Slama D. Enterprise SOA Service-Oriented Architecture Best Practice. Prentice Hall Professional Technical Reference. Pearson Education, Inc, 2005 г. – 408 с.

251. Lehman M.M., "The Programming Process", internal IBM report, 1969; reprinted in Program Evolution

252. Lewis T. "Software Architectures: Divine plan or digital Darwinism", - Computer, 8/96, pp. 13-15

253. Lucas H. Information Technology: Strategic Decision Making for Managers: Wiley, 2005.- 446 p.

254. Masuda Y. Information Society as Postindustrial Society/ Wash/ World Future Soc., 1983.

255. McKinsey Global Institute, High Tech Practice, Business Technology office. How IT Enables Productivity Growth, San Francisco, 2002, 212 p.

256. Manifesto for Agile Software Development http://agilemanifesto.org/

257. Martin J. Odell J. Object-oriented methods Pragmatic Considerations. — Printice-Hall, 1996 —559 p.

258. McCabe Thomas J. Watson Arthur H. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric NIST Special Publication 500-235 Computer

Systems Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-0001 September 1996 - 124 p.

259. MacCormack A., "Product-Development Practices That Work", MIT Sloan Management Rev., vol. 42, no. 2, 2001

260. McFeeley Bob. IDEALSM: A User's Guide for Software Process Improvement CMU/SEI-96-HB-001, February 1996 236 p.

261. McLeod R., Jordan E. Systems Development: A Project Management Approach. Wiley, 2002.- 524 p.

262. McHale James Wall Daniel S. Mapping TSP to CMMI CMU/SEI-2004-TR-014 ESC-TR-2004-014 April 2005, Engineering Process Management, 209 p.

263. MK II Functional Point Analysis Counting Practices Manual, Version 1.3.1. United Kingdom Software Metrics Association, 1998. — 92 p.

264. Orr K. Data Quality and Systems Theory. — The Ken Orr Institute.— 1997. 6 p.

265. Park E. Robert, Webb J. Todd Software Cost and Schedule Estimating: A Process Improvement Initiative CMU/SEI-94-SR-3 May 1994 – 106 p

266. Paulk M. C., Cutis B., Chriss M. B., Weber Ch. V. Capability Maturity Model for Software, Version 1.1 CMU SEI, 1993. — 533 p.

267. Paulish Daniel J. Architecture-Centric Software Project Management: A Practical Guide. Addison-Wesley 2002. Paperback, 320 pages.

268. Perry, C. and Greig, I. Estimating the mean and variance of subjective distributions in PERT and decision analysis // Mgmt. Sci. 1975Vol. 21 P. 1477-1480.

269. Porter M. Competitive Strategy: Techniques for Analyzing Industries and Competitors New York: free Press, 1980

270. Randell B., Zurcher F.W., "Iterative Multi-Level Modeling: A Methodology for Computer System Design", Proc. IFIP, IEEE CS Press, 1968.

271. Raharadja C. P. A comparative evaluation of capability maturity model and ISO 9000. Embry-Riddle Aeronautic University Daytona Beach, Florida April 1996

272. Reeves Jack W. What is Software Design? http://www.bleading-edge.com/Publications/C++Journal/Cpjour2.htm ©C++ Journal - 1992

273. Rosenberg L. H. Hyatt L. Developing successful metrics program. — Unisys Government Systems, 1997 — 9 p.

274. Rosenberg L. H. Hyatt L. Software Quality Metrics for Object-Oriented Environments. — Unisys Government Systems, 1996 — 20 p.

275. Royce W., "Managing the Development of Large Software Systems", Proc. Westcon, IEEE CS Press, 1970

276. Russel A. H. Cash flows in networks. // Mgmt. Sci. 1970 Vol. 16 P. 357-373.

277. Schwaber Ken Agile Project Management with Scrum, Microsoft Press 2004 – 163 p.

278. Sisti F., Joseph S. Software Risk Evaluation Method, Version 1.0, SEI 1994. — 141 p.

279. Shewhart W., Statistical Method from the Viewpoint of Quality Control, Dover, 1986 (reprint from 1939).

280. SCAMPI SM, V1.0 Standard CMMISM Assessment Method for Process

Improvement: Method Description, Version 1.0 CMMI CMU/SEI-2000-TR-009 Product Development Team October 2000

281. Smith J. E. Nau R.F. Valuing Risky Projects. Option Pricing Theory and Decision Analysis. // Mgmt. Sci. 1995 Vol. 41 P. 795-816.

282. The Standish Group. The Chaos Report 1994. http://www.standishgroup.com/sample_research/chaos_1994_1.php

283. Strum J. Visual Basic project management. Wrox Press, 1999 - 550 p.

284. Takeuchi H. , Nonaka, I. "The New New Product Development Game", Harvard Business Rev., Jan. 1986

285. Talbot B.F. Resource-constrained project scheduling with time-resource tradeoff: The nonpreemptive case. // Mgmt. Sci. 1982 Vol. 28 P. 1197-1210.

286. Tarek K. Abdel-Hamid, Stuart Madnik. Software Project Dynamics. Englewood Clifs, NJ: Prentice-Hall, 1993 – 346 p.

287. Thomsett Rob. Double Dummi Spit and other Estimation Games. American Programmer, June 1996.

288. Troutt M. D. On Generality of the PERT average time formula. // Decision Sci. 1989 Vol. 20 P. 410-412

289. UK Department of Trade and Industry and British Computer Society. TickIT: A Guide to Software Quality Management System Construction and Certification Usage EN29001. 1992.

290. Yourdon Ed. Managing the System of Life Cycle: A Software Development Methodology Overview, New York, Yourdon Press, 1982

291. Weinberg Gerald M. An Introduction to General Systems Thinking. Dorset House Publishing Company, Incorporated; 2001 - 320 pages

292. Weinberg Gerald M. Quality Software Management: Systems Thinking (Quality Software Management). Dorset House Publishing Company, 1991 - 336 pages

293. Yordon Ed. Decline and fall of the American Programmer. — Yordon Press.: 1993. —352 p.

4. ПРИЛОЖЕНИЯ

4.1. Управление проектами: определения и структура

4.1.1. Определения понятия «проект»

Проект определяется как процесс, предприятие, намерение, усилие и действие. Под проектом понимается комплекс работ, направленных на достижение запланированной и, как правило, уникальной цели. Проект включает в себя замысел (проблему), средства его реализации (решения проблемы) и получаемые в процессе реализации результаты.

Существующие стандарты и методологии определяют **«проект»** как:

- Уникальный процесс, состоящий из набора взаимоувязанных и контролируемых работ с датами начала и окончания, предпринятый, чтобы достичь цели соответствия конкретным требованиям, учитывая ограничения по времени, затратам и ресурсам [ISO/TR 10006: 1997 (E) Quality Management – Guidelines to quality in project Management];
- Процесс достижения поставленной цели в рамках особого, конкретного комплекса условий [ISO 9000:2000 Quality Management Systems – Fundamental and Vocabulary].
- Предприятие (намерение), которое в значительной степени характеризуется неповторимостью условий в их совокупности, например: задание цели; временные, финансовые, людские и другие ограничения; разграничения от других намерений; специфическая для проекта организация его осуществления [DIN 69901. Германия, 1987];
- Временное усилие (действие), предпринятое для создания уникального продукта или услуги [A Guide to the Project Management Body of Knowledge. [PMI Standards Committee. 2000 Edition., 2000]

