Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«7» сентября 2023 г.

Теория и практика программной инженерии

Учебно-методические материалы по направлению подготовки
09.03.03 Прикладная информатика
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент департамента информационных технологий и автоматики		Ситников И.О.
Старший преподаватель департамента информационных технологий и автоматики		Спиричева Н.Р.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ACM – Association for Computing Machinery

API – Application Progpam Interface

ER – Entity – Relationship

IEC – International Electrotechnical Commission

ISO – International System Organization

IDL – Interface Definition Language

ISO – International Standard Organization

ОМ – объектная модель

ООП – объектно-ориентированный подход

OMA – Object Management Architecture

ООМ – объектно-ориентированная методология

СОД – система обработки данных

ОС – операционная система

ORB – Object Request Broker

ПС – программная система

CMM – Capability Maturity Model

COM – Component Object Model

CORBA – Common Object Request Broker Architecture

COP – Component-Oriented Programming

CBSE – Component-Based Software Engineering

СРР – структура разбивки работ в проекте

CPM – Critical Path Method (метод критического пути)

PERT – Program Evaluationand Review Technique (анализ методом PERT)

PMBOK – Project Management Bodyof Knowledge (ядро знаний в области управления проектами)

SQA –Software Quality Assurance (гарантирование качества)

V&V – Verification and Validation (верификация и валидация)

ПИК – повторного использования компонент

ПИ – программная инженерия

ПО – программное обеспечение

РП – распределенное приложение

ЖЦ – жизненный цикл

SWEBOK – SoftWare Engineering of Body Knowledge

RMI – Remote Method Invocation

ВВЕДЕНИЕ

Термин «программная инженерия» (Software engineering) появился примерно 30 лет назад. К моменту его появления компьютерные программы проникли во все сферы человеческой деятельности, а их разработка стала массовым занятием. Практически нет ни одной сферы человеческой деятельности, где бы ни применялись компьютерные технологии.

Программное обеспечение играет важную роль практически во всех аспектах повседневной жизни: государственном управлении, банковском деле и финансах, образовании, транспорте, индустрии развлечений, медицине, сельском хозяйстве и юриспруденции. Количество, размеры и области применения компьютерных программ резко увеличились. В результате сотни миллиардов долларов затрачиваются на разработку программного обеспечения, и от эффективности этих программ зависят заработки и даже жизни большинства людей. Программные продукты помогли нам стать эффективнее и продуктивнее. Они помогают в решении задач и предоставляют среду для работы и развлечений, во многих случаях более защищенную, более гибкую и менее ограничивающую. Однако, несмотря на все эти успехи, **достижение адекватной стоимости, сроков разработки и качества программных продуктов является серьезной проблемой**. Существует множество причин возникновения проблем, включая следующие:

• Программные продукты относятся к самым сложным системам, которые создаются человеком, и программное обеспечение по самой своей природе обладает рядом существенных и неотъемлемых свойств (таких как сложность, незримость и изменяемость), которые затрудняют работу.

• Методы и процессы программирования, которые эффективно работают для одного человека или для небольшой команды при разработке программ умеренных размеров, плохо масштабируются для разработки крупных и сложных систем (т.е. систем, состоящих из миллионов строк кода и требующих нескольких лет работы сотен разработчиков программного обеспечения).

• Скорость изменения компьютерных и программных технологий создает потребность в новых и эволюционирующих программных продуктах. Пользовательские ожидания и конкурентная борьба, возникающие в таких условиях, существенно затрудняют возможность выпускать качественное программное обеспечение в приемлемые сроки.

Примерно каждые 10 лет происходит смена языков программирования и ОС. Это приводит к необходимости изменять ранее изготовленные и функционирующие программы применительно к новым языкам и ОС. Например, преобразованием Фортран и Кобол программ в современные языки (C#, JAVA и др.) занимаются огромные коллективы программистов из третьих стран и СНГ.

Эффективность разработчиков в зависимости от квалификации колеблется в отношении 20:200, отсюда требуется повышать уровень их знаний. На сегодня ядро стабильных знаний по программной инженерии составляет 75 % от тех знаний, что используются в практической программисткой деятельности.

Эти условия поставили перед теоретиками и умудренными опытом программистами разработку новых инженерных методов создания и управления процессами проектирования, а перед прикладниками создание стандартов, регламентируюших эти процессы.

1. ОПРЕДЕЛЕНИЕ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Инженерное дело, инженерия (от фр. ingénierie, также инжиниринг от англ. engineering, исходно от лат. ingenium — изобретательность; выдумка; знания, искусный) – область человеческой интеллектуальной деятельности, дисциплина, профессия, задачей которой является применение достижений науки, техники, использование законов (природы) и природных ресурсов для решения конкретных проблем, целей и задач человечества.

Прошло более 40 лет с момента первой организованной формальной дискуссии о программной инженерии (ПИ) как научной дисциплине на Конференции НАТО по программной инженерии, состоявшейся в 1968 году. Термин «программная инженерия» сейчас широко используется. Однако по-прежнему существуют разногласия и различные мнения по значению данного термина. Приведенные ниже определения дают несколько различных представлений о значении и природе программной инженерии. Тем не менее, им всем присуща одна общая черта: все они сходятся в том, что программная инженерия – нечто большее, чем просто написание программного кода (coding) и включает в себя аспекты качества, управления и экономики, а также знание и применение на практике этих принципов и дисциплин.

В течение многих лет давались различные определения дисциплины программной инженерии. Например, рассмотрим следующие **определения**:

• «Установление и использование правильных инженерных принципов (методов) для экономичного получения надежного и работающего на реальных машинах программного обеспечения».

• «Программная инженерия является такой формой инженерии, которая применяет принципы информатики (computer science) и математики для получения рентабельных решений в области программного обеспечения».

- «Применение систематического, дисциплинированного, поддающегося количественному определению подхода к разработке, эксплуатации и сопровождению программного обеспечения».

Все эти определения можно привести к более упрощенному виду:

Программная инженерия – это наука, которая изучает вопросы создания, сопровождения и внедрения программного обеспечения с заданным качеством, в заданные сроки и в рамках заранее определенного бюджета.

Каждое из этих определений содержит отдельные аспекты, повлиявшие на общее понимание программной инженерии, представленное в данном документе. Одно из наиболее важных наблюдений состоит в том, что программная инженерия основывается на информатике и математике. Однако, в духе инженерных традиций, она выходит за рамки этого технического базиса и использует результаты более широкого диапазона дисциплин.

Данные определения явно формулируют, что программная инженерия посвящена систематическим, управляемым и эффективным методам создания высококачественного программного обеспечения. Поэтому особое внимание уделяется анализу и оценке, спецификации, проектированию и эволюции программного обеспечения. Кроме того, в рамки данной дисциплины попадают вопросы, связанные с управлением и качеством, новизной и творчеством, стандартами, индивидуальными навыками и командной работой, а также профессиональной деятельностью, которые играют жизненно важную роль в программной инженерии.

1.1. ОТЛИЧИЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ ОТ ТРАДИЦИОННОЙ ИНЖЕНЕРИИ

Программная инженерия (Software Engineering) является отраслью компьютерной науки, изучает вопросы построения программ для компьютеров, отражает закономерности развития в ней знаний, обобщает накопленный опыт

программирования в виде комплексов общих знаний и правил регламентации инженерной деятельности разработчиков ПО.

Как инженерная дисциплина охватывает все аспекты создания ПО, начиная от разработки требований до создания, сопровождения и снятия с эксплуатации ПО, а также оценку трудозатрат, производительности и качества.

Standish Group, проанализировав работу сотен американских корпораций и итоги выполнения нескольких десятков тысяч проектов, связанных с разработкой ПО, в своем докладе с красноречивым названием «Хаос» [11] пришла к следующим неутешительным выводам (Рисунок 1):

- Только 35 % проектов завершились в срок, не превысили запланированный бюджет и реализовали все требуемые функции и возможности.
- 46 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме. Среднее превышение сроков составило 120%, среднее превышение затрат 100%, обычно исключалось значительное число функций.
- 19 % проектов полностью провалились и были аннулированы до завершения.

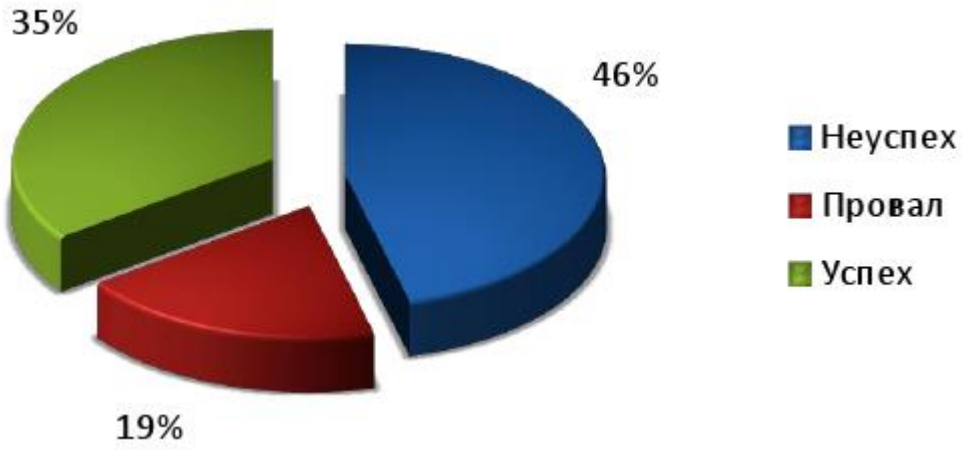


Рис. 1. Результаты анализа успешность программных проектов за 2009 год

Сразу возникают вопросы «Кто виноват?» и «Что делать?».

Кто виноват? Никто. Просто существует особая специфика в производстве программ, по сравнению с любой другой производственной деятельностью, потому

что то, что производят программисты — нематериально, это коллективные ментальные модели, записанные на языке программирования. И с этой спецификой мы обязаны считаться.

Что делать? Управлять людьми. Успех, а равно и провал, проектов по производству ПО лежат в области психологии.

Гуру программирования Ф. Брукс в 1975 году писал [11]: «Программист, подобно поэту, работает почти непосредственно с чистой мыслью. Он строит свои замки в воздухе и из воздуха, творя силой воображения. Трудно найти другой материал, используемый в творчестве, который столь же гибок, прост для шлифовки или переработки и доступен для воплощения грандиозных замыслов».

То, что производят программисты нематериально — это коллективные мысли и идеи, выраженные на языке программирования. Отрасль еще только зарождается. Время вхождения в профессию сильно меньше, чем в других инженерных дисциплинах. Разрабатывать ПО точно не сложнее, чем делать ракеты. Просто в силу уникальности отрасли опыт профессионалов, накопленный в материальном производстве и изложенный в стандарте PMI PMBOK [9], мало способствует успеху в управлении программным проектом. Управлять разработкой ПО надо иначе.

Инженерия изменений программных продуктов выполняется методами реинжинерии, реверсной инженерии (перепрограммирование) и рефакторинга программных компонентов и интерфейсов. Применение готовых продуктов (модулей, программ, систем и т.п.) в новых разработках привело к их инженерии, при которой компоненты становятся коммерческим продуктом, приносят прибыль разработчикам и сокращают затраты при создании новых систем за счет их накопления в репозитариях или электронных библиотеках.

Программостроение больших программных проектов становится инженерным по своей сути. В нем, кроме программистов, участвуют:

- менеджеры, которые планируют и управляют проектом, отслеживают сроки и затраты;

- инженеры службы хранения готовых компонентов;
- технологи, которые определяют инженерные методы и стандарты, регламентирующие и регулирующие процесс построения программных проектов;
- тестировщики, которые проверяют правильность выполнения процессов, сбор данных при тестировании и оценку качества компонентов и системы в целом.

Инструменты поддержки разработки ПО совершили гигантский скачок в своем развитии и теперь обычной практикой стало создание ПС с использованием современных визуальных и диаграммных CASE средств и UML.

Таким образом, возникновение программной инженерии определено следующими факторами:

- появление разнообразных сложных методов анализа и моделирования ПО;
- большое количество ошибок в ПО;
- необходимость в эффективной организации работы коллективов разработчиков ПО;
- повторное использование готовых программных компонентов и высокотехнологических средств разработки и управления ПО;
- реинженерия и рефакторинга отдельных компонентов систем, их адаптация к постоянно изменяющимся условиям и средам функционирования.

Программная инженерия **качественно отличается** от других инженерных дисциплин нематериальностью программного обеспечения и дискретной природой его функционирования. В отличие от результатов классических инженерий программа не может быть проверена точными методами естественных наук (как например, в механике или строительстве). Тестирование продукта – это единственный способ убедиться в его качестве. Именно поэтому стоимость тестирования составляет существенную стоимость ПО.

Программная инженерия отличается от традиционной инженерии особой природой программного обеспечения, в связи, с чем основной упор делается на абстракцию, моделирование, организацию и представление информации, а также на управление изменениями. Программная инженерия также включает в себя деятельность по реализации проекта и контролю качества, которая в традиционном инженерном цикле обычно относится к фазам проектирования производственного процесса и производства. Кроме того, непрерывная эволюция (т.е. «сопровождение») также является критически важной для программного обеспечения.

Программная инженерия стремится интегрировать принципы математики и информатики с инженерными подходами, разработанными для производства осязаемых материальных артефактов.

Основываясь на математике и компьютинге, программная инженерия занимается разработкой систематических моделей и надежных методов производства высококачественного программного обеспечения, и данный подход распространяется на все уровни – от теории и принципов до реальной практики создания программного обеспечения.

Знания разработчиков ПО отличаются большим разнообразием, являются не согласованными и разнородными, ориентированными на разные предметные области, поэтому мировая компьютерная общественность пришла к необходимости систематизировать знания в области программной инженерии, создав ядро знаний SWEBOK (Software Engineering Body of Knowledge).

В начале 90–х профессиональные ассоциации ACM (Association for Computing Machinery) и IEEE Computer Society (Institute of Electrical and Electronics Engineers) начали разработку стандартов в области программной инженерии и обучении этой дисциплине. В частности специальный комитет разработал документ под названием IEEE Guideto the Software Engineering Body of Knowledge (SWEEBOK) [2], доступная редакция которого вышла в 2004 г. По смыслу данный

документ описывает знания инженера в области программной инженерии со стажем 4 года.

1.2. SWEBOK

Ядра знаний SWEBOK является основополагающим документом, отображает мнение многих зарубежных и отечественных специалистов в области программной инженерии и согласуется с современными регламентированными процессами ЖЦ ПО стандарта ISO/IEC 12207. В этом ядре знаний содержится описание 10 областей, каждая из которых представлена согласно принятой всеми участниками создания этого ядра общей схемы описания, включающей определение понятийного аппарата, методов и средств, а также инструментов поддержки инженерной деятельности. Описание каждой области вносит определенный запас знаний, который должен практически использоваться на соответствующих процессах ЖЦ с учетом приведенного стандарта.

Для наглядного представления понятийного аппарата областей SWEBOK проведено условное разбиение областей (рис. 2, 3) на основные (пять процессов проектирования ПС) и дополнительные, организационные методы и подходы, которые отображают инженерию управления проектированием ПС (конфигурацией, проектами, качеством и т.д.). В каждой области приведены ключевые понятия, подходы и методы проектирования разных типов ПС. Данное разбиение областей на главные и вспомогательные области соответствует структуре процессов стандарта ISO/IEC 12207, выполнение которых определяется знаниями, содержащимися в ядре SWEBOK и изученными разработчиками ПС.

Далее приводится изложение каждой в отдельности области знаний ядра знаний SWEBOK, их назначение и роль при проектировании и реализации программных продуктов.

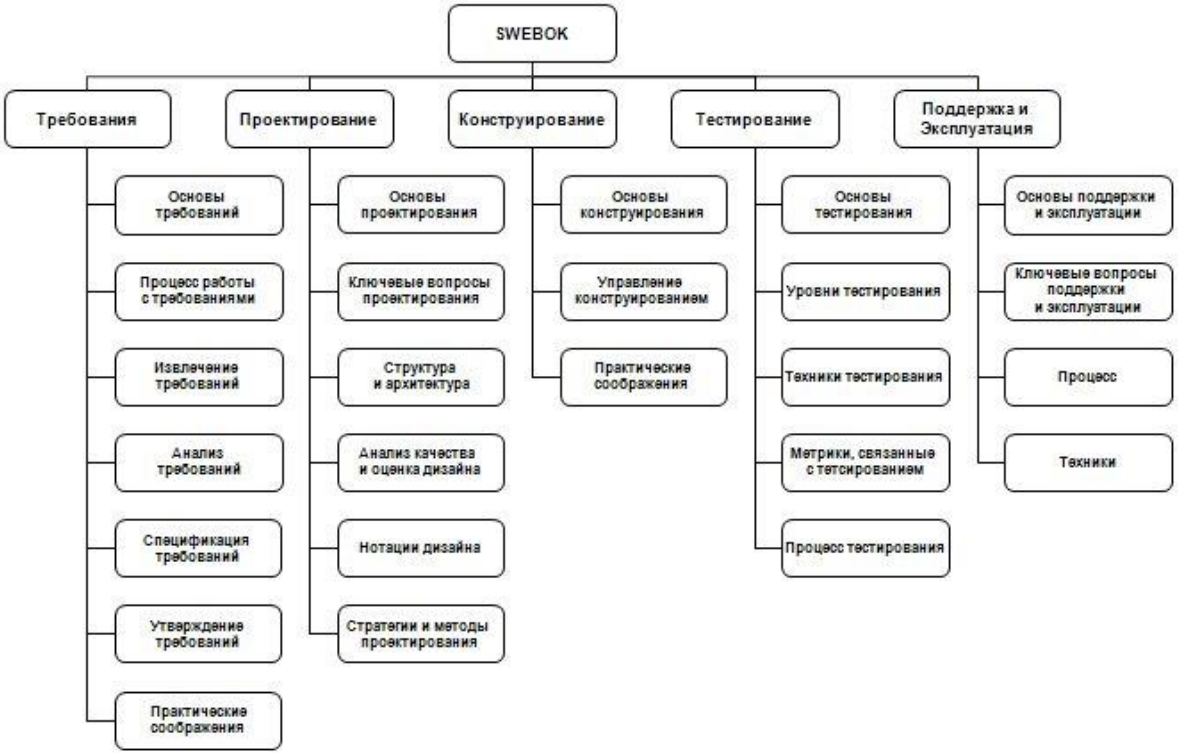


Рис. 2. Первые пять областей знаний SWEBOK на русском языке

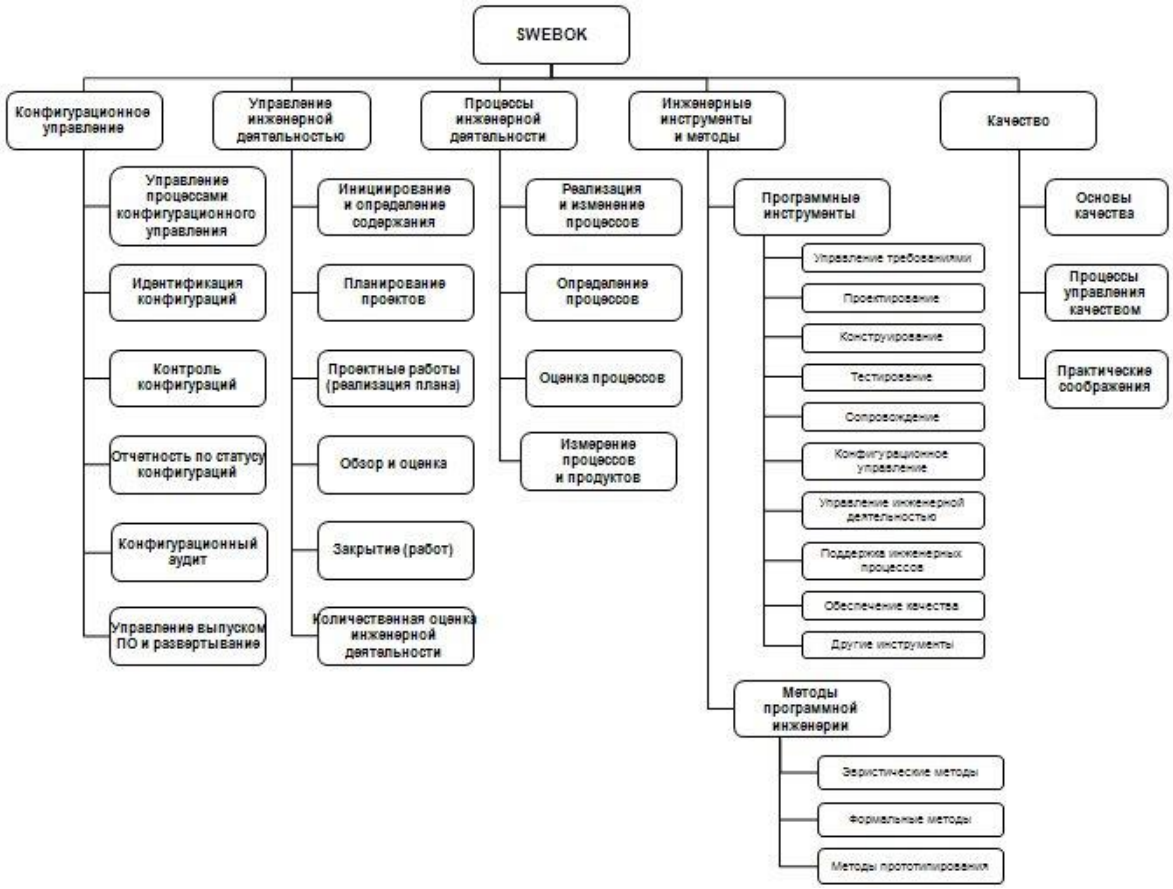


Рис. 3. Вторые пять областей знаний SWEBOK на русском языке

SWEBOK описывает 10 областей знаний (рис.2,3):

Software requirements – программные требования

Требования – это свойства, которыми должно обладать ПОдля адекватного задания функций, а также условия и ограничения на ПО, данные, среду выполнения и технику. Требования отражают потребности людей (заказчиков, пользователей, разработчиков), заинтересованных в создании ПО.