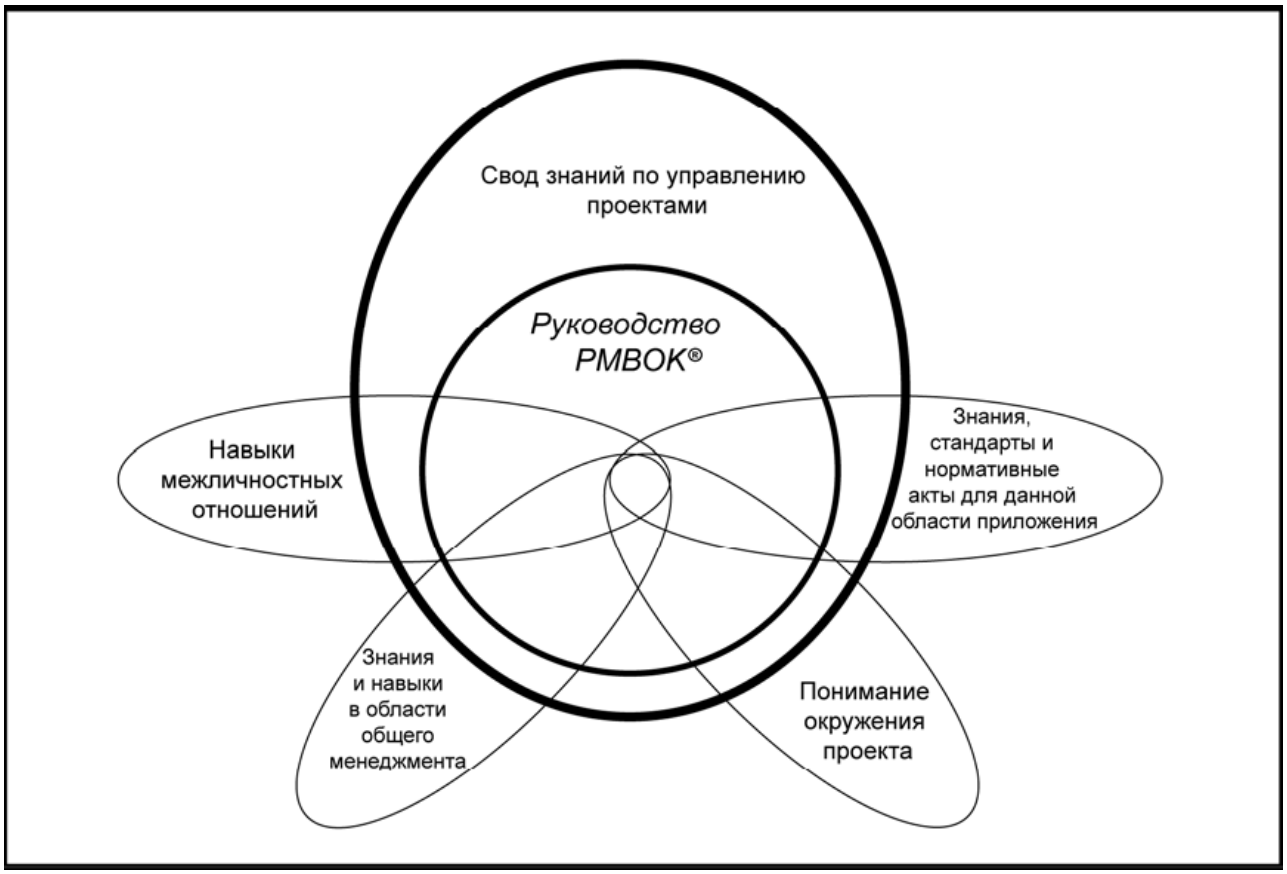
4.1.2. Состав деятельности по управлению проектами PMBok

В управление проектом входит:

- Определение требований
- Установка четких и достижимых целей
- Уравновешивание противоречащих требований по качеству, содержанию,
- времени и стоимости
- Коррекция характеристик, планов и подхода в соответствии с мнением и

- ожиданиями различных участников проекта.

4.1.3. Экспертные области, необходимые для команды управления проектом (PMBOK)



4.1.4. Терминология проектной методологии PRINCE

Business case – сценарий бизнеса

Organization - организация

Plans - план

Controls – объекты контроля

Stages – стадии, этапы проекта

Management of risk – управление риском

Quality in a project environment – управление качеством в границах проекта

Configuration management – урпавление конфигруацией

Change control – управление изменениями

4.1.5. Процессы управления проектами CMMI

Project Planning – планирование проекта

Project Monitoring and Control –мониторинг и контроль проекта

Supplier Agreement Management – управление договорами с поставщиками

Integrated Project Management for IPPD (or Integrated Project Management) – интегрированное управление проектами

Risk Management –управление рисками

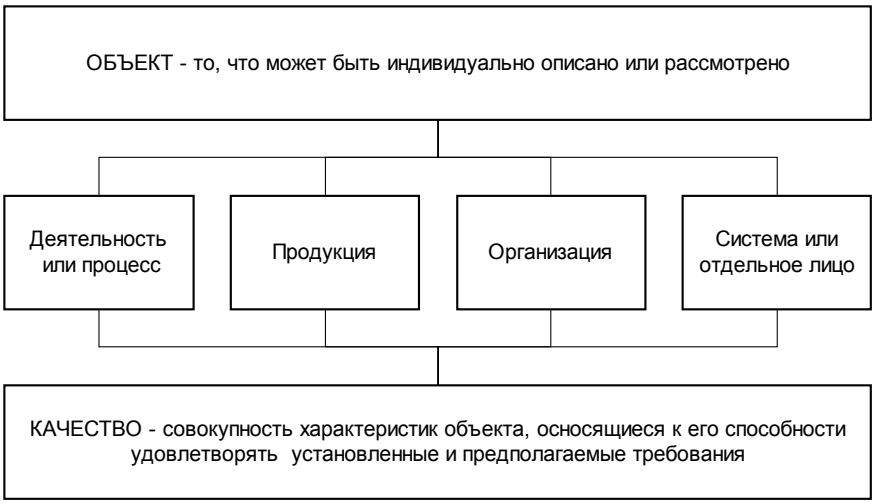
Integrated Teaming – интегрированное управление командой

Integrated Supplier Management – интегрированное управление поставщиками

Quantitative Project Management – количественное управление проектами

4.2. Терминология в области качества

4.2.1. Определение качества в ISO 8402



4.2.2. Определения словаря качества

Качество – совокупность характеристик объекта, относящихся к его способности удовлетворять установленные и предполагаемые требования.

Объект – это то, что может быть индивидуально описано или рассмотрено: деятельность или процесс, продукция, организация, система или отдельное лицо.

Качество продукции — совокупность свойств продукции, обуславливающих ее пригодность удовлетворить определенные потребности в соответствии с ее назначением.

4.3. Управление процессами: термины и определения

4.3.1. Определения бизнес-процессов

В теоретических исследованиях и стандартах используются следующие определения понятия «*бизнес-процесс»*:

- структурированный набор действий, охватывающий различные сущности предприятия и подчиненный определенной цели (ISO/CD 15531-1);
- «множество законченных состыкованных работ, которые в совокупности создают продукцию, имеющую потребительскую ценность для клиента» (Дж. Мартин);
- «набор активностей, которые преобразуют несколько видов входных характеристик в выход, имеющий ценность для потребителя» (М. Хаммер, Дж. Чампи) [197];
- специфически упорядоченная во времени и в пространстве совокупность работ, с указанием начала и конца и точным определением входов и выходов» (Т. Давенпорт) [222];
- самый большой элемент, если рассматривать поток работ, который пронизывает всю организацию, начинается у внешних поставщиков и заканчивается у внешних покупателей (DoD);
- «набор из одной или нескольких процедур или действий, которые совместно реализуют цель, обычно с помощью организационной структуры, определяющей функциональные роли и взаимоотношения» (Workflow Management Coalition, WFMC).