Software design – дизайн ПО (архитектура)

Проектирование ПО – процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного результата.

Software construction – конструирование программного обеспечения

Конструирование ПО – создание работающего ПОс привлечением методов кодирования, верификации и тестирования компонентов.

Software testing – тестирование

Тестирование ПО – это процесс проверки работы программы в динамике, основанный на выполнении конечного набора тестовых данных и сравнения полученных результатов с запланированными вначале.

Software maintenance – эксплуатация (поддержка) программного обеспечения

Сопровождение ПО – совокупность действий по обеспечению работы ПО, а также по внесению изменений в случае обнаружения ошибок в процессе эксплуатации, по адаптации ПО к новой среде функционирования, а также по повышению производительности или других характеристик ПО. Сопровождение (согласно стандартам ISO/IEC 12207 и ISO/IEC 14764) считается модификацией программного продукта в процессе эксплуатации при условии сохранения целостности продукта. Сопровождение, как эволюционная разработка программных систем, так как сданная в эксплуатацию система не всегда является полностью завершенной, и требует изменений в течение всего срока эксплуатации.

Software configuration management – Управление конфигурацией – дисциплина идентификации компонентов системы, определения функциональных и физических характеристик аппаратного и программного обеспечения для

проведения контроля внесения изменений и трассирования конфигурации на протяжении ЖЦ.

Software engineering management – управление инженерией ПО (менеджмент) – руководство работами команды разработчиков ПО в процессе выполнения плана проекта, определение критериев и оценка процессов и продуктов проекта с использованием общих методов управления, планирования и контроля работ.

Software engineering process – процессы программной инженерии.

Данная область знаний связана со всеми элементами управления процессами ЖЦ ПО, изменения которых проводятся в связи с их совершенствованием. Цель управления в применении лучших процессов, соответствующих реальной практике выполнения конкретного проекта.

Software engineering tools and methods – инструменты и методы

Методы и средства включают среду разработки, средства и методы разработки, используемые на процессах ЖЦ. Средства обеспечивают спецификацию требований, конструирование и сопровождение ПО. Методы обеспечивают проектирование, реализацию и выполнение ПОна процессах, а также достижение качества процессов и продуктов.

– *Softwarequality* – качество программного обеспечения.

Набор характеристик продукта или сервиса, которые характеризуют его способность удовлетворить установленным или предполагаемым потребностям заказчика.

В дополнение к перечисленным областям знаний SWEBOK также включает обзор смежных дисциплин, связь с которыми представлена как фундаментальная, важная и обоснованная для программной инженерии (рис.5):

Computer engineering

Computer science

Management

Mathematics

Project management

Quality management

Systems engineering

Рис. 5. Области знаний связанных дисциплин на русском языке

Кроме SWEEBOK была также опубликован отчет специальной комиссии ACM и IEEE Computer Science, Содержащий рекомендации по преподаванию программной инженерии и информатики. Существует перевод на русский язык [3].

1.3 Методы и средства программной инженерии

В данном курсе более подробно будет рассмотрена одна из областей знаний SWEEBOK, а именно Software engineering tools and methods –методы и средства программной инженерии (МСПИ).

В соответствии со SWEEBOK (рис.6):

1. Инструменты программной инженерии (Software Engineering Tools)

1.1. Инструменты работы с требованиями (Software Requirements Tools)

1.2. Инструменты проектирования (Software Design Tools)

1.3. Инструменты конструирования (Software Construction Tools)

1.4. Инструменты тестирования (Software Testing Tools)

1.5. Инструменты сопровождения (Software Maintenance Tools)

1.6. Инструменты конфигурационного управления (Software Configuration Management Tools)

1.7. Инструменты управления инженерной деятельностью (Software Engineering Management Tools)

1.8. Инструменты поддержки процессов (Software Engineering Process Tools)

1.9. Инструменты обеспечения качества (Software Quality Tools)

1.10. Дополнительные аспекты инструментального обеспечения (Miscellaneous Tool Issues)

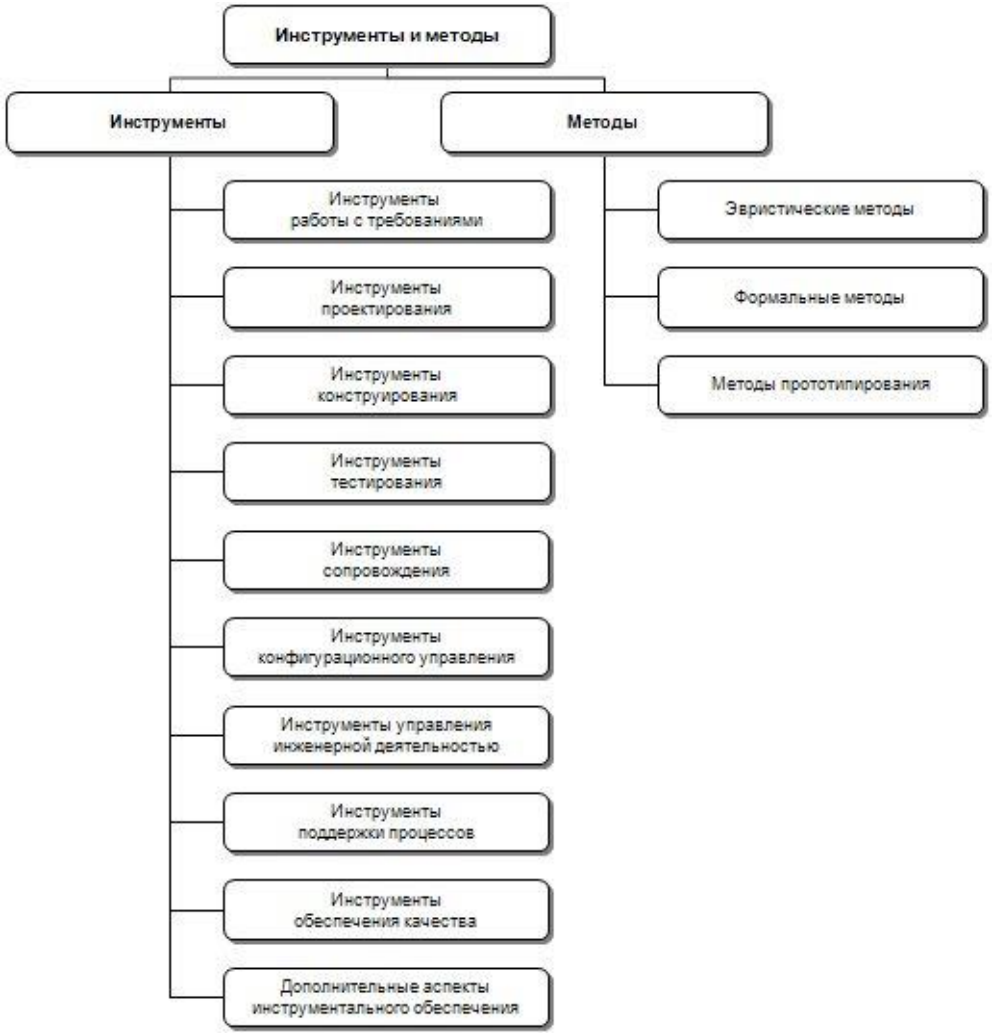


Рисунок 6. Область знаний “Инструменты и методы программной инженерии”

Программные инструменты предназначены для обеспечения поддержки процессов жизненного цикла программного обеспечения. Инструменты позволяют автоматизировать определенные повторяющиеся действия, уменьшая загрузку инженеров рутинными операциями и помогая им сконцентрироваться на творческих, нестандартных аспектах реализации выполняемых процессов. Инструменты часто *проектируются* с целью поддержки конкретных (частных) методов программной инженерии, сокращая административную нагрузку, ассоциированную с “ручным” применением соответствующих методов. Так же, как и методы программной инженерии, инструменты призваны сделать программную инженерию более систематической деятельностью и по своему содержанию (предлагаемой функциональности) могут варьироваться от поддержки отдельных индивидуальных задач вплоть до охвата всего жизненного цикла (в этом случае часто говорят об инструментальной платформе или просто платформе разработки).

Методы программной инженерии представляют собой определенные систематические подходы, которые накладывают определенные структурные ограничения на деятельность в рамках программной инженерии с целью более вероятного и скорого, с точки зрения соответствующего метода, достижением успеха. Методы обычно предоставляют соответствующие соглашения (нотацию), словарь терминов и понятий и процедуры выполнения идентифицированных (и охватываемых методом) задач, а также рекомендации по оценке и проверке выполняемого процесса и получаемого в его результате продукта [3].

Методы, как и инструменты, варьируются по содержанию (охватываемой области применения) от отдельной фазы жизненного цикла (или даже процесса) до всего жизненного цикла. Данная область знаний касается только методов, охватывающих множество фаз (этапов) жизненного цикла.

К методам программной инженерии относятся:

эвристические методы (heuristic methods),

формальные методы (formal methods),

методы прототипирования (prototyping methods).

Эвристические методы включают:

Структурные методы (structured methods). При таком подходе системы строится с функциональной точки зрения, начиная с высокоуровневого понимания поведения системы с постепенным уточнением низкоуровневых деталей. (такой подход иногда также называют «проектированием сверху-вниз»;

Методы, ориентированные на данные (data-oriented methods). Отправной точкой такого подхода являются структуры данных, которыми манипулирует создаваемое программное обеспечение. Функции в этом случае являются вторичными.

Объектно-ориентированные методы, которые рассматривают предметную область как коллекцию объектов, а не функций;

Методы, ориентированные на конкретную область применения (domain-specificmethods). Такие специализированные методы разрабатываются с учетом специфики решаемых задач, например систем реального времени, безопасности и защищенности.

Формальные методы основаны на формальных спецификациях, анализе, доказательстве и верификации программ. Формальная спецификация записывается на языке, синтаксис и семантика которого определены формально и основаны на математических концепциях (алгебре, теории множеств, логике).

[1]: “Термин *формальные методы* подразумевает ряд операций, в состав которых входит создание формальной спецификации системы, анализ и доказательство спецификаций, реализация системы на основе преобразования формальной спецификации в программы и верификация программ. Все эти действия зависят от формальной спецификации программного обеспечения. Формальная спецификация – это системная спецификация, записанная на языке, словарь, синтаксис и семантика которого определены формально. Необходимость формального определения языка предполагает, что этот язык основывается на математических концепциях. Здесь используется область математики, которая

называется дискретной математикой и основывается на алгебре, теории множеств и алгебре логики”.

Различаются следующие категории формальных методов:

- Языки и нотации спецификации (specification languages and notations), ориентированные на модель, свойства и поведение, примером такого рода методов являются формальные методы описания требований, интерес к которым периодически возникает на протяжении всей истории программной инженерии;
- Уточнение спецификации (refinements pecification) путем трансформации в конечный результат, близкий к конечному исполняемому программному продукту;
- Методы доказательства (верификации) (verification/proving properties), использующие утверждения (теоремы), пред- и постусловия, которые формально описываются и применяются для установления правильности спецификации программ. Эти методы применялись в основном в теоретических экспериментах и более 25 лет их практическое применение было ограничено из-за трудоемкости и экономической невыгодности. В 2005 г. проблема верификации приобрела вновь актуальность в связи с разработкой нового международного проекта по верификационному ПО "Целостный автоматизированный набор инструментов для проверки корректности ПС", ставящим следующие перспективные задачи:
- разработка единой теории построения и анализа программ;
- построение многостороннего интегрированного набора инструментов верификации на всех производственных процессах
- разработка формальных спецификаций, их доказательство и проверка правильности, генерация программ и тестовых примеров, уточнение, анализ и оценка;

– создание репозитария формальных спецификаций, верифицированных программных объектов разных типов и видов.

Предполагается, что формальные методы верификации будут охватывать все аспекты создания и проверки правильности программ. Это приведет к созданию мощной верификационной производственной основы и значительному сокращению ошибок в ПО.

Методы прототипирования (Prototyping Methods) основаны на прототипировании ПО т.е. разработке программных компонент для временного использования.

Существуют следующие подходы, касающиеся прототипирования:

создание временно используемых прототипов (throwaway – одноразовый, предназначенный для непродолжительного применения)

эволюционное прототипирование (в подавляющем большинстве случаев предполагает превращение прототипа в конечный продукт, прим. автора)

разработка исполняемых спецификаций (часто основывается на формальных методах);

Техники оценки/исследования (evaluation) результатов прототипирования касаются того, как именно будут использованы результаты создания прототипа (например, будет ли он трансформирован в продукт, создается он для оценки нагрузочных способностей и других аспектов масштабируемости и т.п.).

ЛИТЕРАТУРА

1. Соммервилл, Иан. Инженерия программного обеспечения. 6-е издание. Издательский дом “Вильямс”, 2002. – 624 с.

2. The Guide to the Software Engineering Body of Knowledge. SWEBOK, IEEE Computer Society Professional Practices Committee, [Электронный ресурс] – режим доступа: http://www.computer.org/portal/web/swebok/ htmlformat.

3. Орлик С. Руководство к своду знаний по программной инженерии [Электронный документ] – Режим доступа: http://swebok.sorlik.ru/download.html.

4. Липаев В.В. Программная инженерия. Учебник. – М.: ТЕИС, 2006. – 608 с.

5. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения. – Учебник, М.: 2006. – 320 с.

6. Терехов А.Н. Что такое программная инженерия. Журнал «Программная инженерия», №1, 2010 г. с. 40-45.

7. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

8. Rubinstein D., «Standish Group Report: There's Less Development Chaos Today». 2007. [Электронный ресурс] – режим доступа: http://www. sdtimes.com/content/article.aspx?ArticleID=30247.

9. Брукс Фредерик, «Мифический человеко-месяц, или Как создаются программные комплексы», Пер. с англ., СПб., Символ-Плюс, 1999.

10. PMBOK. Руководство к Своду знаний по управлению проектами, 4-е изд., М: 2010. – 496 с.

11. С. Архипенков. Лекции по управлению программными проектами. [Электронный ресурс] – режим доступа: http://citforum.ru/SE/project/ arkhipenkov_lectures.

2. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ПРОДУКТА

2.1. Исторический экскурс

Появление понятия жизненного цикла программного обеспечения (ПО) было связано с кризисом программирования, который наметился в конце 60-х и начале 70-х годов прошлого века. Суть кризиса состояла в том, что программные проекты все чаще стали выходить из-под контроля: нарушались сроки, превышались запланированные объемы финансирования, результаты не соответствовали требуемым. Многие проекты вообще не доводились до завершения. Кроме того, оказалось, что недостаточно разработать программу, а надо ее еще сопровождать и этап сопровождения часто требует больше средств, чем разработка.

Ситуация была вызвана ростом сложности проектов. Масштабы ее нарастали. Необходимо было принимать меры для радикального усовершенствования принципов и методов разработки ПО с учетом его развития и сопровождения. Заговорили о том, что надо обратиться к опыту промышленного проектирования и производства, где был накоплен опыт успешной разработки не менее сложных проектов.

Методологическую основу промышленной инженерии составляет понятие жизненного цикла изделия (продукта) как совокупности всех действий, которые надо выполнить на протяжении всей «жизни» изделия. Смысл жизненного цикла состоит во взаимосвязанности всех этих действий.

Одним из ключевых понятий управления проектами, в том числе в приложении к индустрии программного обеспечения, является ***жизненный цикл проекта*** **(*Project Lifecycle Management* – PLM).**

Известный эксперт по управлению высокотехнологичными проектами Арчибальд так определяет жизненный цикл проекта:

Жизненный цикл проекта имеет определенные начальную и конечную точки, привязанные к временной шкале. Проект в своем естественном развитии проходит ряд отдельных фаз.

Жизненный цикл проекта включает все фазы от момента инициации до момента завершения. Переходы от одного этапа к другому редко четко определены, за исключением тех случаев, когда они формально разделяются принятием предложения или получением разрешения на продолжение работы. Однако, в начале концептуальной фазы часто возникают сложности с точным определением момента, когда работу можно уже идентифицировать как проект (в терминах управления проектами), особенно если речь идет о разработке нового продукта или новой услуги [1].

Существует общее соглашение о выделении четырех обобщенных фаз жизненного цикла (рис. 1) (в скобках приведены используемые в различных источниках альтернативные термины):

- инициализация (концепция, идентификация, отбор)
- планирование (определение, анализ)
- реализация (выполнение, внедрение, производство и развертывание, проектирование или конструирование, сдача в эксплуатацию, инсталляция, тестирование и т.п.)
- завершение (включая оценивание после завершения или закрытие)

Однако, эти фазы столь широки, что обычно внутри каждой из этих фаз выделяются дополнительные подфазы.

Нередко можно наблюдать частичное совмещение или одновременное выполнение фаз проекта, называемое «быстрым проходом» в строительных и инжиниринговых проектах и «параллелизмом» – в военных и аэрокосмических. Это усложняет планирование проекта и координацию усилий его участников, а также делает более важной роль менеджера проектов.

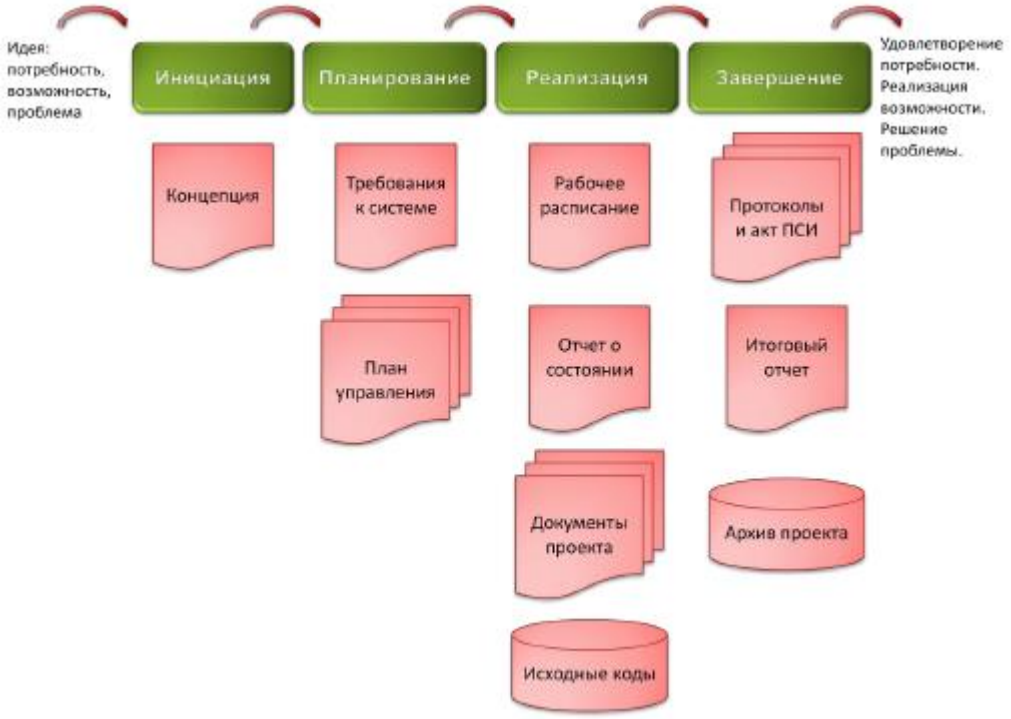


Рис. 1. Жизненный цикл и основные продукты программного проекта

В общем случае *жизненный цикл определяется моделью* и *описывается в форме методологии (метода)*. *Модель или парадигма* жизненного цикла определяет концептуальный взгляд на организацию жизненного цикла и, часто, основные фазы жизненного цикла и принципы перехода между ними. *Методология (метод)* задает комплекс работ, их детальное содержание и ролевую ответственность специалистов на всех этапах выбранной модели жизненного цикла, обычно определяет и саму модель, а также рекомендует *практики (лучшие практические решения)*, позволяющие максимально эффективно воспользоваться соответствующей методологией и ее моделью.

В индустрии программного обеспечения можно (так как это уже конкретная область приложения концепций и практик проектного управления) и необходимо (для обеспечения возможности управления) более четкое разграничение фаз проекта (что *не* подразумевает их линейного и последовательного выполнения).

Ниже приведены определения модели жизненного цикла программной системы, даваемые, например, в различных вариантах стандартов ГОСТ:

- Модель жизненного цикла – структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения ее использования [ГОСТ 12207, 1999].
- Жизненный цикл автоматизированной системы (АС) - совокупность взаимосвязанных процессов создания и последовательного изменения состояния АС, от формирования исходных требований к ней до окончания эксплуатации и утилизации комплекса средств автоматизации АС [ГОСТ 34, 1990].

Один из них – ГОСТ Р ИСО/МЭК 12207 является переводом международного стандарта ISO/IEC 12207, на основе которого, в свою очередь, создан соответствующий стандарт IEEE 12207. Второй – в рамках семейства ГОСТ 34 – разрабатывался в СССР самостоятельно, как стандарт на содержание и оформление документов на программные системы в рамках Единой системы программной документации (ЕСПД) и Единой системы конструкторской документации (ЕСКД). В последние годы, акцент делается на стандарты ГОСТ, соответствующие международным стандартам. В то же время 34-я серия является важным дополнительным источником информации для разработки и стандартизации внутрикорпоративных документов и формирования целостного понимания и видения концепций жизненного цикла в области программного обеспечения.

В определенном контексте, «модель» и «методология» могут использоваться взаимозаменяемым образом, например, когда мы обсуждаем разграничение фаз проекта. Говоря «жизненный цикл», мы, в первую очередь, подразумеваем «модель жизненного цикла». Несмотря на данное в стандартах 12207 определение модели жизненного цикла, все же, *модель* чаще подразумевает именно *общий принцип* организации жизненного цикла, чем детализацию соответствующих работ. Соответственно, определение и выбор модели, в первую очередь, касается вопросов определенности и стабильности требований, жесткости и детализированности плана работ, а также частоты сборки работающих версий создаваемой программной системы.

Скотт Амблер (Scott W. Ambler), автор концепций и практик гибкого моделирования (Agile Modeling) и Enterprise Unified Process (расширение Rational Unified Process), предлагает следующие уровни жизненного цикла, определяемые соответствующим содержанием работ (рис. 2):

- Жизненный цикл разработки программного обеспечения – проектная деятельность по разработке и развертыванию программных систем
- Жизненный цикл программной системы – включает разработку, развертывание, поддержку и сопровождение
- Жизненный цикл информационных технологий (ИТ) – включает всю деятельность ИТ-департамента
- Жизненный цикл организации/бизнеса – охватывает всю деятельность организации в целом

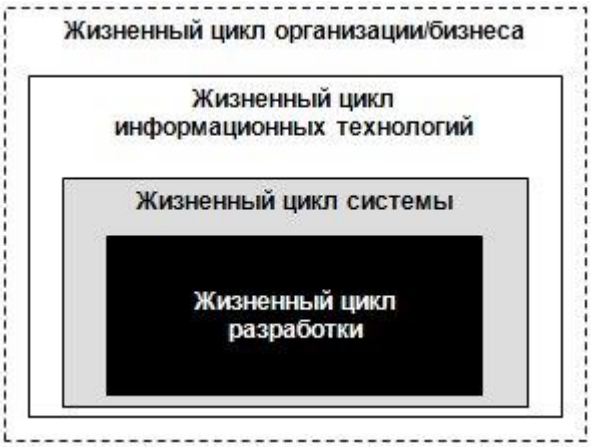


Рис. 2. Содержание четырех категорий жизненного цикла по Амблеру

История разработки стандартов ЖЦ ПО

Разрабатывались стандарты ЖЦ ПО. Наиболее известными являются:

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

250 типовых обязательных требований к процессам и объектам проектирования на этих этапах.

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

1995 г. IEEE 1074. Процессы жизненного цикла для развития программного обеспечения. Охватывает полный жизненный цикл ПС, в котором выделяются шесть крупных базовых процессов. Эти процессы детализируются 16 частными процессами. В последних имеется еще более мелкая детализация в совокупности на 65 процессов-работ. Содержание каждого частного процесса начинается с описания общих его функций и задач и перечня действий — работ при последующей детализации. Для каждого процесса в стандарте представлена входная и результирующая информация о его выполнении и краткое описание сущности процесса. В стандарте внимание сосредоточено преимущественно на непосредственном создании ПС и на процессах предварительного проектирования. В приложении представлены четыре варианта адаптации максимального состава компонентов ЖЦ ПС к конкретным особенностям типовых проектов.

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

- Внедрение стандартов требовало вложения значительных средств, что не всегда окупалось;
- Было неясно, все ли требуемые процессы надо выполнять и в какой мере;
- Различные типы ПО (ИС, реального времени, бизнес системы), различные требования;

– Высокая динамика отрасли и устаревание стандартов;
– Терминологическая неоднозначность различных корпоративных стандартов;
– Во многих случаях применение стандартов было вызвано только требованиями заказчиков, хотя на практике превращалось в тормоз и гробило выполнение проектов.

Одним из фундаментальных взглядов на жизненный цикл является стандарт процессов жизненного цикла ISO/IEC, IEEE, ГОСТ Р ИСО/МЭК 12207.

2.2. Жизненный цикл ПП: структура и организация

Разрешением проблем стандартизации ЖЦ ПО явилась разработка и принятие в 1995 г. стандарта ISO/IEC 12207 – Information Technology – Software Life Cycle Processes (ISO – International Organization of Standardization – Международная организация по стандартизации; IEC – International Electrotechnical Commission – Международная электротехническая комиссия). В 2000 г. он был принят как ГОСТ 12207. Процессы жизненного цикла программных средств.

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

– Введение единой терминологии по разработке и применению ПО (предназначен не только для разработчиков, но и для заказчиков, пользователей, всех заинтересованных лиц);
– Разделение понятий ЖЦ ПО и модели ЖЦ ПО. ЖЦ ПО в стандарте вводится как полная совокупность всех процессов и действий по созданию и применению ПО, а модель ЖЦ – конкретный вариант организации ЖЦ, обоснованно (разумно) выбранный для каждого конкретного случая;
– Описание организации ЖЦ и его структуры (процессов);
– Выделение процесса адаптации стандарта для построения конкретных моделей ЖЦ.

Цель разработки данного стандарта была определена как создание общего руководства по организации жизненного цикла программного обеспечения для

формирования общего понимания жизненного цикла ПО всеми заинтересованными сторонами и участниками процесса разработки приобретения, поставки, эксплуатации, поддержки и сопровождения программных систем, а также возможности управления, контроля и совершенствования процессов жизненного цикла.

Данный стандарт определяет жизненный цикл как структуру декомпозиции работ. Детализация, техники и метрики проведения работ – вопрос программной инженерии. Организация последовательности работ – модель жизненного цикла. Совокупность моделей, процессов, техник и организации проектной группы задаются методологией. В частности, выбор и применение метрик оценки качества программной системы и процессов находятся за рамками стандарта 12207, а концепция совершенствования процессов рассматривается в стандарте ISO/IEC 15504 «Information Technology – Software Process Assessment» («Оценка процессов <в области> программного обеспечения»).

Необходимо отметить заложенные в стандарте ключевые концепции рассмотрения жизненного цикла программных систем.

Организация стандарта и архитектура жизненного цикла

Стандарт определяет область его применения, дает ряд важных определений (таких, как заказчик, разработчик, договор, оценка, выпуск – релиз, программный продукт, аттестация и т.п.), процессы жизненного цикла и включает ряд примечаний по процессу и вопросам адаптации стандарта.

Стандарт описывает 17 процессов жизненного цикла, распределенных по трем категориям – группам процессов (названия представлены с указанием номеров разделов стандарта, следуя определениям на русском и английском языке, определяемыми [ГОСТ 12207, 1999] и оригинальной версией ISO/IEC 12207, соответственно):

1. **Основные процессы жизненного цикла** – Primary Processes

1. Заказ – Acqusition
2. Поставка – Supply
3. Разработка – Development
4. Эксплуатация – Operation
5. Сопровождение - Maintenance

2. **Вспомогательные процессы жизненного цикла** – Supporting Processes

1. Документирование – Documentation
2. Управление конфигурацией – Configuration Management
3. Обеспечение качества – Quality Assurance
4. Верификация – Verification
5. Аттестация – Validation
6. Совместный анализ – Joint Review
7. Аудит– Audit
8. Решение проблем – Problem Resolution

3. **Организационные процессы жизненного цикла** – Organizational Processes

1. Управление – Management
2. Создание инфраструктуры – Infrastructure
3. Усовершенствование – Improvement
4. Обучение – Training

Стандарт определяет высокоуровневую архитектуру жизненного цикла. Жизненный цикл начинается с идеи или потребности, которую необходимо удовлетворить с использованием программных средств (может быть и не только их). Архитектура строится как набор процессов и взаимных связей между ними. Например, основные процессы жизненного цикла обращаются к вспомогательным

процессам в то время, как организационные процессы действуют на всем протяжении жизненного цикла и связаны с основными процессами.

Дерево процессов жизненного цикла представляет собой структуру декомпозиции жизненного цикла на соответствующие процессы (группы процессов). Декомпозиция процессов строится на основе двух важнейших принципов, определяющих правила разбиения (partitioning) жизненного цикла на составляющие процессы. Эти принципы:

1. **Модульность**

- задачи в процессе являются функционально связанными;
- связь между процессами – минимальна;
- если функция используется более, чем одним процессом, она сама является процессом;
- если Процесс Y используется Процессом X и только им, значит Процесс Y принадлежит (является его частью или его задачей) Процессу X, за исключением случаев потенциального использования Процесса Y в других процессах в будущем.

2. **Ответственность**

- каждый процесс находится под ответственностью конкретного лица (управляется и/или контролируется им), определенного для заданного жизненного цикла, например, в виде роли в проектной команде;
- функция, чьи части находятся в компетенции различных лиц, не может рассматриваться как самостоятельный процесс.

Общая иерархия (декомпозиция) составных элементов жизненного цикла выглядит следующим образом:

- группа процессов
    - процессы
        - работы
            - задачи

Ниже приведен краткий обзор основных процессов жизненного цикла, явно демонстрирующий связь вопросов, касающихся непосредственно самой программной системы, с системными аспектами ее функционирования и обеспечения ее эксплуатации.

Основные процессы жизненного цикла

Приобретение

Процесс приобретения (как его называют в ГОСТ – «заказа») определяет работы и задачи заказчика, приобретающего программное обеспечение или услуги, связанные с ПО, на основе контрактных отношений. Процесс приобретения состоит из следующих работ (названия ГОСТ 12207 даны в скобках, если предлагают другой перевод названий работ оригинального стандарта):

- Initiation – инициирование (подготовка)
- Request-for-proposal preparation – подготовка запроса на предложение (подготовка заявки на подряд)
- Contract preparation and update –подготовка и корректировка договора
- Supplier monitoring – мониторинг поставщика (надзор за поставщиком)
- Acceptance and completion – приемка и завершение (приемка и закрытие договора)

Все работы проводятся в рамках проектного подхода.

Поставка

Процесс поставки, в свою очередь, определяет работы и задачи поставщика. Работы также проводятся с использованием проектного подхода. Процесс включает следующие работы:

- Initiation – инициирование (подготовка)
- Preparation of response – подготовка предложения (подготовка ответа)
- Contract – разработка контракта (подготовка договора)
- Planning – планирование
- Execution and control – выполнение и контроль

- Review and evaluation –проверка и оценка
- Delivery and completion – поставка и завершение (поставка и закрытие договора)

Разработка

Процесс разработки определяет работы и задачи разработчика. Процесс состоит из следующих работ:

- Process implementation – определение процесса (подготовка процесса)
- System requirements analysis – анализ системных требований (анализ требований к системе)
- System design – проектирование системы (проектирование системной архитектуры)
- Software requirements analysis – анализ программных требований (анализ требований к программным средствам)
- Software architectural design – проектирование программной архитектуры
- Software detailed design – детальное проектирование программной системы (техническое проектирование программных средств)
- Software coding and testing – кодирование и тестирование (программирование и тестирование программных средств)
- Software integration – интеграция программной системы (сборка программных средств)
- Software qualification testing – квалификационные испытания программных средств
- System integration – интеграция системы в целом (сборка системы)
- System qualification testing – квалификационные испытания системы
- Software installation – установка (ввод в действие)
- Software acceptance support – обеспечение приемки программных средств

Стандарт отмечает, что работы проводятся с использованием проектного подхода и могут пересекаться по времени, т.е. проводиться одновременно или с наложением, а также могут предполагать рекурсию и разбиение на итерации.

Эксплуатация

Процесс разработки определяет работы и задачи оператора службы поддержки. Процесс включает следующие работы:

- Process implementation – определение процесса (подготовка процесса)
- Operational testing – операционное тестирование (эксплуатационные испытания)
- System operation – эксплуатация системы
- User support – поддержка пользователя

Сопровождение

Процесс разработки определяет работы и задачи, проводимые специалистами службы сопровождения. Процесс включает следующие работы:

- Process implementation – определение процесса (подготовка процесса)
- Problem and modification analysis – анализ проблем и изменений
- Modification implementation – внесение изменений
- Maintenance review/acceptance – проверка и приемка при сопровождении
- Migration – миграция (перенос)
- Software retirement – вывод программной системы из эксплуатации (снятие с эксплуатации)

Важно понимать, что *стандарт 12207 не определяет последовательность и разбиение выполнения процессов во времени,* адресуя этот вопрос также работам по адаптации стандарта к конкретным условиям и окружению и применению выбранных моделей, практик, техник и т.п.

Адаптация стандарта

Адаптация стандарта подразумевает применение требований стандарта к конкретному проекту или проектам, например, в рамках создания внутрикорпоративных регламентов ведения проектов программного обеспечения.

Адаптация включает следующие виды работ:

- Определение исходной информации для адаптации стандарта
- Определение условий выполнения проекта

- Отбор процессов, работ и задач, используемых в проекте или соответствующих регламентах
- Документирование требований, решений и процессов, связанных с адаптацией и полученных в ее результате

Адаптация также подразумевает выбор модели (или комбинации моделей) жизненного цикла, а также применение соответствующих методологий, детализирующих процедуры выполнения процессов, работ и задач в рамках заданных границ (содержания) жизненного цикла программного обеспечения и организационной структуры и ролевой ответственности в конкретной организации (ее подразделении) и/или в проектной группе.

2.3. ISO 15504. Процессы ЖЦ ПО

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

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

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

ISO 15504. Классификация процессов

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

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

ISO 15504. CUS: Потребитель-поставщик

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

Включает следующие процессы:

- CUS.1 Процесс приобретения (Acquisition process)
  - CUS.1.1 Процесс подготовки приобретения (Acquisition preparation process)
  - CUS.1.2 Процесс выбора поставщика (Supplier selection process)
  - CUS.1.3 Процесс мониторинга поставщика (Supplier Monitoring process)
  - CUS.1.4 Процесс приемки (Customer Acceptance process)
- CUS.2 Поставки (Supply process)
- CUS.3 Процесс выявления требований (Requirements process)
- CUS.4 Эксплуатации (Operation process)

CUS.4.1 Процесс эксплуатационного использования (Operational use process)

  - CUS.4.2 Процесс поддержки потребителя (Customer support process)

ISO 15504. ENG: Инженерные процессы

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

Включает следующие процессы:

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

2.1.1.1. ISO 15504. SUP: Вспомогательные процессы

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

Включает следующие процессы:

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

2.1.1.3. ISO 15504. MAN: Управленческие процессы

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

К управленческой категории относятся следующие процессы:

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

2.1.1.4. ISO 15504. ORG: Организационные процессы

Организационная категория процессов состоит из процессов, которые устанавливают цели функционирования организации и создают активы процессов, продуктов и ресурсов, которые, будучи использованы в проектах организации, способствуют выполнению ее целей. Хотя организационные практики в целом

относятся не только к процессам, относящимся к программным средствам, последние выполняются в общем контексте организации, и для их эффективного использования необходимо соответствующее окружение. Кратко, эти организационные процессы:

- создают инфраструктуру организации;
- используют все лучшее из того, что имеется (передовой опыт) во всех частях организации (эффективные процессы, лучшие навыки, качественный программный код, хорошие средства поддержки);
- делают это общедоступным в рамках всей организации;
- создают базу для постоянного совершенствования во всей организации.