4.3.2. Уровни усилий в улучшении процессов

- **Непрерывное улучшение процессов** (Continuous Process Improvement, CPI) – устранение вариаций в качестве выпускаемых продуктов и оказываемых услуг.
- **Ре-дизайн бизнес-процессо**в (Business Process Redesign) - удаление активностей, не создающих дополнительной стоимости. Сокращение времени выполнения процесса, снижение стоимости.
- **Реинжиниринг бизнес-процессов** (Business process reengineering) – радикальная трансформация процессов с использованием новых технологий для достижения существенного выигрыша в производительности процесса, эффективности и качестве.

4.4. Модели жизненного цикла

4.4.1. Водопадная модель У. Ройса

Описанная Ройсом модель имеет следующие этапы:

- Описание требований (requirements)
- Дизайн (design)
- Реализация (realization)
- Верификация (verification)
- Поддержка (maintenance)

Расширенная каскадная модель выглядит так:

- Описание требований (requirements)
- Дизайн (design)
- Кодирование или Реализация (realization)
- Интеграция (Integration)
- Тестирование и отладка (testing, debugging)
- Установка (Installation)
- Поддержка (maintenance)

4.4.2. Жизненный цикл MSF

MSF является итеративной, предполагает использование объектно-ориентированного моделирования. Microsoft Solution Framework (MSF) включает четыре фазы:

- анализ
- проектирование
- разработка
- стабилизация.

4.4.3. Жизненный цикл RUP

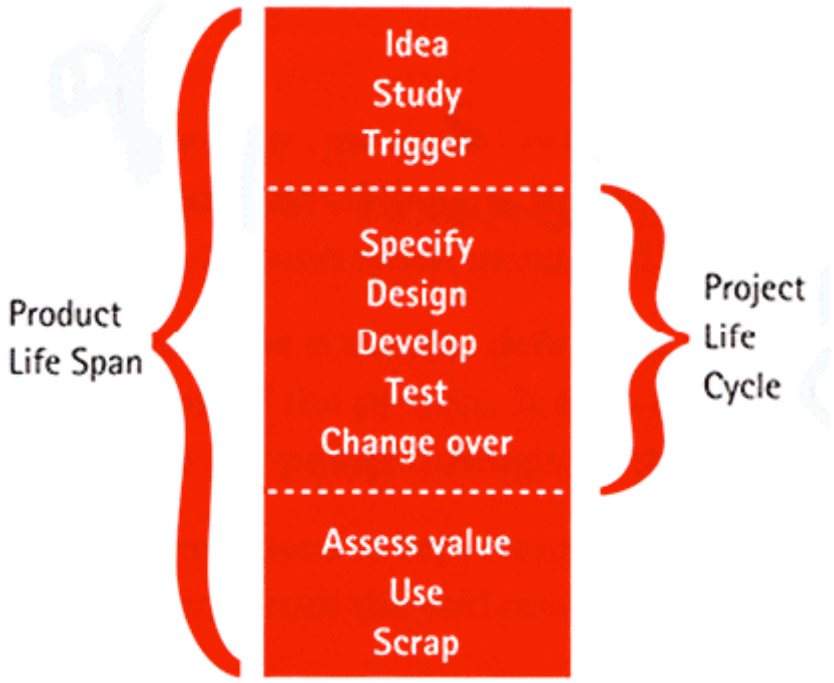
RUP использует итеративную модель разработки. В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта.

Полный жизненный цикл разработки продукта состоит из четырех фаз, каждая из которых включает в себя одну или несколько итераций:

- **Начало (Inception).** Основная цель этой фазы — достичь компромисса между всеми заинтересованными лицами относительно задач проекта. На этой фазе определяются основные цели проекта, руководитель проекта и бюджет проекта, основные средства его выполнения — технологии, инструменты, ключевой персонал, а также происходит, возможно, апробация выбранных технологий с целью подтверждения возможности достичь целей с их помощью, составляются предварительные планы проекта. Формируются видение и границы проекта. Создается экономическое обоснование (business case). Определяются основные требования, ограничения и ключевая функциональность продукта. Создается базовая версия модели прецедентов. Оцениваются риски. При завершении фазы Начало оценивается достижение вехи целей жизненного цикла Lifecycle Objective Milestone, которое предполагает соглашение заинтересованных сторон о продолжении проекта.
- **Проектирование (Elaboration)**. Основная цель этой фазы — на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы. Анализ предметной области и построение исполняемой архитектуры. Документирование требований (включая детальное описание для большинства прецедентов использования). Успешное выполнение фазы Проектирование означает достижение вехи архитектуры жизненного цикла (Lifecycle Architecture Milestone).
- **Построение (Construction).** Основная цель этой фазы — детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры. Реализация большей части функциональности продукта. Фаза Построение завершается первым внешним релизом системы и вехой начальной функциональной готовности (Initial Operational Capability).
- **Внедрение (Transition)**. Цель этой фазы — сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей. Создается финальная версия продукта и передается от разработчика к заказчику. Включает в себя программу бета-тестирования, обучение пользователей, а также определение качества продукта. В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, фаза

Внедрение повторяется снова. Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки.

4.4.4. Жизненный цикл продукта и проекта PRINCE



Жизненный цикл проекта

- Идея
- Изучение
- Начало
- *Спецификация*
- *Дизайн*
- *Разработка*
- *Тестирование*
- *Внесение изменений*
- Оценка результатов
- Использование
- Закрытие (утилизация)
- Курсивом выделены этапы жизненного цикла.

4.4.5. Модель жизненного цикла объектной разработки

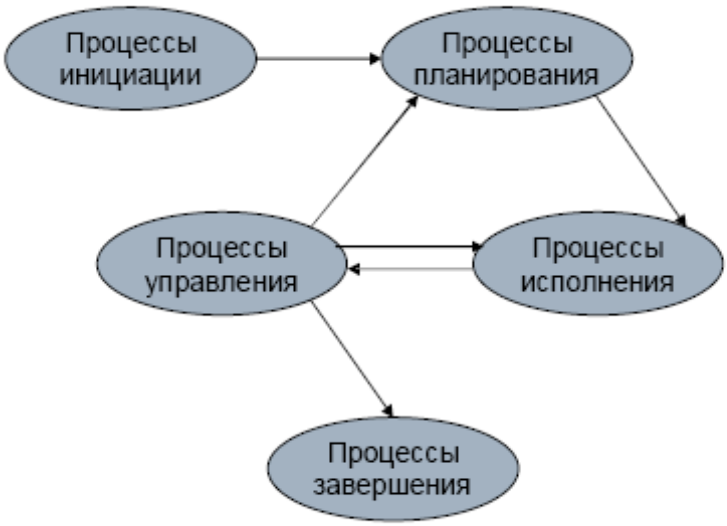
Жизненный цикл кластера включает следующие этапы (проектирование и реализация не разделены четко):

- *спецификация*: идентификация классов (абстракций данных) кластера и их главных особенностей и ограничений;
- *проектирование*: определение архитектуры классов и их отношений;
- реализация: завершение классов во всех деталях;
- *верификация и валидация*: контроль классов кластера (статическая проверка, тестирование и другие методы);
- *обобщение*: подготовка к повторному использованию.

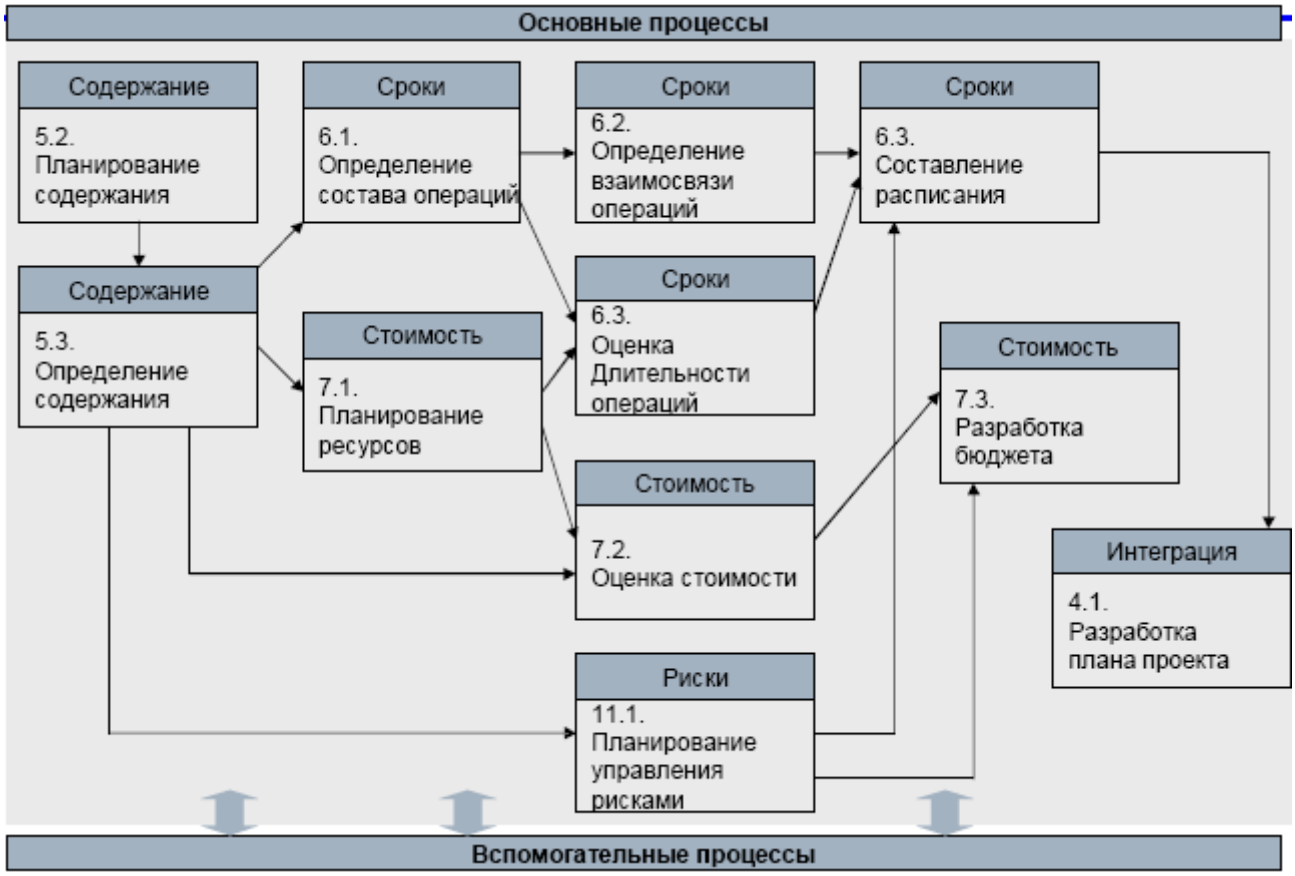
- Этап обобщения может содержать следующие действия:
- *Абстрагирование*: введение абстрактных классов там, где это необходимо.
- *Факторизацию* (Factoring): распознавание первоначально несвязанных классов, которые являются фактически вариантами того же самого понятия, так что для них можно ввести общего родителя.
- *Добавление утверждений*, особенно постусловий и инвариантов, отражающих углубленное понимание семантики класса и его особенностей. Вероятно, придется также добавить предусловие, но это скорее исправление ошибки, означающее незащищенность подпрограммы на должном уровне.
- *Добавление предложений* для обработки исключений, возможность которых первоначально игнорировалась.
- *Добавление документации.*