Организационной категории принадлежат процессы:

- ORG.1 Процесс организационных установок (Organizational alignment process)
- ORG.2 Процесс усовершенствования (Improvement process)
    - ORG.2.1 Процесс создания процессов (Process establishment process)
    - ORG.2.2 Процесс аттестации процессов (Process assessment process)
    - ORG.2.3 Процесс усовершенствования процессов (Process improvement process)
- ORG.3 Процесс административного управления кадрами (Humanre source managemen tprocess)
- ORG.4 Процесс создания инфраструктуры (Infrastructure process)
- ORG.5 Процесс измерения (Measurement process)
- ORG.6 Процесс повторного использования (Reuse process)

2.4. Модели жизненного цикла

Модель жизненного цикла ПО описывается набор фаз (этапов, стадий) проекта по созданию ПО, в которых выполняются отдельные процессы, разбитые на операции и задачи. В глоссарии PMI даются следующие определения этих понятий:

Жизненный цикл проекта. Набор обычно последовательных фаз проекта, количество и состав которых определяется потребностями управления проектом организацией или организациями, участвующими в проекте.

Фаза проекта. Объединение логически связанных операций проекта, обычно завершающихся достижением одного из основных результатов.

Процесс. Набор взаимосвязанных ресурсов и работ, благодаря которым входные воздействия преобразуются в выходные результаты.

Операция, работа. Элемент работ проекта. У операций обычно имеется ожидаемая длительность, потребность в ресурсах, стоимость. Операции могут далее подразделяться на задачи.

В этих определениях существенным является следующее:

- Состав, количество и, можно добавить, порядок выполнения фаз определяется особенностью проекта.
- Каждая фаза завершается получением одного из основных результатов, в то время как процесс или задача – просто значимого результата.

Для схемы модели жизненного цикла ПО характерно следующее:

- Результатом выполнения каждой фазы является некоторая модель ПО. Описание требований – модель того, что должен делать программный продукт; результат анализа – модель основных архитектурных решений и GUI, …
- Результат выполнения каждой фазы является входом следующей фазы и фазы должны выполняться в определенной моделью ЖЦ последовательности.
- Некоторые процессы могут выполняться на нескольких фазах, некоторые – в пределах одной.

В стандарте ISO 12207 модель жизненного цикла (lifecycle model) определяется как структура, состоящая из процессов, работ и задач, включающих в себя разработку, эксплуатацию и сопровождение программного продукта, охватывающая жизнь системы от установления требований к ней до прекращения

ее использования. При этом конкретные модели определяются особенностью задач, ограничениями на ресурсы, опытом разработчиков и т.д. Между тем, известны некоторые типовые модели ЖЦ ПО, которые проявили себя в определенных условиях, имеют определенные преимущества, недостатки и условия применимости. Эти типовые модели устанавливают некоторые принципы организации модели жизненного цикла ПО.

Наиболее часто говорят о следующих моделях жизненного цикла:

- Каскадная (водопадная) или последовательная
- Итеративная и инкрементальная – эволюционная (гибридная, смешанная)
- Спиральная (spiral) или модель Боэма

Легко обнаружить, что в разное время и в разных источниках приводится разный список моделей и их интерпретация. Например, ранее, инкрементальная модель понималась как построение системы в виде последовательности сборок (релизов), определенной в соответствии с *заранее подготовленным планом* и заданными (уже сформулированными) *и неизменными требованиями*. Сегодня об инкрементальном подходе чаще всего говорят в контексте постепенного наращивания функциональности создаваемого продукта.

Может показаться, что индустрия пришла, наконец, к общей «правильной» модели. Однако, каскадная модель, многократно «убитая» и теорией и практикой, продолжает встречаться в реальной жизни. Спиральная модель является ярким представителем эволюционного взгляда, но, в то же время, представляет собой единственную модель, которая уделяет явное внимание *анализу и предупреждению рисков*. Поэтому, я попытался именно представленным выше образом выделить три модели – каскадную, эволюционную и спиральную. Их мы и обсудим.

Каскадная (водопадная) модель

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

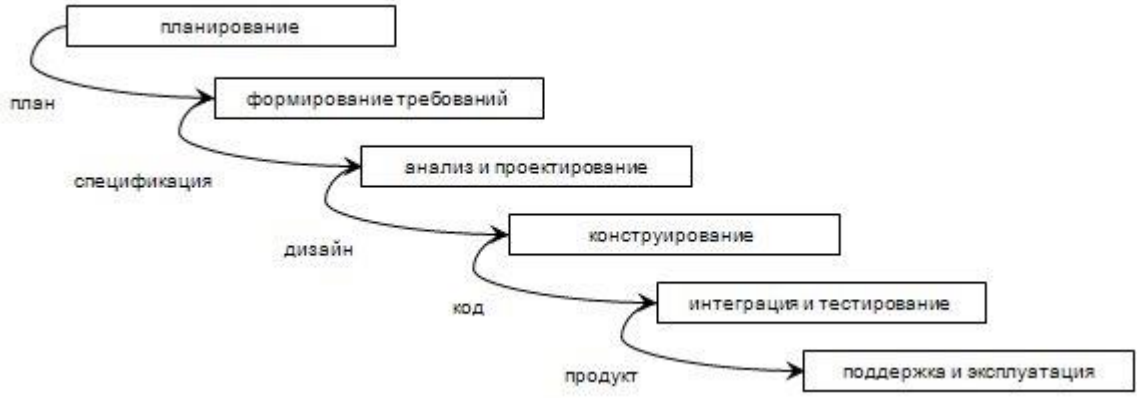


Рисунок 3. Каскадная модель жизненного цикла.

На рисунке 3 изображены типичные фазы каскадной модели жизненного цикла и соответствующие активы проекта, являющиеся для одних фаз выходами, а для других – входами. Данная модель имеет ряд важных аспектов, характерных для водопадной модели: «Водопадная схема включает несколько важных операций, применимых ко всем проектам:

- составление плана действий по разработке системы;
- планирование работ, связанных с каждым действием;
- применение операции отслеживания хода выполнения действий с контрольными этапами.

В связи с тем, что упомянутые задачи являются неотъемлемым элементом всех хорошо управляемых процессов, практически не существует причин, препятствующих утверждению полнофункциональных, классических методов руководства проектом, таких как анализ критического пути и промежуточные контрольные этапы

Будучи активно используема (де факто и, например, в свое время, как часть соответствующего отраслевого стандарта в США), эта модель продемонстрировала свою «проблемность» в подавляющем большинстве ИТ-проектов, за исключением, может быть, отдельных проектов обновления программных систем для критически-важных программно-аппаратных комплексов (например, авионики или медицинского оборудования). Практика показывает, что в реальном мире, особенно в мире бизнес-систем, каскадная модель не должна применяться.

Специфика таких систем (если можно говорить о «специфике» для подавляющего большинства создаваемых систем) – требования характеризуются высокой динамикой корректировки и уточнения, невозможностью четкого и однозначного определения требований до начала работ по реализации (особенно, для новых систем) и быстрой изменчивостью в процессе эксплуатации системы.

Фредерик Брукс во втором издании своего классического труда «Мифический человеко-месяц» так описывает главную беду каскадной модели [Брукс, 1995, с.245]:

«Основное заблуждение каскадной модели состоит в предположениях, что проект проходит через весь процесс *один раз*, архитектура хороша и проста в использовании, проект осуществления разумен, а ошибки в реализации устраняются по мере тестирования. Иными словами, каскадная модель исходит из того, что все ошибки будут сосредоточены в реализации, а потому их устранение происходит равномерно во время тестирования компонентов и системы».

В каскадной модели переход от одной фазы проекта к другой предполагает полную корректность результата (выхода) предыдущей фазы. Однако, например, неточность какого-либо требования или некорректная его интерпретация, в результате, приводит к тому, что приходится «откатываться» к ранней фазе проекта и требуемая переработка не просто выбивает проектную команду из графика, но приводит часто к качественному росту затрат и, не исключено, к прекращению проекта в той форме, в которой он изначально задумывался. Кроме того, эта модель не способна гарантировать необходимую скорость отклика и внесение соответствующих изменений в ответ на быстро меняющиеся потребности пользователей, для которых программная система является одним из инструментов исполнения бизнес-функций. И таких примеров проблем, порождаемых самой природой модели, можно привести достаточно много. Достаточно для чего? Для отказа от каскадной модели жизненного цикла.

Итеративная и инкрементальная модель – эволюционный подход

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

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

Эволюционная модель подразумевает не только сборку работающей (с точки зрения результатов тестирования) версии системы, но и ее развертывание в реальных операционных условиях с анализом откликов пользователей для определения содержания и планирования следующей итерации. «Чистая» инкрементальная модель не предполагает развертывания промежуточных сборок (релизов) системы и все итерации проводятся по заранее определенному плану наращивания функциональности, а пользователи (заказчик) получает только результат финальной итерации как полную версию системы. В идеале, поскольку на каждом шаге мы имеем работающую систему:

- можно очень рано начать тестирование пользователями;
- можно принять стратегию разработки в соответствии с бюджетом, полностью защищающую от перерасхода времени или средств (в частности, за счет сокращения второстепенной функциональности).

Таким образом, Значимость эволюционного подхода на основе организации итераций особо проявляется в снижении неопределенности с завершением каждой итерации. В свою очередь, снижение неопределенности позволяет уменьшить риски. Рисунок 3 иллюстрирует некоторые идеи эволюционного подхода, предполагая, что итеративному разбиению может быть подвержен не только жизненный цикл в целом, включающий перекрывающиеся фазы – формирование требований, проектирование, конструирование и т.п., но и каждая фаза может, в свою очередь, разбиваться на уточняющие итерации, связанные, например, с детализацией структуры декомпозиции проекта – например, архитектуры модулей системы.

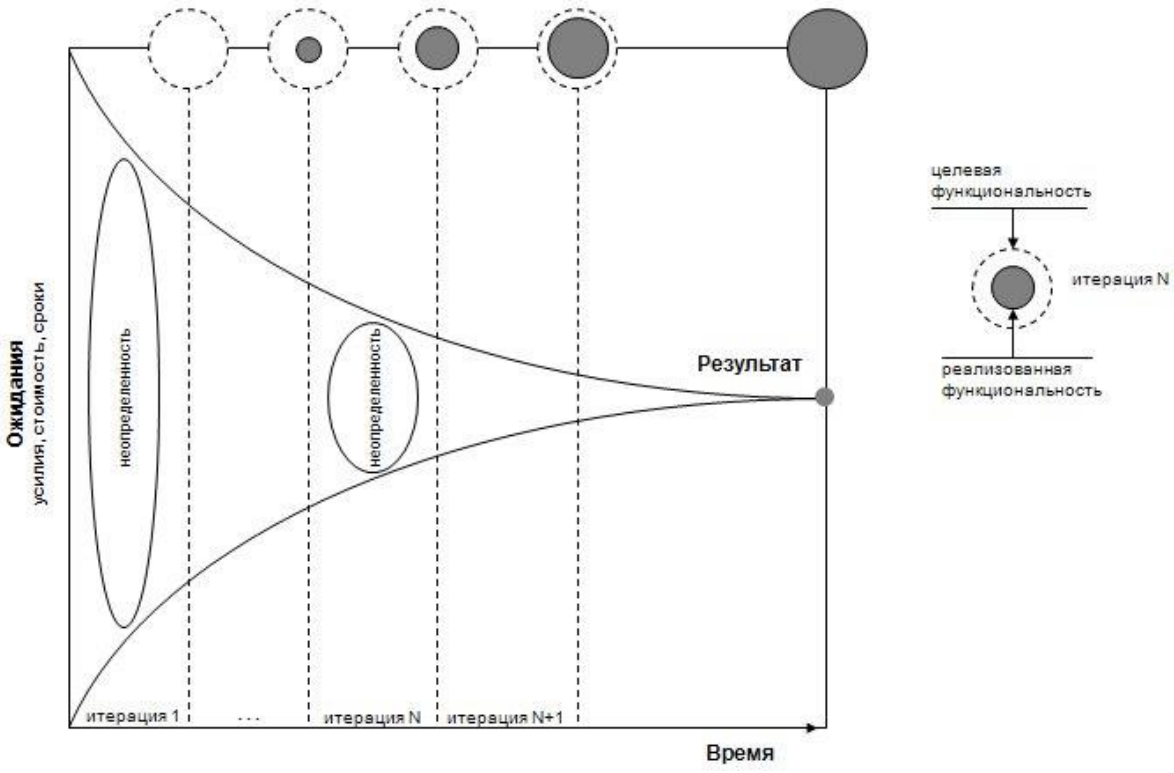


Рисунок 3. Снижение неопределенности и инкрементальное расширение функциональности при итеративной организация жизненного цикла.

Наиболее известным и распространенным вариантом эволюционной модели является *спиральная модель*, ставшая уже по-сути самостоятельной моделью, имеющей различные сценарии развития и детализации.

Спиральная модель

Спиральная модель (представлена на рисунке 4) была впервые сформулирована Барри Боэмом (Barry Boehm) в 1988 году [Boehm, 1988]. Отличительной особенностью этой модели является специальное внимание *рискам*, влияющим на организацию жизненного цикла.

Боэм формулирует «top-10» наиболее распространенных (по приоритетам) рисков (используется с разрешения автора):

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. «Разрыв» в квалификации специалистов разных областей знаний.

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

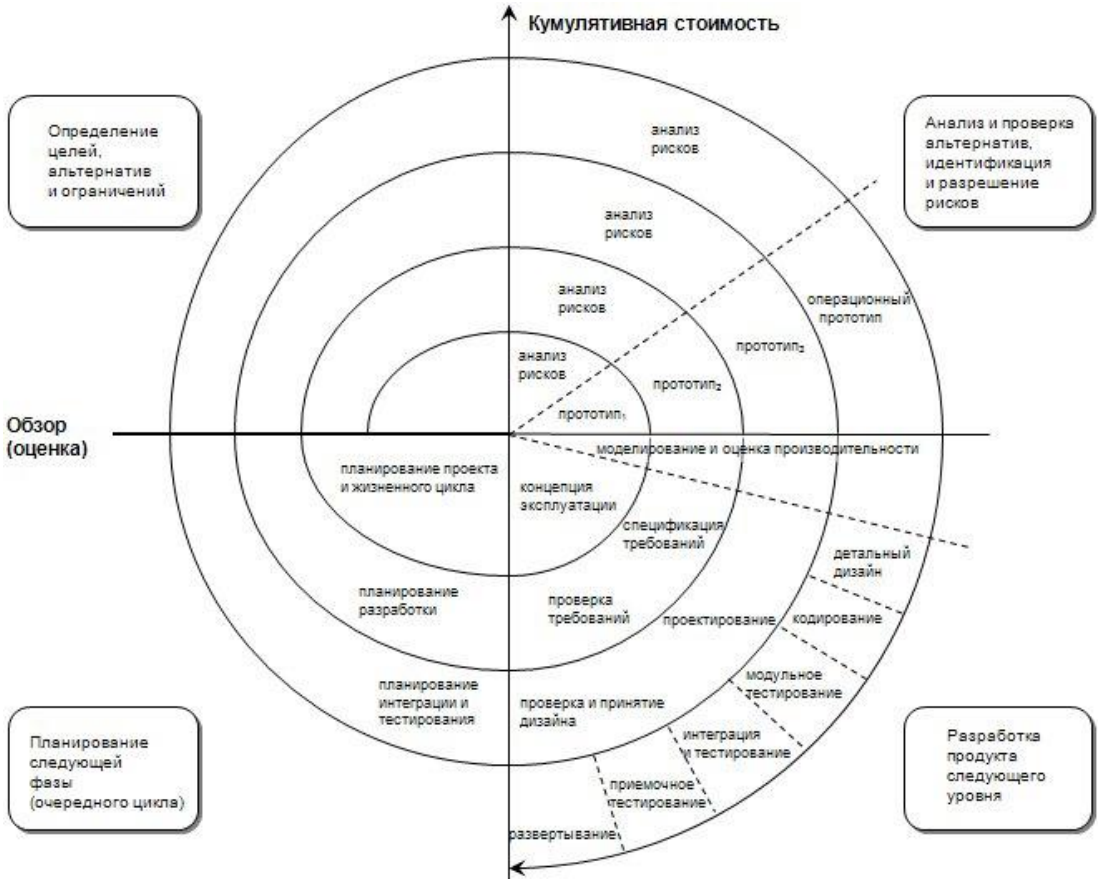


Рисунок 4. Оригинальная спиральная модель жизненного цикла разработки по Боэму.

Главное достижение спиральной модели состоит в том, что она предлагает спектр возможностей адаптации удачных аспектов существующих моделей процессов жизненного цикла. В то же время, ориентированный на риски подход позволяет избежать многих сложностей, присутствующих в этих моделях. В определенных ситуациях спиральная модель становится эквивалентной одной из существующих моделей. В других случаях она обеспечивает возможность наилучшего соединения существующих подходов в контексте данного проекта.

Спиральная модель обладает рядом преимуществ:

Модель уделяет специальное внимание раннему анализу возможностей повторного использования. Это обеспечивается, в первую очередь, в процессе идентификации и оценки альтернатив.

Модель предполагает возможность эволюции жизненного цикла, развитие и изменение программного продукта. Главные источники изменений заключены в

целях, для достижения которых создается продукт. Подход, предусматривающий скрытие информации о деталях на определенном уровне дизайна, позволяет рассматривать различные архитектурные альтернативы так, как если бы мы говорили о единственном проектном решении, что уменьшает риск невозможности согласования функционала продукта и изменяющихся целей (требований).

Модель предоставляет механизмы достижения необходимых параметров качества как составную часть процесса разработки программного продукта. Эти механизмы строятся на основе идентификации всех типов целей (требований) и ограничений на всех «циклах» спирали разработки. Например, ограничения по безопасности могут рассматриваться как риски на этапе специфицирования требований.

Модель уделяет специальное внимание предотвращению ошибок и отбрасыванию ненужных, необоснованных или неудовлетворительных альтернатив на ранних этапах проекта. Это достигается явно определенными работами по анализу рисков, проверке различных характеристик создаваемого продукта (включая архитектуру, соответствие требованиям и т.п.) и подтверждение возможности двигаться дальше на каждом «цикле» процесса разработки.

Модель позволяет контролировать источники проектных работ и соответствующих затрат. По сути речь идет об ответе на вопрос – как много усилий необходимо затратить на анализ требований, планирование, конфигурационное управление, обеспечение качества, тестирование, формальную верификацию и т.д. Модель, ориентированная на риски, позволяет в контексте конкретного проекта решить задачу приложения адекватного уровня усилий, определяемого уровнем рисков, связанных с недостаточным выполнением тех или иных работ.

Модель не проводит различий между разработкой нового продукта и расширением (или сопровождением) существующего. Этот аспект позволяет избежать часто встречающегося отношения к поддержке и сопровождению как ко «второсортной» деятельности. Такой подход предупреждает большого количество

проблем, возникающих в результате одинакового уделения внимания как обычному сопровождению, так и критичным вопросам, связанным с расширением функциональности продукта, всегда ассоциированным с повышенными рисками.

Модель позволяет решать интегрированный задачи системной разработки, охватывающей и программную и аппаратную составляющие создаваемого продукта. Подход, основанный на управлении рисками и возможности своевременного отбрасывания непривлекательных альтернатив (на ранних стадиях проекта) сокращает расходы и одинаково применим и к аппаратной части, и к программному обеспечению.»

Описывая созданную спиральную модель, Боэм обращает внимание на то, что обладая явными преимуществами по сравнению с другими взглядами на жизненный цикл, необходимо уточнить, детализировать шаги, т.е. циклы спиральной модели для обеспечения целостного контекста для всех лиц, вовлеченных в проект.. Организация ролей (ответственности членов проектной команды), детализация этапов жизненного цикла и процессов, определение активов (артефактов), значимых на разных этапах проекта, практики анализа и предупреждения рисков – все это вопросы уже конкретного процессного фреймворка или, как принято говорить, *методологии разработки*.