4.5. Модели процессов

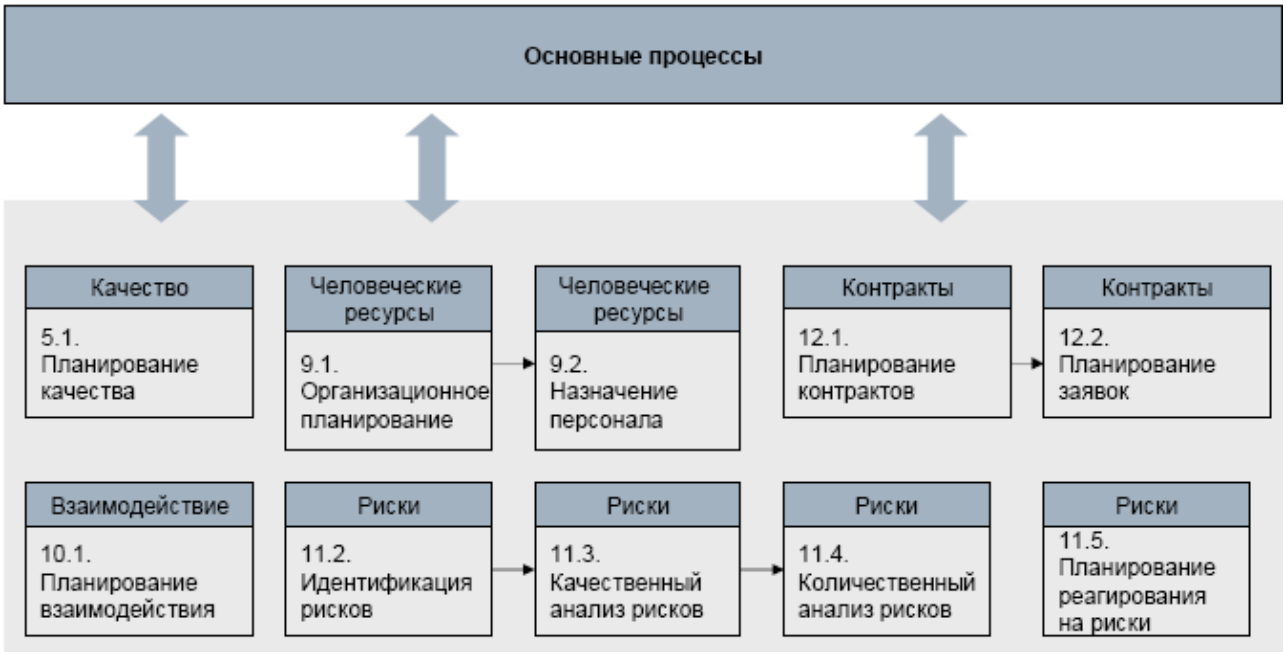
4.5.1. Модель процессов PMBoK



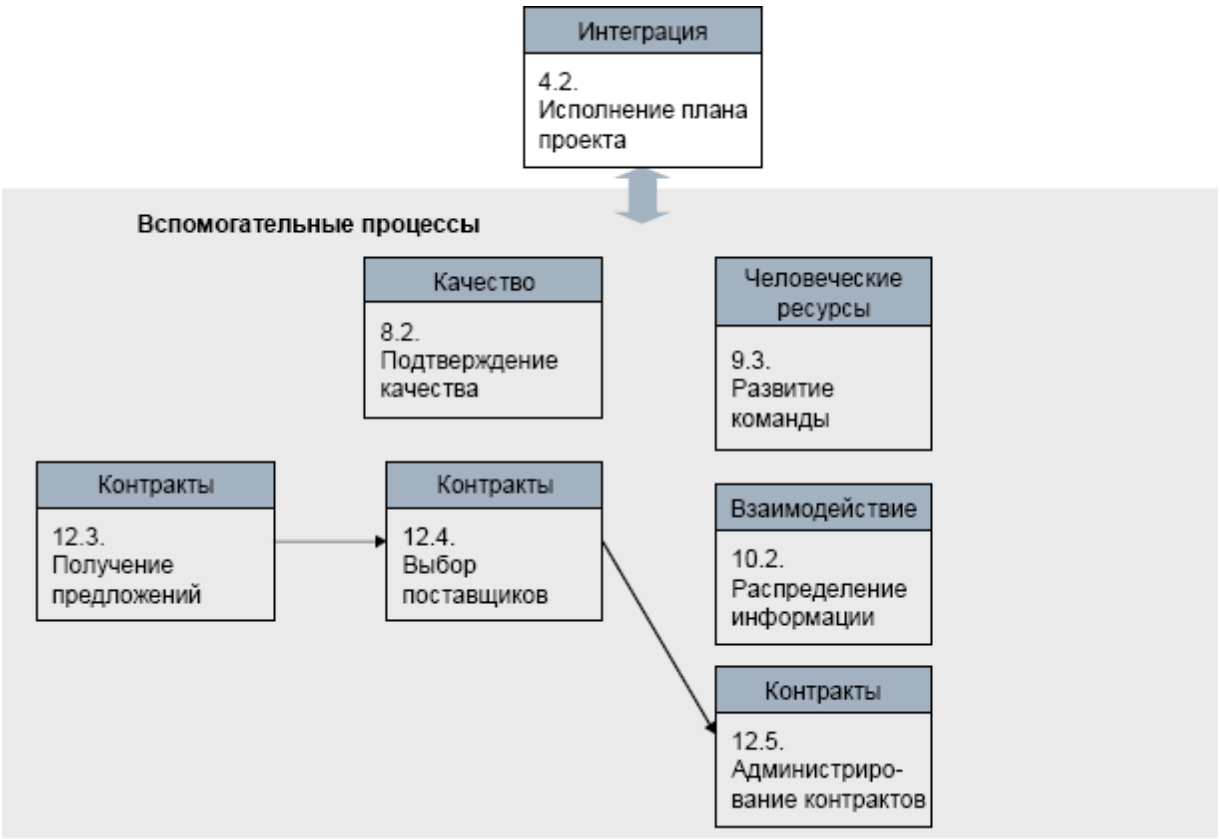
4.5.1.1. Процессы планирования (основные)



4.5.1.2. Процессы планирования (вспомогательные)



4.5.1.3. Процессы исполнения



4.5.2. Модель процессов ISO 12 207 (процессы организации)

4.5.3. Модель процессов Oracle CDM (процессы жизненного цикла)

RD - Определение производственных требований,

ES - Исследование существующих систем,

TA - Определение технической архитектуры,

DB - Проектирование и построение БД,

MD - Проектирование и реализация модулей,

CV - Конвертирование данных,

DO - Документирование,

TE - Тестирование,

TR - Обучение,

TS - Переход к новой системе,

PS - Поддержка и сопровождение.

4.5.4. Модель процессов RUP

Рациональный Унифицированный Процесс состоит из девяти рабочих процессов:

- *Моделирование бизнес-процессов* – описывается структура и динамика организации (заказчика);
- *Разработка требований* – описывается основанный на прецедентах метод постановки требований;
- *Анализ и проектирование* – описываются различные виды архитектуры системы:
- *Реализация* – собственно разработка программ, автономное тестирование и интеграция;
- *Тестирование* – описываются тестовые сценарии, процедуры и метрики для измерения числа ошибок;
- *Развертывание* – охватывает конфигурирование поставленной системы;
- Управление конфигурацией – управление изменениями и поддержание целостности артефактов проекта;
- *Управление проектом* – описывает разные стратегии работы с итеративным процессом;
- *Анализ среды* – рассматриваются вопросы инфраструктуры, необходимой для разработки системы.

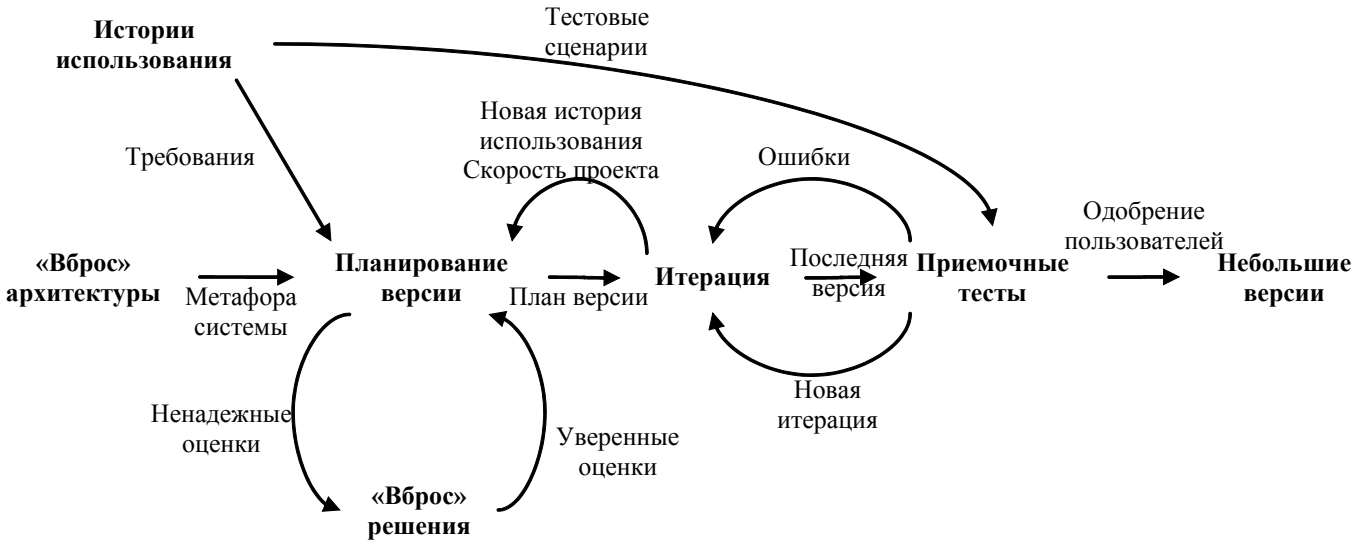
4.5.5. Модель процессов PRINCE (процессы управления проектами)

- Начало проекта - Starting up a project (SU)
- Инициация проекта - Initiating a project (IP)
- Управление проектом - Directing a project (DP)
- Контроль за этапом проекта - Controlling a stage (CS)
- Управление поставкой продукта - Managing product delivery (MP)
- Управление границами этапов проекта - Managing stage boundaries (SB)
- Закрытие проекта - Closing a project (CP)
- Планирование - Planning (PL)

4.5.6. Процессы SPICE (процессы организации)

Категория процессов	Список процессов
Процессы «Покупатель-поставщик» (CUS)	1. приобретение программного обеспечения (услуг)
	2. заключение контракта
	3. определение потребностей покупателя
	4. осуществление совместных проверок
	5. доставка и установка ПО
	6. обслуживание ПО
	7. оказание прочих услуг покупателю
	8. оценка степени удовлетворенности покупателя
Производственные процессы (ENG)	1. разработка требований к системе
	2. разработка требований к ПО
	3. разработка ПО (программирование)
	4. сборка ПО и тестирование ПО
	5. тестирование системы
	6. поддержка системы и ПО
Управление проектами (PRO)	1. планирование ЖЦ ПО
	2. разработка плана проекта
	3. формирование команды проекта
	4. управление требованиями
	5. управление качеством
	6. управление риском
	7. управление ресурсами и выполнение графика проекта
	8. управление субконтрактами
Поддерживающие процессы (SUP)	1. разработка документации
	2. управление конфигурацией
	3. поддержка системы качества
	4. разрешение конфликтов
	5. дружеские обзоры
Организационные (ORG)	1. организация бизнеса
	2. определение процессов
	3. улучшение процессов
	4. обучение
	5. обеспечение повторного использования
	6. обеспечение среды разработки ПО
	7. обеспечение специалистами

4.5.7. Модель процесса XP



4.5.8. Модель процессов PSP

PSP включает четыре процесса, которые последовательно добавляются разработчиком:

PSP0 - Базовый собственный процесс (The Baseline Personal Process);

PSP1 - Планирование (Personal Planning Process);

PSP2 - Управление качеством (Personal Quality Management);

PSP3 - Циклический процесс (Cyclic Personal Process).

4.6. Модели рисков

4.6.1. Источники рисков MSF

Источники риска	Возможные последствия
Расписание	Превышение сметы
Бюджет	Отставание от графика
Управление разработкой	Несоответствие продукта спецификации
Процессы разработки	Прекращение проекта
Среда разработки	Непланируемые изменения в штате
Персонал	Неудовлетворенность клиента
Технология	Потеря репутации фирмы-разработчика
Характеристики проекта	Деморализация персонала
Клиент / пользователь	

4.6.2. Таксономия риска SEI (278, стр. A-70)

A. Продукт	B. Среда Разработки	C. Проектные ограничения
1. Требования	1. Процесс Разработки	1. Ресурсы
Стабильность (неизменность требований, среды)	Формальность	Расписание
Полнота	Пригодность	Персонал
Обоснованность	Управление процессом (контроль процесса)	Бюджет
Понятность	Известность	Помещения и оборудование
Исполнимость	Управление продуктом (контроль продукта)	2. Контракт
Наличие прецедентов (новизна)	2. Система Разработки	Тип контракта (условия платежа, т.д.)
Масштабность (сложность)	Производительность	Ограничения
2. Проект	Пригодность	Зависимости
Функциональная исполнимость	Практичность	3. Взаимодействие с внешней средой
Сложность реализации	Известность	Клиент
Интерфейсы (взаимодействие с другими системами)	Надежность	Подрядчики
Производительность	Поддержка системы разработки	Субподрядчики
Тестопригодность	Готовность к поставке	Генеральный подрядчик
Аппаратные ограничения	3. Процессы управления	Корпоративный менеджмент
ПО других фирм	Планирование	Поставщики

A. Продукт	B. Среда Разработки	C. Проектные ограничения
3. Кодирование и тестирование элементов	Организация процесса	Политика
Реализуемость	Опыт и квалификация менеджмента	
Тестируемость элементов	Коммуникации	
Имплементация	4. Методы управления	
4. Интеграция и системное тестирование	Отслеживание проекта (мониторинг)	
Окружающая среда	Управление персоналом	
Интеграция продукта	Обеспечение качества	
Системная интеграция	Конфигурационное управление	
5. Аспекты разработки и внедрения	5. Условия работы	
Поддерживаемость	Отношение к качеству	
Надежность	Кооперация	
Безопасность	Коммуникации	
Человеческий фактор	Моральный климат	
Спецификации		

4.7. Структуры методологий

4.7.1. Структура системы MSF

Ядром системы MSF являются шесть основных моделей:

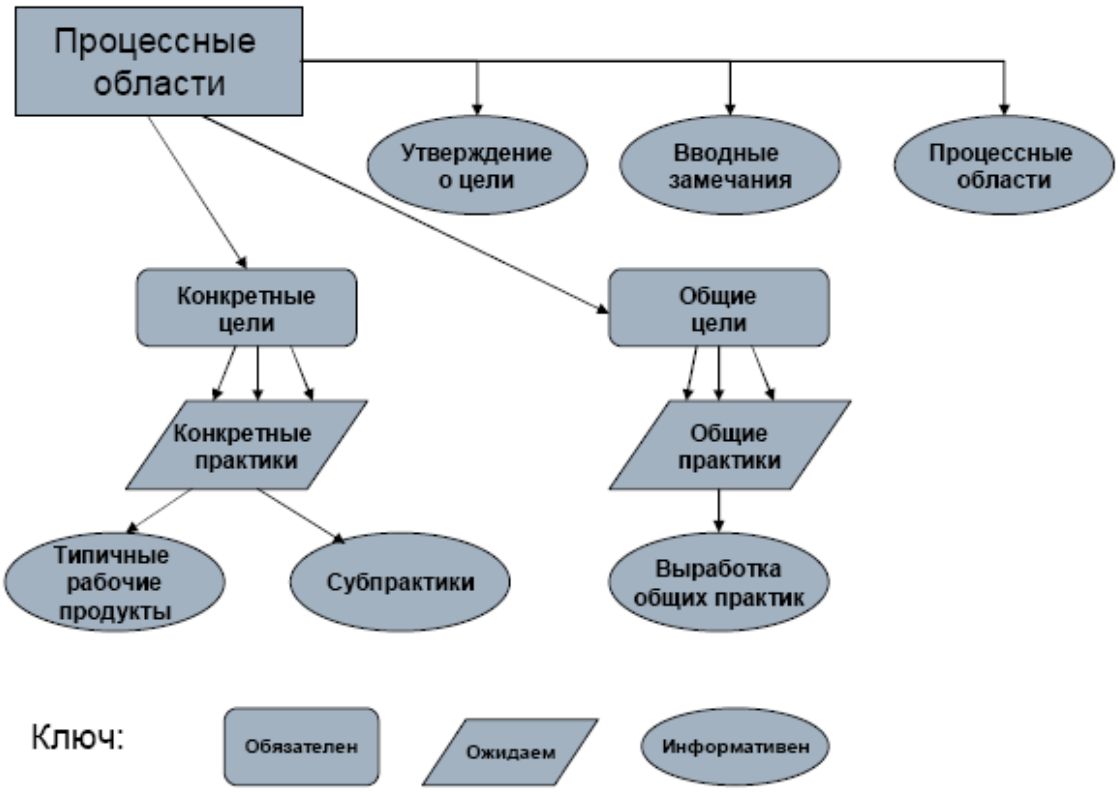
- *Модель производственной архитектуры* – набор принципов, предусматривающих быстрое создание приложения посредством выпуска версий. Основная цель – сокращение времени и затрат. Используются четыре перспективы: бизнес, приложения, информация и технология.
- *Модель проектной группы* – описывает роли и обязанности каждого участника, распределение ответственности и работу группы;
- *Модель процесса разработки ПО* – описывает фазы, этапы, виды деятельности и результаты процесса, и их связь с моделью проектной группы;
- *Модель управления рисками* – описывает путь активного управления рисками;
- *Модель процесса проектирования* – описывает трехфазный, ориентированный на конечного пользователя, непрерывный процесс разработки;
- *Модель приложения* - реализует трехуровневый, ориентированный на сервисы метод проектирования и разработки ПО.

4.7.2. Модель CMM/CMMI

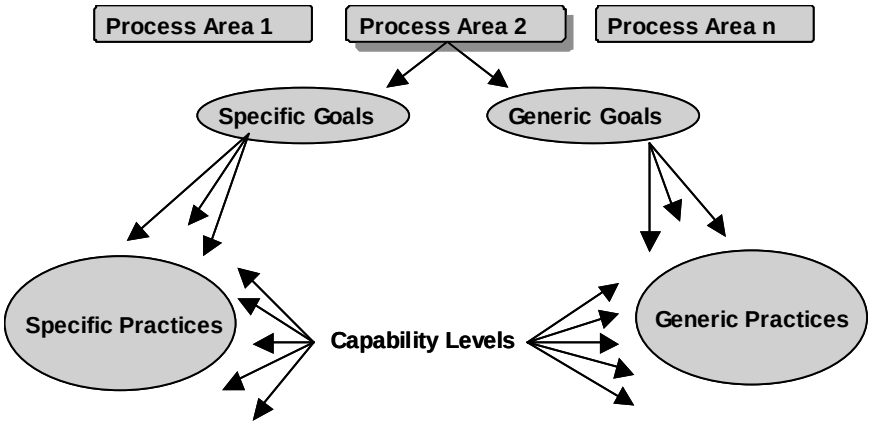
4.7.2.1. Ключевые практики CMM

- желание и готовность выполнять (commitment to perform) – следование документированной политике, определение ответственного;
- наличие возможности выполнять (ability to perform) - выделение ресурсов и финансирования, обучение, обеспечение необходимых условий;
- выполненные действия (activities performed) – разработка планов, следование плану, внесение изменений в план;
- измерение и анализ (measurement and analysis) – определение статуса работ, измерение результатов работ, производительности процесса;
- оценка результатов и процессов (verifying implementation) – проверка результатов менеджментом, группой обеспечения качества, прочие проверки.