Действительно, детализация процессов, ролей и активов – вопрос методологии. Однако, рассматривая (спиральная) модель разработки, являясь концептуальным взглядом на создание продукта, требует, как и в любом проекте, *определения ключевых контрольных точек проекта – milestones*. Это, в большой степени, связано с попыткой ответить на вопрос «где мы?». Вопрос, особенно актуальный для менеджеров и лидеров проектов, отслеживающих ход их выполнения и планирующих дальнейшие работы.

В 2000 году [11], представляя анализ использования спиральной модели Боэм формулирует 6 ключевых характеристик или практик, обеспечивающих успешное применение спиральной модели:

1. Параллельное, а не последовательное определение артефактов (активов) проекта
2. Согласие в том, что на каждом цикле уделяется внимание:
    - целям и ограничениям, важным для заказчика
    - альтернативам организации процесса и технологических решений, закладываемых в продукт
    - идентификации и разрешению рисков
    - оценки со стороны заинтересованных лиц (в первую очередь заказчика)
    - достижению согласия в том, что можно и необходимо двигаться дальше
3. Использование соображений, связанных с рисками, для определения уровня усилий, необходимого для каждой работы на всех циклах спирали.
4. Использование соображений, связанных с рисками, для определения уровня детализации каждого артефакта, создаваемого на всех циклах спирали.
5. Управление жизненным циклом в контексте обязательств всех заинтересованных лиц на основе трех контрольных точек:
    - Life Cycle Objectives (LCO)
    - Life Cycle Architecture (LCA)
    - Initial Operational Capability (IOC)
6. Уделение специального внимания проектным работам и артефактам создаваемой системы (включая непосредственно разрабатываемое программное обеспечение, ее окружение, а также эксплуатационные характеристики) и жизненного цикла (разработки и использования).

Эволюционирование спиральной модели, таким образом, связано с вопросами детализации работ. Особенно стоит выделить акцент на большем внимании вопросам уточнения – требований, дизайна и кода, т.е. придание большей важности вопросам итеративности, в том числе, увеличения их количества при сокращении длительности каждой итерации.

В результате, можно определить общий набор контрольных точек в сегодняшней спиральной модели:

- *Concept of Operations (COO)* – концепция <использования> системы;
- *Life Cycle Objectives (LCO)* – цели и содержание жизненного цикла;
- *Life Cycle Architecture (LCA)* – архитектура жизненного цикла; здесь же возможно говорить о готовности концептуальной архитектуры целевой программной системы;
- *Initial Operational Capability (IOC)* – первая версия создаваемого продукта, пригодная для опытной эксплуатации;
- *FinalOperationalCapability (FOC)* – готовый продукт, развернутый (установленный и настроенный) для реальной эксплуатации.

Таким образом, мы приходим к возможному современному взгляду (см., например, представление спиральной модели в [Фатрелл, Шафер и Шафер, 2003, с.159]) на итеративный и инкрементальный – эволюционный жизненный цикл в форме спиральной модели, изображенной на рисунке 5.

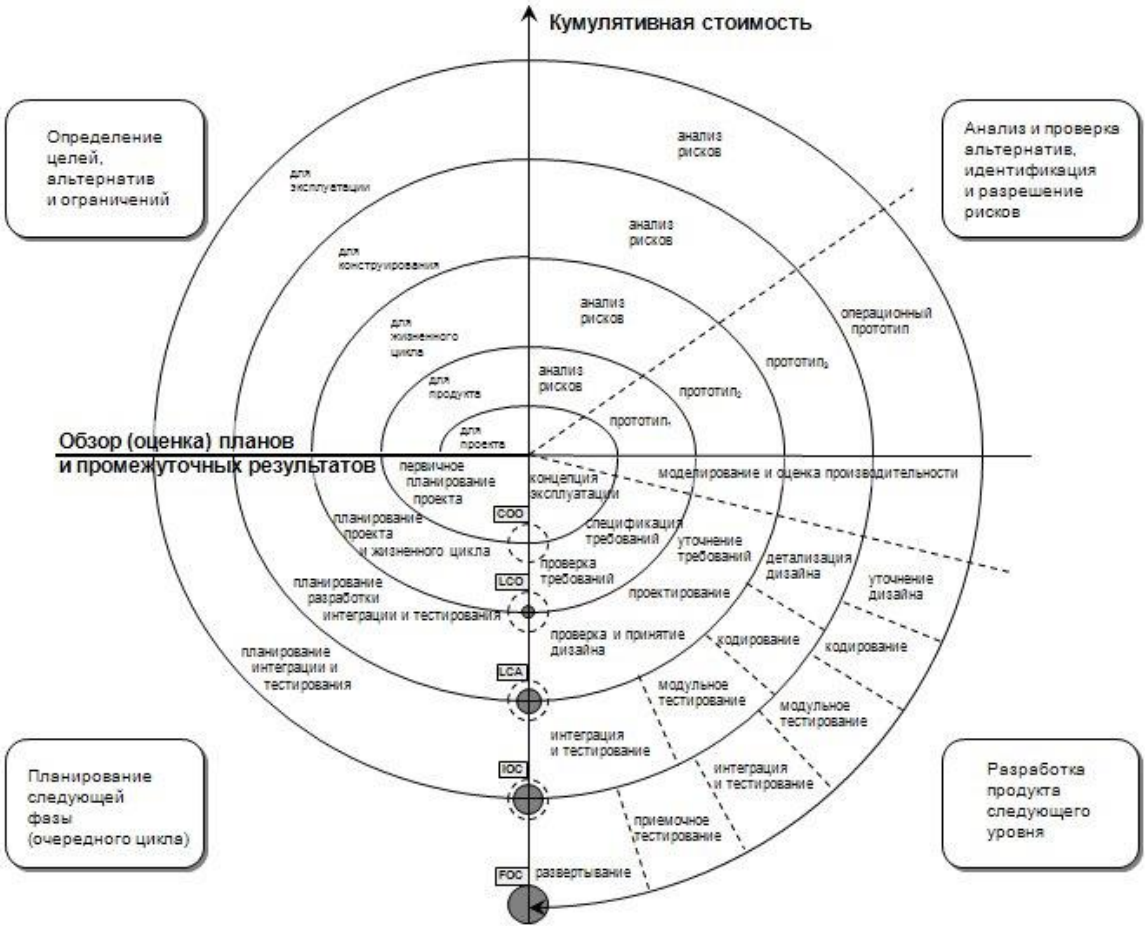


Рисунок 5. Обновленная спиральная модель с контрольными точками проекта.

Похоже, нам удалось более четко и естественно определить контрольные точки проекта, в определенной степени, подчеркнув эволюционную природу жизненного цикла. Теперь же пора взглянуть на жизненный цикл в контексте методологий, не просто детализирующих ту или иную модель, но добавляющих к ним ключевой элемент – *людей*. *Роли*, как представление различных функциональных групп работ, связывает создание, модификацию и использование активов проектов с конкретными участниками проектных команд. В совокупности с *процессами* и *активами* (*артефактами*) они позволяют нам создать целостную и подробную картину жизненного цикла.

Так как взглядов на детализацию описания жизненного цикла может быть много – безусловно, существуют различные методологии, среди которых наибольшее распространение получили:

- Rational Unified Process (RUP)
- Enterprise Unified Process (EUP)
- Microsoft Solutions Framework (MSF) в обоих представлениях: MSF for Agile и MSF for CMMI (анонсированная изначально как «MSF Formal»)
- Agile-практики (eXtreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), SCRUM,...).

2.5. Анализ моделей жизненного цикла программного продукта

Каскадная модель. Принципы

Каскадная модель (водопад – waterfall) включает выполнение следующих фаз:

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

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

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

Основными принципами каскадной модели являются:

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

Каскадная модель. Преимущества и недостатки

Каскадная модель имеет следующие преимущества:

- Проста и понятна заказчикам, т.к. часто используется другими организациями для отслеживания проектов, не связанных с разработкой ПО
- Проста и удобна в применении:
  - процесс разработки выполняется поэтапно.
  - ее структурой может руководствоваться даже слабо подготовленный в техническом плане или – неопытный персонал;
  - она способствует осуществлению строгого контроля менеджмента проекта;

- Каждую стадию могут выполнять независимые команды (все документировано)
- Позволяет достаточно точно планировать сроки и затраты

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

- Попытка вернуться на одну или две фазы назад, чтобы исправить какую-либо проблему или недостаток, приведет к значительному увеличению затрат и сбою в графике;
- Интеграция компонент, на которой обычно выявляется большая часть ошибок, выполняется в конце разработки, что сильно увеличивает стоимость устранения ошибок;
- Запаздывание с получением результатов – если в процессе выполнения проекта требования изменились, то получится устаревший результат

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

Каскадная модель. Применимость

Каскадная модель впервые четко сформулирована в 1970 году Ройсом [8]. На начальном периоде она сыграла ведущую роль как метод регулярной разработки сложного ПО. В семидесятых-восьмидесятых годах XX века модель была принята как стандарт министерства обороны США. Со временем недостатки каскадной модели стали проявляться все чаще и возникло мнение, что она безнадежно устарела. Между тем, каскадная модель не утратила своей актуальности при решении следующих типов задач:

- Требования и их реализация максимально четко определены и понятны; используется неизменяемое определение продукта и вполне понятные технические методики. Это задачи типа:
  - научно-вычислительного характера (пакеты и библиотеки научных программ типа расчета несущих конструкций зданий, мостов);
  - операционные системы и компиляторы;
  - системы реального времени управления конкретными объектами.

Кроме того, каскадная модель применима в условиях:

- Повторная разработка типового продукта (автомати-зированного бухгалтерского учета, начисления зарплаты, …)
- Выпуск новой версии уже существующего продукта, если вносимые изменения вполне определены и управляемы (перенос уже существующего продукта на новую платформу)
- И наконец, принципы каскадной модели находят применение как элементы моделей других типов, о чем речь пойдет ниже.

Спиральная модель. Принципы

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

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

Циклический характер разработки ПО отражен в спиральной модели ЖЦ, описанной Б. Боэмом в 1988 году [9]. Спиральная модель была предложена как альтернатива каскадной модели, учитывающая повторяющийся характер разработки ПО. Основные принципы спиральной модели можно сформулировать следующим образом:

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

Спиральная модель. Схема

Схема работы спиральной модели выглядит следующим образом. Разработка вариантов продукта представляется как набор циклов раскручивающейся спирали. Каждому циклу спирали соответствует такое же количество стадий, как и в модели каскадного процесса. При этом, начальные стадии, связанные с анализом и планированием представлены более подробно с добавлением новых элементов. В каждом цикле выделяются четыре базовые фазы:

- определение целей, альтернативных вариантов и ограничений.
- оценка альтернативных вариантов, идентификация и разрешение рисков.

- разработка продукта следующего уровня.
- планирование следующей фазы.

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

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

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

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

Отметим некоторые особенности спиральной модели:

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

Спиральная модель. Преимущества и недостатки

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

- Более тщательное проектирование (несколько начальных итераций) с оценкой результатов проектирования, что позволяет выявить ошибки проектирования на более ранних стадиях.
- Поэтапное уточнение требований в процессе выполнения итераций, что позволяет более точно удовлетворить требованиям заказчика
- Участие заказчика в выполнении проекта с использованием прототипов программы. Заказчик видит, что и как создается, не выдвигает необоснованных требований, оценивает реальные объемы финансирования.
- Планирование и управление рисками при переходе на следующие итерации позволяет разумно планировать использование ресурсов и обосновывать финансирование работ.
- Возможность разработки сложного проекта «по частям», выделяя на первых этапах наиболее значимые требования.

Основные недостатки спиральной модели связаны с ее сложностью:

- Сложность анализа и оценки рисков при выборе вариантов.
- Сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий)

- Сложность оценки точки перехода на следующий цикл
- Бесконечность модели – на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Спиральная модель. Применимость

Спиральную модель целесообразно применять при следующих условиях:

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

Другие типы моделей ЖЦ ПО

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

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

2.1.1.4. Итерационная модель

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

2.1.1.5. V-образная модель

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

2.1.1.6. Инкрементная (пошаговая) модель

Инкрементная разработка представляет собой процесс поэтапной реализации всей системы и поэтапного наращивания (приращения) функциональных возможностей. На первом шаге необходим полный заранее сформулированный набор требований, которые делятся по некоторому признаку на части. Далее выбирается первая группа требований и выполняется полный проход по каскадной модели. После того, как первый вариант системы, выполняющий первую группу требований сдан заказчику, разработчики переходят к следующему шагу (второму инкременту) по разработке варианта, выполняющего вторую группу требований.

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

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

2.1.1.6. Модель быстрого прототипирования

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

Начало жизненного цикла разработки помещено в центре эллипса. Совместно с пользователем разрабатывается предварительный план проекта на основе предварительных требований. Результат начального планирования –

документ, описывающий в общих чертах примерные графики и результативные данные.

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

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

При разработке производственной версии программы может понадобиться более высокий уровень функциональных возможностей, различные системные ресурсы, необходимых для обеспечения полной рабочей нагрузки, или ограничения во времени. После этого следуют тестирование в предельных режимах, определение измерительных критериев и настройка, а затем, как обычно, функциональное сопровождение.

Вопросы для контроля

1. Что такое жизненный цикл программного продукта?
2. Что такое процесс, действие, задача?
3. Какие типы процессов и конкретные процессы вы запомнили?
4. Что такое модель жизненного цикла ПО?
5. Какие типы моделей вы знаете? В чем их преимущества, недостатки, область применимости?

ЛИТЕРАТУРА

1. Шафер Д., Фатрел Р., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат. Пер. с англ. – М.: Вильямс, 2003. – 1136 с.
2. ГОСТ Р ИСО/МЭК 12207-99. Процессы жизненного цикла программных средств. [Электронный ресурс] – Режим доступа: http://www.staratel.com/iso/InfTech/DesignPO/ISO12207/ISO12207-99/ISO12207.htm.
3. С.Н. Баранов. Управление программным проектом. [Электронный ресурс] – Режим доступа: http://www.exams.icqinfo.ru/edu/tp-part2.rar.
4. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504) ISBN: 5-212-00884-0/ Изд: АйТи, Книга и бизнес. [Электронный ресурс] – Режим доступа: http://www.ntrlab.ru/rus/method/iso15504.
5. Скопин И.Н. Основы менеджмента программных проектов. Лекция №11: Модели жизненного цикла в некоторых реальных методологиях программирования. [Электронный ресурс] – Режим доступа: http://www.intuit.ru/department/se/msd/11/1.html.
6. В. Липаев. Стандарты, регламентирующие жизненный цикл сложных программных комплексов [Электронный ресурс] – Режим доступа: http://www.pcweek.ru/year1998/N24/CP1251/Reviews/chapt1.htm.
7. В. В. Кулямин. Технологии программирования. Компонентный подход. Лекция 2. Жизненный цикл и процессы разработки ПО. МГУ. ВМК. Каф. Системного программирования.
8. W.W. Royce. Managing the Development of Large Software Systems. [Электронный ресурс] – Режим доступа: http://facweb.cti.depaul.edu/jhuang/is553/Royce.pdf

9. Итеративная и инкрементальная разработка: краткая история. [Электронный ресурс] – Режим доступа: http://www.sibinfo.ru/news/03_10_14/iid_history.shtml.
10. Barry Boehm, "A Spiral Model of Software Developmentand Enhancement", IEEE Computer, Vol.21, No.5, 1988. [Электронный ресурс] – Режим доступа: www.computer.org/computer/homepage /misc/Boehm/r5061.pdf.
11. Боэм Б. Инженерное проектирование программного обеспечения. – М.: Радио и связь, 1985. – 512 с.

3. УНИФИЦИРОВАННЫЙ ПРОЦЕСС РАЗРАБОТКИ И ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ

3.1. Модели жизненного цикла MSF, RUP, XP

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

Рассмотрим особенности моделей жизненного цикла трех наиболее известных промышленных технологий:

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

Rational Unified Process и экстремальное программирование являются примерами итеративных процессов, построенных на основе различных

предположений о природе разработки программного обеспечения и, соответственно, достаточно сильно отличаются.

RUP является примером так называемого «тяжелого» процесса, предполагающего поддержку разработки исходного кода ПО большим количеством вспомогательных действий. Примерами подобных действий являются разработка планов, технических заданий, многочисленных проектных моделей, проектной документации, и пр. Основная цель такого процесса — отделить успешные практики разработки и сопровождения ПО от конкретных людей, умеющих их применять. Многочисленные вспомогательные действия дают надежду сделать возможным успешное решение задач по конструированию и поддержке сложных систем с помощью имеющихся работников, не обязательно являющихся суперпрофессионалами.

Для достижения этого выполняется иерархическое пошаговое детальное описание предпринимаемых в той или иной ситуации действий, чтобы можно было научить обычного работника действовать аналогичным образом. В ходе проекта создается много промежуточных документов, позволяющих разработчикам последовательно разбивать стоящие перед ними задачи на более простые. Эти же документы служат для проверки правильности решений, принимаемых на каждом шаге, а также отслеживания общего хода работ и уточнения оценок ресурсов, необходимых для получения желаемых результатов.

Экстремальное программирование, наоборот, представляет так называемые «живые» (agile) методы разработки, называемые также «легкими» процессами. Они делают упор на использовании хороших разработчиков, а не хорошо отлаженных процессов разработки. Живые методы избегают фиксации четких схем действий, чтобы обеспечить б***о***льшую гибкость в каждом конкретном проекте, а также выступают против разработки дополнительных документов, не вносящих непосредственного вклада в получение готовой работающей программы.

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

3.1.1. Модель Microsoft Solution Framework

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

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

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

Основными фазами модели MSF являются:

- Создание общей картины приложения (Envisioning).На этом этапе решаются следующие основные задачи: оценка существующей ситуации; определение состава команды, структуры проекта, бизнес-целей, требований и профилей пользователей; разработка концепции решения и оценка риска. Устанавливаются две промежуточные вехи: "Организован костяк команды" и "Создана общая картина решения".

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

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

3.1.2. Модель Rational Unified Process

Основными фазами RUP являются:

- **Фаза начала проекта (Inception).** Определяются основные цели проекта, бюджет проекта, основные средства его выполнения – технологии, инструменты, ключевой персонал, составляются предварительные планы проекта. Основная цель этой фазы – достичь компромисса между всеми заинтересованными лицами относительно задач проекта.
- **Фаза проработки (Elaboration).** Основная цель этой фазы – на базе основных, наиболее существенных требований разработать стабильную базовую архитектуру продукта, которая позволяет решать поставленные перед системой задачи и в дальнейшем используется как основа разработки системы.
- **Фаза построения (Construction).** Основная цель этой фазы – детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры.
- **Фаза передачи (Transition).** Цель фазы – сделать систему полностью доступной конечным пользователям. Здесь происходит окончательное развертывание системы в ее рабочей среде, подгонка мелких деталей под нужды пользователей.

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

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

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

Поддерживающими деятельностями являются:

- Развертывание (Deployment). Цели – развернуть систему в ее рабочем окружении и оценить ее работоспособность.
- Управление конфигурациями и изменениями (Configuration and Change Management). Определение элементов, подлежащих хранению и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.

- Управление проектом (Project Management). Включает планирование, управление персоналом, обеспечения связей с другими заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.
- Управление средой проекта (Environment). Настройка процесса под конкретный проект, выбор и смена технологий и инструментов, используемых в проекте.

3.2. Унифицированный процесс Rational

Модель жизненного цикла RUP[1,2] является довольно сложной, детально проработанной итеративно-инкрементной моделью с элементами каскадной модели. В модели RUP выделяются 4 основные фазы, 9 видов деятельности (процессов). Кроме того, в модели описывается ряд практик, которые следует применять или руководствоваться для успешного выполнения проекта. RUP ориентирован на поэтапное моделирование создаваемого продукта с помощью языка UML.

Исторически RUP является развитием модели процесса разработки, принятой в компании Ericsson в 70-х–80-х годах XX века. Эта модель была создана Джекобсоном (Ivar Jacobson), впоследствии, в 1987 г., основавшим собственную компанию Objectory AB именно для развития технологического процесса разработки ПО как отдельного продукта, который можно было бы переносить в другие организации. После вливания Objectory в Rational в 1995 г. разработки Джекобсона были интегрированы с работами Ройса (Walker Royce, сын автора «классической» каскадной модели), Крухтена (Philippe Kruchten) и Буча (Grady Booch), а также с развивавшимся параллельно *универсальным языком моделирования (Unified Modeling Language, UML)*.

RUP основан на трех ключевых идеях.

- Весь ход работ направляется итоговыми целями проекта, выраженными в виде ***вариантов использования (use cases)*** – сценариев взаимодействия результирующей программной системы с пользователями или другими

системами, при выполнении которых пользователи получают значимые для них результаты и услуги. Разработка начинается с выделения вариантов использования и на каждом шаге контролируется степенью приближения к их реализации.

- Основным решением, принимаемым в ходе проекта, является ***архитектура*** результирующей программной системы. Архитектура устанавливает набор компонентов, из которых будет построено ПО, ответственность каждого из компонентов (т.е. решаемые им подзадачи в рамках общих задач системы), четко определяет интерфейсы, через которые они могут взаимодействовать, а также способы взаимодействия компонентов друг с другом.
- Архитектура является одновременно основой для получения качественного ПО и базой для планирования работ и оценок проекта в терминах времени и ресурсов, необходимых для достижения определенных результатов. Она оформляется в виде набора графических моделей на языке UML.

Основой процесса разработки являются *планируемые и управляемые итерации*, объем которых (реализуемая в рамках итерации функциональность и набор компонентов) определяется на основе архитектуры.

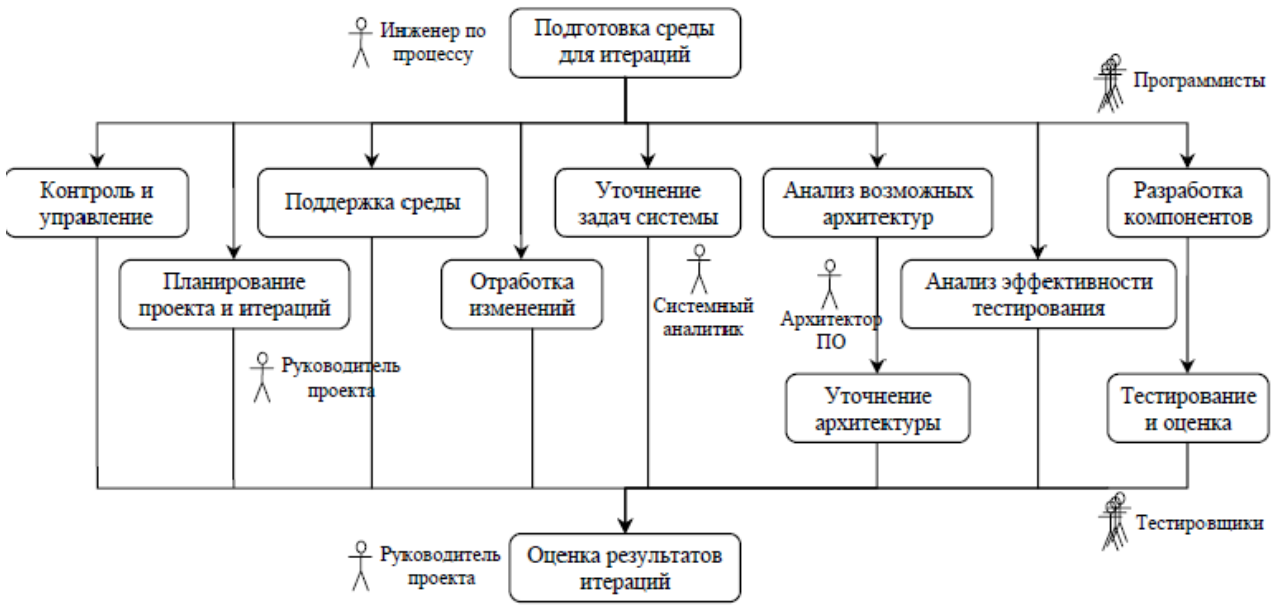


Рис. 3.1. Пример хода работ на фазе начала проекта.

RUP выделяет в жизненном цикле на 4 основные фазы, в рамках каждой из которых возможно проведение нескольких итераций. Кроме того, разработка системы может пройти через несколько циклов, включающих все 4 фазы.

1. Фаза начала проекта (Inception)

Основная цель этой фазы – достичь компромисса между всеми заинтересованными лицами относительно задач проекта и выделяемых на него ресурсов. На этой стадии определяются основные цели проекта, руководитель и бюджет, основные средства выполнения – технологии, инструменты, ключевые исполнители. Также, возможно, происходит апробация выбранных технологий, чтобы убедиться в возможности достичь целей с их помощью, и составляются предварительные планы проекта.На эту фазу может уходить около 10 % времени и 5 % трудоемкости одного цикла. Пример хода работ – на рис. 3.1.

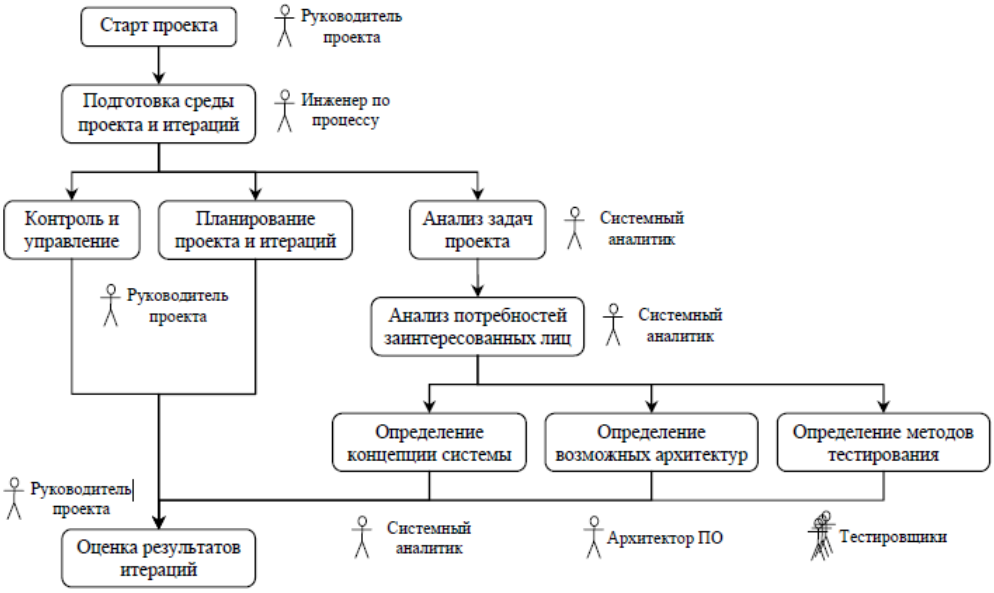


Рис. 3.2. Пример хода работ на фазе проектирования.

2. Фаза проектирования (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.

На эту фазу может уходить около 30 % времени и 20 % трудоемкости одного цикла. Пример хода работ представлен на Рис. 3.2.

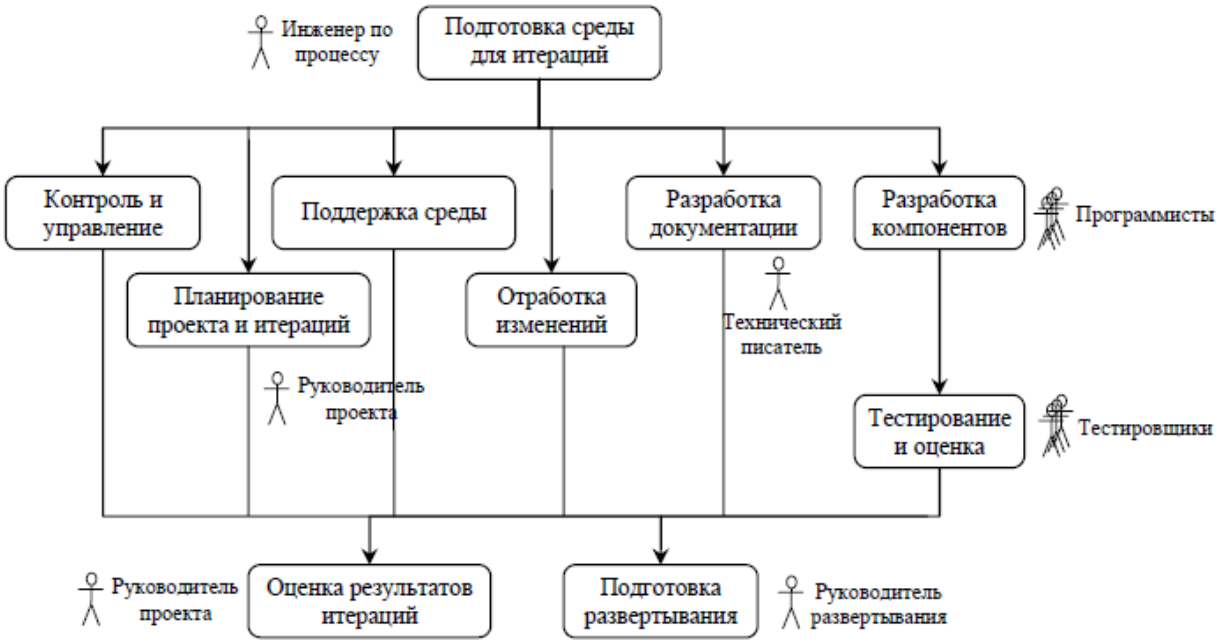


Рис. 3.3. Пример хода работ на фазе построения.

3. Фаза построения (Construction)

Основная цель этой фазы – детальное прояснение требований и разработка системы, удовлетворяющей им, на основе спроектированной ранее архитектуры. В результате должна получиться система, реализующая все выделенные варианты использования. На эту фазу уходит около 50 % времени и 65 % трудоемкости одного цикла. Пример хода работ на этой фазе представлен на Рис. 3.3.

4. Фаза внедрения (Transition)

Цель этой фазы – сделать систему полностью доступной конечным пользователям. На этой стадии происходит развертывание системы в ее рабочей среде, бета-тестирование, подгонка мелких деталей под нужды пользователей.

На эту фазу может уходить около 10 % времени и 10 % трудоемкости одного цикла. Пример хода работ представлен на Рис. 3.4.

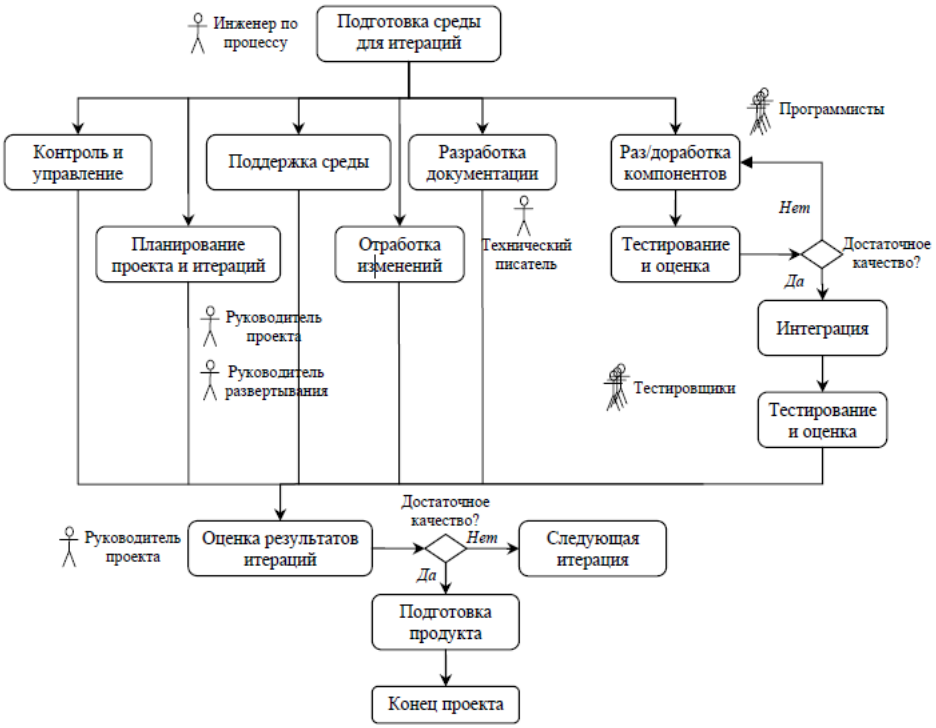


Рис. 3.4. Пример хода работ на фазе внедрения.

Артефакты, вырабатываемые в ходе проекта, могут быть представлены в виде баз данных и таблиц с информацией различного типа, разных видов документов, исходного кода и объектных модулей, а также моделей, состоящих из отдельных элементов. Основные артефакты и потоки данных между ними согласно RUP изображены на рис. 3.4.

Наиболее важные с точки зрения RUP артефакты проекта – это модели , описывающие различные аспекты будущей системы. Большинство моделей представляют собой наборы диаграмм UML. Основные используемые виды моделей следующие.

3.2.1. Модель вариантов использования (Use-Case Model).

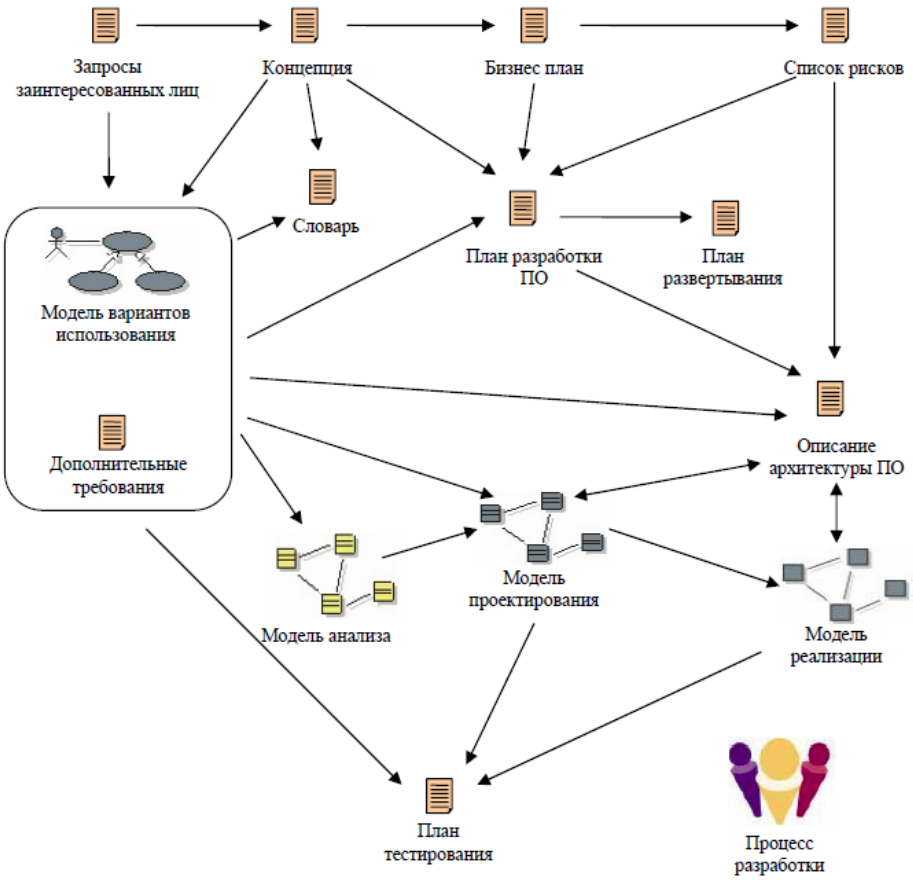
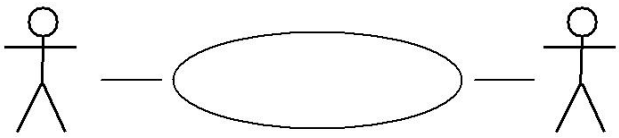


Рис. 3.5. Основные артефакты проекта по RUP и потоки данных между ними.

Эта модель определяет *требования к ПО* – то, что система должна делать – в виде набора вариантов использования. Каждый вариант использования задает сценарий взаимодействия системы с ***действующими лицами (actors)*** или ролями, дающий в итоге значимый для них результат. Действующими лицами могут быть не только люди, но и другие системы, взаимодействующие с рассматриваемой. Вариант использования определяет основной ход событий, развивающийся в нормальной ситуации, а также может включать несколько альтернативных сценариев, которые начинают работать только при специфических условиях.

Модель вариантов использования служит основой для проектирования и оценки готовности системы к внедрению.



Заказ товара

Клиент Оператор заказов

Рис. 3.6. Пример варианта использования и действующих лиц.

Модель анализа (Analysis Model).

Она включает основные классы, необходимые для реализации выделенных вариантов использования, а также возможные связи между классами. Выделяемые классы разбиваются на три разновидности – *интерфейсные, управляющие* и *классы данных*. Эти классы представляют собой набор сущностей, в терминах которых работа системы должна представляться пользователям. Они являются понятиями, с помощью которых достаточно удобно объяснять себе и другим происходящее внутри системы, не слишком вдаваясь в детали.

Интерфейсные классы (boundary classes) соответствуют устройствам или способам обмена данными между системой и ее окружением, в том числе пользователями. ***Классы данных (entity classes)*** соответствуют наборам данных, описывающих некоторые однотипные сущности внутри системы. Эти сущности являются абстракциями представлений пользователей о данных, с которыми работает система. ***Управляющие классы (control classes)*** соответствуют алгоритмам, реализующим какие-то значимые преобразования данных в системе и управляющим обменом данными с ее окружением в рамках вариантов использования.

Каждый класс может играть несколько ролей в реализации одного или нескольких вариантов использования. Каждая роль определяет его обязанности и свойства, тоже являющиеся частью модели анализа.

В рамках других подходов модель анализа часто называется ***концептуальной моделью*** системы. Она состоит из набора классов, совместно реализующих все варианты использования и служащих основой для понимания работы системы и объяснения ее правил.

Модель проектирования (Design Model).

Модель проектирования является детализацией и специализацией модели анализа. Она также состоит из классов, но более четко определенных, с более точным и детальным распределением обязанностей, чем классы модели анализа. Классы модели проектирования должны быть специализированы для конкретной используемой платформы. Каждая такая платформа может включать операционные системы всех вовлеченных машин; используемые языки программирования; интерфейсы и классы конкретных компонентных сред, таких как J2EE, .NET, COM или CORBA; интерфейсы выбранных для использования систем управления базами данных, СУБД, например, Oracle или MS SQL Server; используемые библиотеки разработки пользовательского интерфейса, такие как swing или swt в Java, MFC или gtk; интерфейсы взаимодействующих систем и пр.

Модель реализации (Implementation Model).