4.7.2.2. Компонетны CMMI



4.7.2.3. Структура CMMI, Непрерывное представление



4.7.2.4. Структура модели CMM. Уровни зрелости.

Оптимизирующий (5)

управление изменением процесса

управление изменением технологии

предотвращение дефектов

Управляемый (4)

управление качеством

количественное управление

Определенный (3)

товарищеские обзоры

межгрупповая координация

технологи производства

интегрированное управление

повышение квалификации

определение процесса

нацеленность процесса

Повторяемый (2)

управление конфигурацией

обеспечение качества

управление субподрядчиками

отслеживание проекта

планирование проекта

управление требованиями

Начальный (1)

4.7.3. Ключевые области процессов CMMI

- *Управление процессами* – активности, пересекающие границы проектов, развитие организации, обучение, разработка, оценка, совершенствование процессов. Обеспечивают способность организации документировать и распространять лучшие практики, обеспечивают обучение организации;
- *Управление проектами* – активности, связанные с управлением проектами, включая планирование, контроль, управление рисками, взаимоотношения с подрядчиками, формирование команды;
- *Разработка* – активности, связанные с проектированием и разработкой, включая разработку требований и управление ими, программирование, интеграцию, валидацию и верификацию;
- *Поддержка* – активности, обеспечивающие выполнение прочих процессов: управление конфигурацией, обеспечение качества, оценка и измерения, анализ принятых решений.

4.7.4. Структура SPICE

- *Первая часть* является информативной и объясняет назначение стандарта, раскрывает его структуру.
- *Вторая* часть содержит описание модели процессов: перечень процессов и их базовых практик (base practice), а так же список общих практик (generic practices), определяющих возможности процесса.
- *Третья часть* представляет собой руководство по оценке процессов, содержит критерии оценки.
- *Четвертая часть* содержит конкретные рекомендации по организации оценки.
- *В пятой части* описываются требования к инструментам оценки, предъявляются требования к различным аспектам инструментов оценки.
- *В шестой части* содержаться требования к квалификации специалиста по оценке процессов.
- *Седьмая часть* позволяет определить исходную информацию, необходимую для оценки процессов, а так же использовать результаты оценки для улучшения процессов в различных ситуациях.
- *Восьмая честь* определяет исходную информацию и использование полученных результатов для оценки возможностей процессов. Девятая часть содержит словарь специальных терминов.
- *Девятая часть* содержит глоссарий стандарта.

Архитектура SPICE двумерна и состоит из так называемых "уровней возможностей", их насчитывается 6 (плюс 9 атрибутов процессов и 32 правила менеджмента); категорий процессов (5) и типовых процессов (29), а также типовых практик (200). SPICE предлагает законченную и подробную модель, предоставляющую пользователям достаточную свободу в выборе путей к улучшению работы.

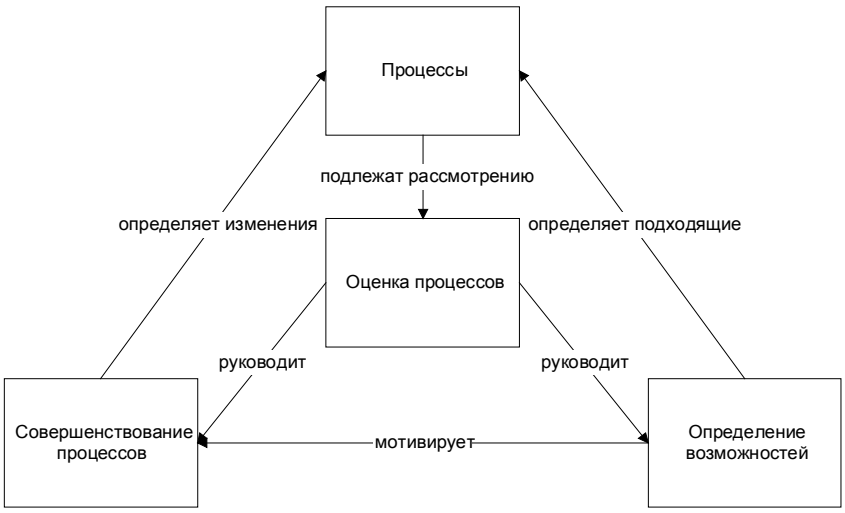
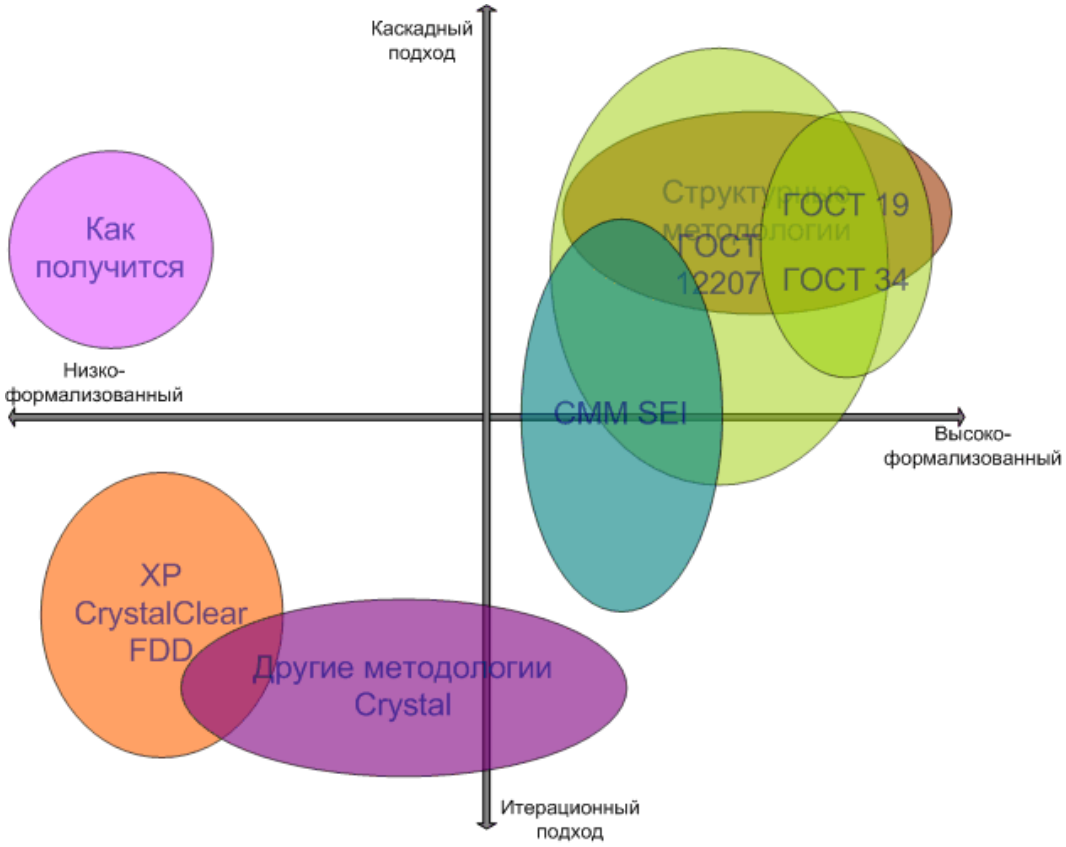


Рис. 20. Структура SPICE

4.8. Выбор методологий

4.8.1. Схема выбора методологий А. Закис (Luxoft)



4.8.2. Схема выбора методологий Crystal

Размер команды проекта

Ущерб от дефектов (Критичность)				
Жизнь (L)	L6	L20	L40	L80
Невозмес тимые средства (E)	E6	E20	E40	E80
Дискреци онные средства (D)	D6	D20	D40	D80
Удобства (C)	C6	C20	C40	C80
	Clear кварц	Yellow топаз	Orange рубин	Red сапфир

Рис. 21. Выбор методологий семейства Crystal

4.9. Метрики

4.9.1. Метрики оценки классов

Для оценки класса используются следующие метрики:

- Взвешенное количество методов на класс (WMC – Weighted Methods per Class). Существует несколько вариантов этой метрики: суммирование числа методов (NM – Number of Methods), числа строк методов, или цикломатических сложностей методов. Эти метрики используются для оценки времени и усилий, необходимых для разработки и поддержки класса. Считается, что классы с большим количеством методов отличаются большей специфичностью, что затрудняет повторное использование.
- Количество откликов (реакций) на класс (RFC – Response for a Class) – количество методов класса, которые могут быть вызваны извне в результате одного сообщения (включая вызовы в иерархии наследования). Большое значение этой метрики позволяет выявить классы, сложные для отладки и тестирования.

- Единство методов (LCOM – Lack of cohesion). Эта метрика измеряет сходство методов класса, используя входящие параметры и атрибуты. Существует по меньшей мере два способа оценки связанности: определение количества (%) использующих его методов. Для каждого элемента данных (поля базы данных), определение повторяющихся наборов атрибутов в различных методах;
- Связанность между классами объектов (CBO – Coupling Between Objects) позволяет определить количество классов, с которыми связан данный класс. Это означает, что один класс использует методы или экземпляры другого класса. Слишком большая связанность классов отрицательно влияет на модульность проекта и не позволяет повторно использовать классы, а сильно взаимосвязанная система требует большего количества тестов и времени на тестирование. Эта метрика учитывает все виды отношений: наследование, ассоциации.
- Глубина дерева наследования (DIT - Depth of Inheritance Tree) Определяется для каждого класса как количество классов-предков в иерархии наследования. Большое количество предков делает поведение класса менее предсказуемым. Глубокие деревья наследования усложняют проект, поскольку включают большее число атрибутов и методов в классах-потомках.
- Количество потомков (NOC – Number of Child). Чем больше потомков у класса, тем больше повторное использование его методов. Это положительное свойство.

4.9.2. Метрика цикломатической сложности МакКейба

Цикломатическая сложность измеряет количество логики (decision logic) в отдельном модуле. Цикломатическая сложность рассчитывается по формуле $v(G) = e-n+2$, где e – количество дуг, n – количество вершин в графе и равна количеству базовых маршрутов исполнения программы. В том случае, если алгоритм содержит только операции бинарного выбора, то $v(G) = p+1$, где p – количество выборов.

4.9.3. Метрика COCOMO

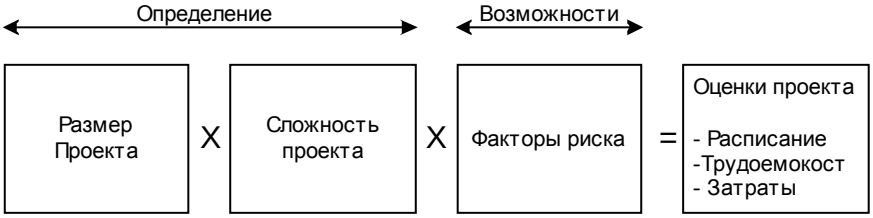


Рис. 22. Модель оценки проекта

Усилия по разработке измеряются в COCOMO II в месяцах работы (person month, PM). $PM_{nominal} = A*(Size)^B$, где A – константа, отражающая рост трудоемкости проекта с увеличением размера, *Size* – размер проекта (KSLOC), B - отражает эффект отдачи от масштаба для проектов различного размера.

Коэффициент B может принимать различные значения в зависимости от существующего масштаба проекта. Для небольших проектов $B < 1$, следовательно, увеличение размера проекта в два раза приведет менее чем к двукратному увеличению трудоемкости. На этапе прототипирования B принимается равным 1. Для крупных проектов $B > 1$, причиной отрицательной отдачи от масштаба служит усложнение коммуникаций в проекте.

COCOMO II предлагает алгоритм определение коэффициента отдачи от масштаба.

$B = 0.91 + 0.01*\sum W_i$, где W_i – определяется в зависимости от результатов анализа факторов, представленных в Табл. 10. Значение W_i меняется от 5 до 0 ростом показателя фактора масштабирования, таким образом коэффициент B оказывается в диапазоне значений от 1.01 до 1.25.

Модель COCOMO II содержит подробное описание методик оценки факторов масштабирования. Для каждого фактора приводится список влияющих атрибутов с возможными вариантами значений.

Фактор масштабирования	Очень низкое	Низкое	Среднее	Высокое	Очень Высокое	Абсолютное
Наличие прецедентов	Полностью отсутствует	Практически отсутствует	В некоторых случаях отсутствует	В общем имеется	Имеется в большинстве случаев	Имеется во всех случаях
Жесткость требований (Гибкость условий разработки)	Жесткие, неизменяемые	Изменяемые в редких случаях	В общем изменяемые	Гибкие в большинстве случаев	Большая гибкость	Заданы только общие цели
Полнота анализа факторов риска	Низкая (20%)	Некоторые (40%)	Часто (60%)	Большинство (75%)	Почти все (90%)	Все (100%)
Сплоченность команды[18] (эффективность коммуникаций[19])	Очень сложные коммуникации	Есть некоторые сложности взаимодействия	В основном кооперация	Кооперация	Высокая кооперация	Беспрепятствен-ное взаимодействие
Зрелость процессов разработки	Существует два подхода к оценке: используя соответствующий уровень СММ или взвешенное значение для ключевых областей процесса.					

Табл. 10. Факторы масштабирования проекта

Затем рассчитанная трудоемкость корректируется с использованием факторов затрат. $PM_{adjusted} = PM_{nominal} * \prod EM_i$, где EM_i – мультипликативные факторы затрат. Модель содержит аналитическую формулу расчета длительности разработки (Tdev), которая позволяет оценить продолжительность проекта, зная приведенную трудоемкость.

$T_{dev} = 3.36\, PM^{0.28+0.2*(B-1.01)}$. Интересно, что время выполнения проекта трудоемкостью один месяц будет более чем в три раза дольше при любом значении коэффициента масштабирования, длительность проекта при трудоемкости в пять месяцев оказывается сопоставима с трудоемкостью, в диапазоне от десяти до тысячи месяцев продолжительность проекта оказывается менее двадцати четырех месяцев. На наш взгляд аналитическая формула отражает реальное положение дел, результаты подтверждаются фактическими данными – действительно, не один проект не выполняется быстрее, чем за три месяца, но большинство сложных проектов удается завершить в течение двух лет.

---

[18] Команда включает в себя всех участников проекта: спонсоров проекта, клиентов, разработчиков, контрагентов.

[19] Следует отметить, что модель учитывает такой важный фактор, как культурные различия между спонсорами, заказчиком и исполнителем. Этот аспект имеет большое значение для оффшорных проектов, выполняемых в России