Под *моделью реализации* в рамках RUP и UML понимают набор *компонентов* результирующей системы и связей между ними. Под компонентом здесь имеется в виду ***компонент сборки*** – минимальный по размерам кусок кода системы, который может участвовать или не участвовать в определенной ее конфигурации, единица сборки и конфигурационного управления. Связи между компонентами представляют собой зависимости между ними. Если компонент зависит от другого компонента, он не может быть поставлен отдельно от него.

Часто компоненты представляют собой отдельные файлы с исходным кодом. Далее мы познакомимся с компонентами J2EE, состоящими из нескольких файлов.

Модель развертывания (Deployment Model).

Модель развертывания представляет собой набор ***узлов*** системы, являющихся физически отдельными устройствами, которые способны обрабатывать информацию — серверами, рабочими станциями, принтерами, контроллерами датчиков и пр., со *связями* между ними, образованными различного рода сетевыми соединениями. Каждый узел может быть нагружен некоторым множеством компонентов, определенных в модели реализации. Цель построения модели развертывания — определить физическое положение компонентов распределенной системы, обеспечивающее выполнение ею нужных функций в тех местах, где эти функции будут доступны и удобны для пользователей.

В рамках этой модели определяются ***тестовые варианты*** или ***тестовые примеры (test cases)*** и *тестовые процедуры (test scripts)*. Первые являются определенными сценариями работы одного или нескольких действующих лиц с системой, разворачивающимися в рамках одного из вариантов использования. Тестовый вариант включает, помимо входных данных на каждом шаге, где они могут быть введены, условия выполнения отдельных шагов и корректные ответы системы для всякого шага, на котором ответ системы можно наблюдать. В отличие от вариантов использования, в тестовых вариантах четко определены входные данные, и, соответственно, тестовый вариант либо вообще не имеет альтернативных сценариев, либо предусматривает альтернативный порядок действий в том случае, если система может вести себя недетерминировано и выдавать разные результаты в ответ на одни и те же действия. Все другие альтернативы обычно заканчиваются вынесением вердикта о некорректной работе системы.

Тестовая процедура представляет собой способ выполнения одного или нескольких тестовых вариантов и их составных элементов (отдельных шагов и проверок). Это может быть инструкция по ручному выполнению входящих в тестовый вариант действий или программный компонент, автоматизирующий запуск тестов.

RUP также определяет *дисциплины,* включающие различные наборы деятельностей, которые в разных комбинациях и с разной интенсивностью выполняются на разных фазах. В документации по процессу каждая дисциплина сопровождается довольно большой диаграммой, поясняющей действия, которые нужно выполнить в ходе работ в рамках данной дисциплины, артефакты, с которыми надо иметь дело, и роли вовлеченных в эти действия лиц.

Моделирование предметной области (бизнес-моделирование, Business Modeling)

Задачи этой деятельности – понять предметную область или бизнес-контекст, в которых должна будет работать система, и убедиться, что все заинтересованные лица понимают его одинаково, осознать имеющиеся проблемы, оценить их возможные решения и их последствия для бизнеса организации, в которой будет работать система.

В результате моделирования предметной области должна появиться ее модель в виде набора диаграмм классов (объектов предметной области) и деятельностей (представляющих бизнес-операции и бизнес-процессы). Эта модель служит основой модели анализа.

Определение требований (Requirements)

Задачи – понять, что должна делать система, и убедиться во взаимопонимании по этому поводу между заинтересованными лицами, определить границы системы и основу для планирования проекта и оценок затрат ресурсов в нем. Требования принято фиксировать в виде модели вариантов использования.

Анализ и проектирование (Analysis and Design)

Задачи – выработать архитектуру системы на основе требований, убедиться, что данная архитектура может быть основой работающей системы в контексте ее будущего использования.

В результате проектирования должна появиться модель проектирования, включающая диаграммы классов системы, диаграммы ее компонентов, диаграммы взаимодействий между объектами в ходе реализации вариантов использования, диаграммы состояний для отдельных объектов и диаграммы развертывания.

Реализация (Implementation)

Задачи – определить структуру исходного кода системы, разработать код ее компонентов и протестировать их, интегрировать систему в работающее целое.

Тестирование (Test)

Задачи – найти и описать дефекты системы (проявления недостатков ее качества), оценить ее качество в целом, оценить выполнены или нет гипотезы, лежащие в основе проектирования, оценить степень соответствия системы требованиям.

Развертывание (Deployment)

Задачи – установить систему в ее рабочем окружении и оценить ее работоспособность на том месте, где она должна будет работать.

Управление конфигурациями и изменениями (Configuration and Change Management)

Задачи – определение элементов, подлежащих хранению в репозитории проекта и правил построения из них согласованных конфигураций, поддержание целостности текущего состояния системы, проверка согласованности вносимых изменений.

Управление проектом (Project Management)

Задачи – планирование, управление персоналом, обеспечение взаимодействия на благо проекта между всеми заинтересованными лицами, управление рисками, отслеживание текущего состояния проекта.

Управление средой проекта (Environment)

Задачи – подстройка процесса под конкретный проект, выбор и замена технологий и инструментов, используемых в проекте.

Первые пять дисциплин считаются рабочими, остальные – поддерживающими. Распределение объемов работ по дисциплинам в ходе проекта выглядит, согласно руководству по RUP, примерно так, как показано на Рис. 3.7.

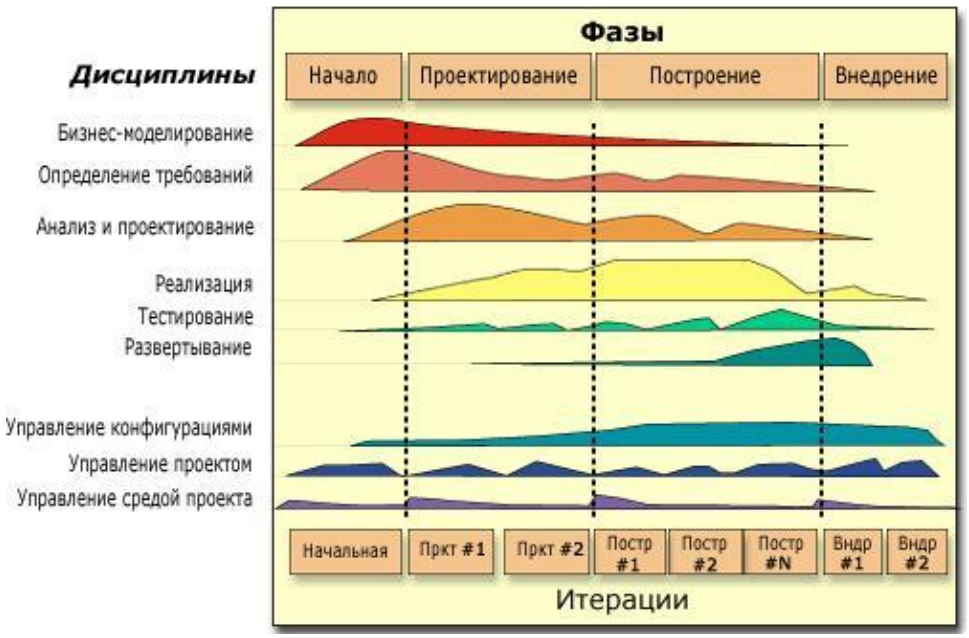


Рис. 3.7. Распределение работ между различными дисциплинами в проекте по RUP.

Перечислим техники, используемые в RUP согласно [3].

- Выработка концепции проекта (project vision) в его начале для четкой постановки задач.

- Управление по плановому снижению рисков и отслеживание их последствий, как можно более раннее начало работ по преодолению рисков.
- Тщательное экономическое обоснование всех действий – делается только то, что нужно заказчику и не приводит к невыгодности проекта.
- Как можно более раннее формирование базовой архитектуры.
- Использование компонентной архитектуры.
- Прототипирование, инкрементная разработка и тестирование.
- Регулярные оценки текущего состояния.
- Управление изменениями, постоянная отработка изменений извне проекта.
- Нацеленность на создание продукта, работоспособного в реальном окружении.
- Нацеленность на качество.
- Адаптация процесса под нужды проекта.

3.3. Экстремальное программирование

Экстремальное программирование (Extreme Programming, XP) [4] возникло как эволюционный метод разработки ПО «снизу-вверх». Этот подход представляет так называемый метод *«живой» разработки (Agile Development Method)*. В группу «живых» методов входят, помимо экстремального программирования, методы SCRUM, DSDM (Dynamic Systems Development Method, метод разработки динамичных систем), Feature-Driven Development (разработка, управляемая функциями системы) и др.

Схема модели XP

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

- «Вброс» архитектуры – начальный этап проекта, на котором создается видение продукта, принимаются основные решения по архитектуре и

применяемым технологиям. Результатом начального этапа является метафора (metaphor) системы, которая в достаточно простом и понятном команде виде должна описывать основной механизм работы системы.

- Истории использования (UserStory) – этап сбора требований, записываемых на специальных карточках в виде сценариев выполнения отдельных функций. Истории использования являются требованиями для планирования очередной версии и одновременной разработки приемочных тестов (Acceptancetests) для ее проверки.
- Планирование версии (релиза). Проводится на собрании с участием заказчика путем выбора UserStories, которые войдут в следующую версию. Одновременно принимаются решения, связанные с реализацией версии. Цель планирования - получение оценок того, что и как можно сделать за 1-3 недели создания следующей версии продукта.
- Разработка проводится в соответствии с планом и включает только те функции, которые были отобраны на этапе планирования.
- Тестирование проводится с участием заказчика, который участвует в составлении тестов.
- Выпуск релиза – разработанная версия передается заказчику для использования или бета-тестирования.

По завершению цикла делается переход на следующую итерацию разработки

Extreme Programming. Принципы

Основные принципы «живой» разработки ПО зафиксированы в манифесте «живой» разработки [5], появившемся в 2000 году:

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

«Живые» методы появились как протест против чрезмерной бюрократизации разработки ПО, обилия побочных, не являющихся необходимыми для получения конечного результата документов, которые приходится оформлять при проведении проекта в соответствии с большинством «тяжелых» процессов, дополнительной работы по поддержке фиксированного процесса организации, как это требуется в рамках, например, CMM. Большая часть таких работ и документов не имеет прямого отношения к разработке ПО и обеспечению его качества, а предназначена для соблюдения формальных пунктов контрактов на разработку, получения и подтверждения сертификатов на соответствие различным стандартам.

«Живые» методы позволяют большую часть усилий разработчиков сосредоточить, собственно, на задачах разработки и удовлетворения реальных потребностей пользователей. Отсутствие кипы документов и необходимости поддерживать их в связном состоянии позволяет более быстро и качественно реагировать на изменения в требованиях и в окружении, в котором придется работать будущей программе.

Тем не менее, XP имеет свою схему процесса разработки (хотя, вообще говоря, широко используемое понимание «процесса разработки» как достаточно жесткой схемы действий противоречит самой идее «живости» разработки), приведенную на рис. 15.

По утверждению авторов XP, эта методика представляет собой не столько следование каким-то общим схемам действий, сколько применение комбинации следующих техник. При этом каждая техника важна, и без ее использования разработка считается идущей не по XP, согласно утверждению Кента Бека (Kent Beck) [4], одного из авторов этого подхода, наряду с Уордом Каннингемом (Ward Cunningham), и Роном Джефрисом (Ron Jeffries).

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

Живое планирование (planning game)

Его задача – как можно быстрее определить объем работ, которые нужно сделать доследующей версии ПО. Решение принимается, в первую очередь, на основе приоритетов заказчика (т.е. его потребностей, того, что нужно ему от системы для более успешного ведения своего бизнеса) и, во вторую, на основе технических оценок (т.е. оценок трудоемкости разработки, совместимости с остальными элементами системы и пр.). Планы изменяются, как только они начинают расходиться с действительностью или пожеланиями заказчика.

Рис. 3.8. Схема потока работ в XP.

Частая смена версий (small releases)

Самая первая работающая версия должна появиться как можно быстрее и тут же должна начать использоваться. Следующие версии подготавливаются через достаточно короткие промежутки времени (от нескольких часов при небольших изменениях в небольшой программе, до месяца-двух при серьезной переработке крупной системы).

Метафора (metaphor) системы

Метафора в достаточно простом и понятном команде виде должна описывать основной механизм работы системы. Это понятие напоминает архитектуру, но

должно гораздо проще, всего в виде одной-двух фраз описывать основную суть принятых технических решений.

Простые проектные решения (simple design)

В каждый момент времени система должна быть сконструирована настолько просто, насколько это возможно. Не надо добавлять функции заранее – только после явной просьбы об этом. Вся лишняя сложность удаляется, как только обнаруживается.

Разработка на основе тестирования (test-driven development)

Разработчики сначала пишут тесты, потом пытаются реализовать свои модули так, чтобы тесты срабатывали. Заказчики заранее пишут тесты, демонстрирующие основные возможности системы, чтобы можно было увидеть, что система действительно заработала.

Постоянная переработка (refactoring)

Программисты постоянно перерабатывают систему для устранения излишней сложности, увеличения понятности кода, повышения его гибкости, но без изменений в его поведении, что проверяется прогоном после каждой переделки тестов. При этом предпочтение отдается более элегантным и гибким решениям, по сравнению с просто дающими нужный результат. Неудачно переработанные компоненты должны выявляться при выполнении тестов и откатываться к последнему целостному состоянию (вместе с зависимыми от них компонентами).

Программирование парами (pair programming)

Кодирование выполняется двумя программистами на одном компьютере. Объединение в пары произвольно и меняется от задачи к задаче. Тот, в чьих руках клавиатура, пытается наилучшим способом решить текущую задачу. Второй программист анализирует работу первого и дает советы, обдумывает последствия тех или иных решений, новые тесты, менее прямые, но более гибкие решения.

Коллективное владение кодом (collective ownership)

В любой момент любой член команды может изменить любую часть кода. Никто не должен выделять свою собственную область ответственности, вся команда в целом отвечает за весь код.

Постоянная интеграция (continuous integration)

Система собирается и проходит интеграционное тестирование как можно чаще, по несколько раз в день, каждый раз, когда пара программистов оканчивает реализацию очередной функции.

40-часовая рабочая неделя

Сверхурочная работа рассматривается как признак больших проблем в проекте. Не допускается сверхурочная работа 2 недели подряд – это истощает программистов и делает их работу значительно менее продуктивной.

Включение заказчика в команду (on-site customer)

В составе команды разработчиков постоянно находится представитель заказчика, который доступен в течение всего рабочего дня и способен отвечать на все вопросы о системе. Его обязанностью являются достаточно оперативные ответы на вопросы любого типа, касающиеся функций системы, ее интерфейса, требуемой производительности, правильной работы системы в сложных ситуациях, необходимости поддерживать связь с другими приложениями и пр.

Использование кода как средства коммуникации

Код рассматривается как важнейшее средство общения внутри команды. Ясность кода – один из основных приоритетов. Следование стандартам кодирования, обеспечивающим такую ясность, обязательно. Такие стандарты, помимо ясности кода, должны обеспечивать минимальность выражений (запрет на дублирование кода и информации) и должны быть приняты всеми членами команды.

Открытое рабочее пространство (open workspace)

Команда размещается в одном, достаточно просторном помещении, для упрощения коммуникации и возможности проведения коллективных обсуждений при планировании и принятии важных технических решений.

Изменение правил по необходимости (just rules)

Каждый член команды должен принять перечисленные правила, но при возникновении необходимости команда может поменять их, если все ее члены пришли к согласию по поводу этого изменения.

Как видно из применяемых техник, XP рассчитано на использование в рамках небольших команд (не более 10 программистов), что подчеркивается и авторами этой методики. Больший размер команды разрушает необходимую для успеха простоту коммуникации и делает невозможным применение многих перечисленных приемов.

Достоинствами XP, если его удается применить, является большая гибкость, возможность быстро и аккуратно вносить изменения в ПО в ответ на изменения требований и отдельные пожелания заказчиков, высокое качество получающегося в результате кода и отсутствие необходимости убеждать заказчиков в том, что результат соответствует их ожиданиям.

Недостатками этого подхода являются невыполнимость в таком стиле достаточно больших и сложных проектов, невозможность планировать сроки и трудоемкость проекта на достаточно долгую перспективу и четко предсказать результаты длительного проекта в терминах соотношения качества результата и затрат времени и ресурсов. Также можно отметить неприспособленность XP для тех случаев, в которых возможные решения не находятся сразу на основе ранее полученного опыта, а требуют проведения предварительных исследований. XP как совокупность описанных техник впервые было использовано в ходе работы над проектом C3 (Chrysler Comprehensive Compensation System, разработка системы учета выплат работникам компании Daimler Chrysler). Из 20-ти участников этого проекта 5 (в том числе упомянутые выше 3 основных автора XP) опубликовали еще во время самого проекта и в дальнейшем 3 книги и огромное количество статей, посвященных XP. Этот проект неоднократно упоминается в различных источниках как пример использования этой методики [6,7,8]. Приведенные ниже данные собраны на основе упомянутых статей [9], за вычетом не подтверждающихся

сведений, и иллюстрируют проблемы некоторых техник XP при их применении в достаточно сложных проектах.

Проект стартовал в январе 1995 года. С марта 1996 года, после включения в него Кента Бека, он проходил с использованием XP. К этому времени он уже вышел за рамки бюджета и планов поэтапной реализации функций. Команда разработчиков была сокращена, и в течение примерно полугода после этого проект развивался довольно успешно. В августе 1998 года появился прототип, который мог обслуживать около 10000 служащих. Первоначально предполагалось, что проект завершится в середине 1999 года и результирующее ПО будет использоваться для управления выплатами 87000 служащим компании. Он был остановлен в феврале 2000 года после 4-х лет работы по XP в связи с полным несоблюдением временных рамок и бюджета. Созданное ПО ни разу не использовалось для работы с данными о более чем 10000 служащих, хотя было показано, что оно справится с данными 30000 работников компании. Человек, игравший роль включенного в команду заказчика в проекте, уволился через несколько месяцев такой работы, не выдержав нагрузки, и так и не получил адекватной замены до конца проекта.

ЛИТЕРАТУРА

1. Ройс У. Управление проектами по созданию программного обеспечения. М.: Лори, 2002. – 140 с.
2. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 320 с.
3. The Spirit of the RUP. [Электронный ресурс] – Режим доступа: http://www06.ibm.com/developer works/ rational/library/content/ RationalEdge / dec01/ TheSpiritoftheRUPDec01.pdf
4. К. Бек. Экстремальное программирование. СПб.: Питер, 2002.
5. http://www.agilemanifesto.org/
6. Beck K., et. al. Chrysler goes to «Extremes». Distributed Computing, 10/1998. p.462 – 479..
7. Cockburn C. Selecting a Project's Methodology. IEEE Software, 04/2000. – p. 130- 148/
8. L. Williams, R. R. Kessler, W. Cunningham, R. Jeffries. Strengthening the Case for Pair Programming. IEEE Software 4/2000. – p. 89-103.
9. Keefer G.. Extreme Programming Considered Harmful for Reliable Software Development. AVOCA Technical Report, 2002.
10. Экстремальное программирование. [Электронный ресурс] – Режим доступа: http://www.avoca-vsm.com/Dateien-Download/ ExtremeProgramming.pdf.
11. Модель процессов MSF, версия 3.1. . [Электронный ресурс] – Режим доступа: http://www.microsoft.com/Rus/Download.aspx? file=/Msdn/Msf/MSF_process_model_rus.doc
12. Колесов А. Введение в методологию Microsoft Solutions Framework. [Электронный ресурс] – Режим доступа: http://www.bytemag.ru/Article.asp?ID=2866
13. Фаулер М. Новые методологии программирования. [Электронный ресурс] – Режим доступа: http://www.maxkir. com/sd/ newmethRUS.html.

14. Extreme Programming. [Электронный ресурс] – Режим доступа: http://www.xprogramming.ru/index.html.
15. Рассел А. Модели жизненного цикла высокотехнологичных проектов [Электронный ресурс] – Режим доступа: http://www. pmprofy.ru/content/rus/107/1073-article.asp
16. Скопин И.Н. Основы менеджмента программных проектов. Лекция №11: Модели жизненного цикла в некоторых реальных методологиях программирования. [Электронный ресурс] – Режим доступа: http://www.intuit.ru/department/se/msd/11/1.html.

4. ТРЕБОВАНИЯ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

4.1. Методологии определения требований в программной инженерии

Каждая программная система представляет собой определенный преобразователь данных, поведение и свойства которого определяются в процессе создания этой системы, ориентированной на решение некоторой проблемы. К программной системе предъявляются требования в виде соглашений между заказчиком и исполнителем.

В общем случае под требованиями к ПС понимают свойства, которыми должна обладать эта система для адекватного выполнения предписанных ей функций. Примерами таких функций могут быть автоматизация присущих персоналу обязанностей, обеспечение руководства организацией информацией, необходимой для принятия решений и др. Т.е. программная система может моделировать достаточно сложную деятельность людей и их организацию, их взаимодействие как между субъектами автоматизируемой области, так с физическим оборудованием компьютерной системы и т.п. [1-5].

4.1.1. Определение понятий и видов требований

Одна из проблем индустрии программного обеспечения – это отсутствие общепринятых определений терминов, которыми пользуются для описания: требований пользователя, требований к ПО, функциональных требований, системных требований, технологических требований, требований к продукту и бизнес-требований. В разных источниках понятия требований определяются, исходя из разных условий и взглядов на то, что по ним создается. Назовем ряд определений в проблематике требований [2,3].

Требования – это «нечто такое, что приводит к выбору дизайна системы".

Требования – это свойства, которыми должен обладать продукт, чтобы представлять какую-то ценность для пользователей.

Требования – это спецификация того, что должно быть реализовано. В них охарактеризовано описание поведения системы, ее свойства и атрибуты. Они могут

быть ограничены процессом разработки системы.

Согласно международному глоссарию по терминологии [6] требования включают описание:

1) условий или возможностей, необходимых пользователю для решения поставленных проблем или достижения целей;

2) условий или возможностей, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворить стандартам, спецификациям или другим формальным документам;

3) документированное представление условий или возможностей проектирования системы.

Виды требований

Требования к продукту охватывают требования как пользователей (внешнее поведение системы), так и разработчиков (некоторые скрытые параметры). Термин *пользователи* относится ко всем заинтересованным лицам в создании системы.

Требования к ПО состоят из трех уровней – бизнес-требования, требования пользователей и функциональные требования. Каждая система имеет свои нефункциональные требования.

Требования пользователей (user requirements) описывают цели и задачи, которые пользователям позволит решить система. К способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие – отклик».

Системные требования (system requirements) обозначают высокоуровневые требования к продукту, которые содержат многие подсистемы или вся система. Из требований для всей системы главными являются функциональные требования к ПО.

Функциональные требования включают описание требований к видам и типам реализуемых функций и документируются в *спецификации требований к ПО* (software requirements specification, SRS), где описано и ожидаемое поведение системы. Спецификация требований к ПО используется при разработке,

тестировании, гарантии качества продукта, управлении проектом и его функциями. В дополнение к функциональным требованиям спецификация содержит нефункциональные требования (защита данных, адаптивность, изменчивость и др.), где описаны цели и атрибуты качества.

Атрибуты качества (quality attributes) представляют собой дополнительное описание функций программного продукта, выраженных через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся легкость и простота использования, легкость перемещения, целостность, эффективность и устойчивость к сбоям. Другие нефункциональные требования описывают внешние взаимодействия между системой и внешним миром, а также ограничения дизайна и реализации. *Ограничения* (constraints) касаются выбора возможности разработки внешнего вида и структуры продукта.

Бизнес-требования (business requirements) содержат высокоуровневые цели организации или заказчиков бизнес–системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей и отдел маркетинга. *Бизнес-правила* (business rules) включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Они не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения на выполнение конкретных вариантов использования или на функции системы, подчиняющимся соответствующим правилам. Бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности.

4.1.2. Анализ и сбор требований

В современных информационных технологиях процесс ЖЦ, на котором фиксируются требования на разработку ПО системы, является определяющим для задания функций, сроков и стоимости работ, а также показателей качества и др.

Анализ и сбор требований является довольно нетривиальной задачей,

поскольку в реальных проектах приходится сталкиваться с такими общими трудностями:

– требования не всегда очевидны и могут исходить из разных источников, их не всегда удается ясно выразить словами;

– имеется много различных типов требований на различных уровнях детализации и их число может стать большим, требующим ими управлять;

требования связаны друг с другом, а также с процессами разработки ПС и постоянно меняются.

Требования имеют уникальные свойства или значения свойств. Например, они не являются одинаково важными и простыми для согласования. Аналитик системы, который занимается анализом и составлением требований, должен иметь определенные знания ПрО и навыки, чтобы справиться с этими трудностями. Он должен уметь:

– анализировать проблему,

– понимать потребности заказчика и пользователей,

– определять функции системы и требования к ним,

– управлять контекстом проекта и изменением требований.

В требованиях к ПС должны отображаться проблемы системы и фиксироваться реальные потребности заказчика, касающиеся функциональных, операционных и сервисных возможностей разрабатываемой системы. В результате создается договор между заказчиком и исполнителем системы на ее создание.

Здесь цена ошибок и нечетких неоднозначных формулировок очень высока, поскольку время и средства могут расходоваться на ненужную заказчику систему или программу. Для внесения изменений в требования может потребоваться повторное проектирование и, соответственно, перепрограммирование всей системы или отдельных ее частей. Как утверждает статистика, процент ошибок в постановке требований и в определении задач системы превышает процент ошибок кодирования. Это является следствием субъективного характера процесса формулирования требований и отсутствия способов его формализации. К примеру,

в США ежегодно расходуется до 82 млрд. долл. на проекты, признанные после реализации не соответствующими требованиям заказчиков.

Существуют стандарты на разработку требований на систему и документы, в которых фиксируются результаты создания программного, технического, организационного и др. видов обеспечения автоматизированных систем (АС) на этапах жизненного цикла, включающие. В приложении 2, 3 дается краткое описание ГОСТ 34.601-90 «Стадии и этапы разработки АС» и ГОСТ 34.201-89 «Документация на разработку АС».

В процессе формулирования требований на систему принимают участие:

– представители от заказчика из нескольких профессиональных групп;

– операторы, обслуживающие систему;

– разработчики системы.

Процесс формулирования требований состоит из нескольких подпроцессов.

Сбор требований

Источниками сведений о требованиях могут быть:

– цели и задачи системы, которые формулирует заказчик. Для однозначного их понимания разработчику системы необходимо их тщательно осмыслить и согласовать с заказчиком;

– действующая система или коллектив, выполняющий ее функции. Система, которую заказывают, может заменять собою старую систему, переставшую удовлетворять заказчика или действующий персонал. Изучение и фиксация ее функциональных возможностей дает основу для учета имеющегося опыта и формулирования новых требований к системе. При этом имеется определенная опасность перенесения недостатков старой системы в новую, поэтому нужно уметь отделить новые требования к реализуемой проблеме от заложенных неудачных решений в старой системе.

Таким образом, требования к системе формулируются исходя из:

– знаний заказчика относительно проблемной области, формулирующего свои проблемы в терминах понятий этой области;

– ведомственных стандартов заказчика и требований к среде функционирования будущей системы, ее исполнительских и ресурсных возможностей.

Методами сбора требований являются:

– интервью с носителями интересов заказчика и операторами;

– наблюдение за работой действующей системы с целью отделения ее проблемных свойств от тех, которые обусловлены структурою кадров;

– сценарии (примеров) возможных случаев выполнении ее функций, ролей лиц, запускающих эти сценарии или взаимодействующих с системой во время ее функционирования.

Продукт процесса сбора требований – неформализованное их описание – основа контракта на разработку между заказчиком и исполнителем системы. Такое описание является входом для следующего процесса инженерии требований – анализа требований. Исполнитель этого процесса выполняет дальнейшее уточнение и формализацию требований, а также их документирование в нотации, понятной коллективу разработчиков системы.

Анализ требований

Это процесс изучения потребностей и целей пользователей, классификация и их преобразование к требованиям системы, к ПО, установление и разрешение конфликтов между требованиями, определение приоритетов, границ системы и принципов взаимодействия со средой функционирования. Требования классифицируются по функциональному и нефункциональному принципу, а также по отношению их к внешней или внутренней стороне системы.

Функциональные требования относятся к заданию функций системы или ее ПО, к способам поведения ПО в процессе выполнения функций и методам преобразования входных данных в результаты.

Нефункциональные требования определяют условия и среду выполнения функций (например, защита и доступ к БД, секретность, взаимодействие компонентов функций и др.).

Разработанные требования специфицируются и отражается в специальном документе, по которому проводится ***с****огласование требований* для достижения взаимопонимания между заказчиком и разработчиком.

Функциональные требования отвечает на вопрос «что делает система», а нефункциональные требования определяют характеристики ее работы (вероятность сбоя системы, защита данных и др.). К основным нефункциональным требованиям, существенным для большинства ПС и выражающих ограничения, актуальные для многих проблемных областей относятся:

– конфиденциальность;

– отказоустойчивость;

– несанкционированный доступ к системе;

– безопасность и защита данных;

– время ожидания ответа на обращение к системе;

– свойства системы (ограничение на память, скорость реакции при обращении к 0системе и т. п.).

Для большинства этих ограничений может быть зафиксирован спектр характерных понятий – *дескрипторов,* которые используются для наименования и раскрытия смыслового названия. Состав дескрипторов для ряда нефункциональных требований зафиксирован в соответствующих международных и ведомственных стандартах, которые позволяют избежать неоднозначности в их толковании.

Функциональные требования отражают семантические особенности проблемной области, а терминологические расхождения для них являются достаточно существенными. Имеется тенденция к созданию стандартизации понятийного базиса большинства проблемных областей, которые имеют опыт компьютеризации.

Следующий шаг анализа требований – установление их приоритетов и избежание конфликтов между ними.

Продукт процесса анализа – построенная модель проблемы, которая

ориентирована на понимание этой модели исполнителем до начала проектирования системы.

К настоящему времени сложилось направление в инженерии программирования – инженерия требования, сущность которой достаточно подробно рассмотрена в соответствующей области знаний ядра SWEBOK и приводится ниже.

4.1.3. Инженерия требований ПО

Инженерная дисциплина анализа и документирования требований к ПО, которая заключается в преобразовании предложенных заказчиком требований к системе в описание требований к ПО, их спецификация и верификация. Она базируется на *модели процесса* определения требований, процессах актеров – действующих лиц, управлении и формировании требований, а также на процессах верификации и повышения их качества.

Модель процесса – это схема процессов ЖЦ, которые выполняются от начала проекта и до тех пор, пока не будут определены и согласованы требования. При этом процессом может быть маркетинг и проверка осуществимости требований в данном проекте.

Управление требованиями к ПО заключается в планировании и контроле выполнения требований и проектных ресурсов в процессе разработки компонентов системы на этапах ЖЦ.

Качество и процесс улучшения требований – это процесс формулировки характеристик и атрибутов качества (надежность, реактивность и др.), которыми должна обладать система и ПО, методы их достижения на этапах ЖЦ и адекватности процессов работы с требованиями.

Управление требованиями к системе – это руководство процессами формирования требований на всех этапах ЖЦ, которое включает управление изменениями и атрибутами требований, отражающими программный продукт, а также проведение мониторинга – восстановления источника требований.

Неотъемлемой составляющей процесса управления является т*рассирование требований* для отслеживания правильности задания и реализации требований к системе и ПО на этапах ЖЦ и обратный процесс отслеживания от полученного продукта к требованиям.

При управлении требований выполняются процессы:

– управления версиями требований,
– управление рисками,
– разработка атрибутов требований,
– контроль статуса требований, измерение усилий в инженерии требований,
– другие.

Управление рисками состоит оценке, предотвращении и контроле появления риска определения отдельных требований. Проводиться планирование работ на проекте по управлению рисками в разработке требований.

4.1.4. Верификация и формализация требований

Сбор требований является начальным и неотъемлемым этапом процесса разработки ПС. Он заключается в определении набора функций, которые необходимо реализовать в продукте. Процесс сбора требований реализуется частично в общении с заказчиком, частично посредством мозговых штурмов разработчиков. Результатом является формирование набора требований к системе, именуемой в отечественной практике техническим заданием.

Фиксация требований (Requirement Capturing), с одной стороны, определяется желаниями заказчика в реализации того или иного свойства. С другой стороны в процессе сбора требований может обнаружиться ошибка, которая приведет к определенным последствиям, устранение которых заберет непредвиденные ресурсы – дополнительное кодирование, перепланирование.

Ошибка может быть тем серьезней, чем позже она будет обнаружена, особенно если это связано с множеством спецификаций. Поэтому одной из составляющей этапа фиксации требований, наряду со сбором является

верификация требований, а именно проверка их на непротиворечивость и полноту.

Автоматизированная верификация требований может производиться лишь после спецификации или формализации требований.

Спецификация требований к ПО – процесс формализованного описания функциональных и нефункциональных требований, требований к характеристикам качества в соответствии со стандартом качества ISO/IEC 12119-94, которые будут учитываться при создании программного продукта на этапах ЖЦ ПО. В спецификации требований отражается структура ПО, требования к функциям, показателям качеству, которых необходимо достигнуть, и к документации. В спецификации задается в общих чертах архитектура системы и ПО, алгоритмы, логика управления и структура данных. Специфицируются также системные требования, нефункциональные требования и требования к взаимодействию с другими компонентами (БД, СУБД, протоколы сети и др.).

Формализация включает в себя определение компонентов системы и их состояний; правил взаимодействия компонентов и определения условий в формальном виде, которые должны выполняться при изменении состояний компонентов. Для формального описания поведения системы используются языки инженерных спецификаций, например, UML. В качестве формальной модели для описания требований используются базовые протоколы, которые позволяют использовать дедуктивные средства верификации в сочетании с моделированием поведения систем путем трассирования.

Валидация (аттестация) требований – это проверка требований, изложенных в спецификации для того, чтобы убедиться, что они определяют данную систему и отслеживание источников требований. Заказчик и разработчик ПО проводят экспертизу сформированного варианта требований с тем, чтобы разработчик мог далее проводить разработку ПО. Одним из методов аттестации является прототипирование, т.е. быстрая отработка отдельных требований на конкретном инструменте и исследование масштабов изменения требований, измерение объема функциональности и стоимости, а также создание моделей

оценки зрелости требований.

Верификация требований – это процесс проверки правильности спецификаций требований на их соответствие, непротиворечивость, полноту и выполнимость, а также на соответствие стандартам. В результате проверки требований делается согласованный выходной документ, устанавливающий полноту и корректность требований к ПО, а также возможность продолжить проектирование ПО.

4.2. Объектно-ориентированная инженерия требований

Структурирование проблемы на отдельные компоненты и обеспечение способов их взаимодействия определяют понятность и «прозрачность» описания требований, полученных в результате процесса анализа.

Типы составляющих компонентов и правила их композиции определяются в программной инженерии как *архитектура системы*. Процесс декомпозиции проблемы, определяющей архитектуру системы, называют *парадигмой программирования,* базирующейся на двух широко распространенных моделях: функции-данные и объектная модель.

Модель функции-данные исторически появилась первой. Согласно этой модели проблема декомпозируется на последовательность функций и обрабатываемых с их помощью данных. Элементами композиции служат данные и функции над ними, их представления должны быть согласованы между собой. Если изменяются некоторые данные, то пересматриваются функции, которые их обрабатывают, и определяются пути их изменений. Т.е. внесение локальных изменений в постановку проблемы требует ревизии всех данных и функций для подтверждения того, что на них не повлияли внесенные изменения.

Объектно-ориентированный подход к разработке программных систем такого недостатка не имеет. В нем общее видение решения проблемы формирования требований осуществляется исходя из следующих постулатов:

– мир составляют объекты, которые взаимодействуют между собой;

– каждому объекту присущ определенный состав свойств или атрибутов, который определяется своим именем и значениями;

– объекты могут вступать в отношения друг с другом;

– значения атрибутов и отношения могут с течением времени изменяться;

– совокупность значений атрибутов конкретного объекта в определенный момент времени определяет его состояние;

– совокупность состояний объектов определяет состояние мира объектов;

– мир и его объекты могут находиться в разных состояниях и порождать некоторые события;

– события могут быть причиною других событий или изменений состояний.

Каждый объект может принимать участие в определенных процессах, разновидностями которых есть:

переходы из одного состояния в другое под влиянием соответствующих событий;

возбуждение определенных событий или посылка сообщений другим объектам;

– операции, которые могут выполнять объекты;

– возможные совокупности действий, которые задают его поведение;

обмен сообщениями.

Объект это определенная абстракция данных и поведения. Множество экземпляров с общим набором атрибутов и поведением составляет класс объектов. Определение класса связано с известным принципом *сокрытия информации*, суть которого можно сформулировать так: сообщайте пользователю только то, что ему нужно. Этот принцип имеет ряд преимуществ:

– пользователь избавлен от необходимости знать лишнее;

– то, что ему не сообщили, он не испортит (защита от намеренных или случайных неправомерных действий;

– все, о чем не знает пользователь, можно изменять.

Таким образом, определение объектов в соответствии с данным принципом состоит из двух частей – видимой и невидимой. Видимая часть содержит все сведения, которые требуется для того, чтобы взаимодействовать с объектом и называется *интерфейса объекта.* Невидимая часть содержит подробности внутреннего устройства объекта, которые «инкапсулированы» (т.е. находятся словно бы в капсуле). Так, например, если объектом является некоторый прибор, который регистрирует показатели температуры, то к видимой его части относится операция показа значения температуры.

Другим важным свойством определения объектов является *наследование*. Один класс объектов наследует другой, если он полностью вмещает все атрибуты и поведение наследуемого класса, но имеет еще и свои атрибуты и (или) поведение. Класс, который наследуют свойства другого, называют *суперклассом*, а класс, которого наследует, называют *подклассом.* Наследственность фиксирует общие и отличающиеся черты объектов и позволяет явно выделять компоненты проблемы, которые можно использовать в ряде случаев при построении нескольких классов–наследников.

Классы могут образовывать иерархии наследников произвольной глубины, где каждый отвечает определенному уровню абстракции, являясь обобщением класса-наследника и конкретизацией класса, который наследует его самого. Например, класс «число» в качестве наследников имеет подклассы: «целые числа», «комплексные числа» и «действительные числа». Все эти подклассы наследуют операции суперкласса (сложения и вычитания), но каждый из них имеет свои особенности выполнения этих операций.

При объектно–ориентированном подходе модели определяются через взаимодействие определенных объектов. В модели требований фигурируют объекты, взаимодействие которых определяет проблему, решаемую с помощью программной системы, а в других моделях (модели проекта, моделях реализации и тестирования) заданный принцип взаимодействия объектов определяет сущность решения этой проблемы (модели проекта и реализации) или проверки

достоверности решения (модель тестирования).

В настоящее время предложен ряд современных методов объектно-ориентированного анализа требований, объектно-ориентированного проектирования программ, объектно-ориентированного программирования (C++, JAVA). Наибольшую ценность среди них имеет проблема согласованности между ними.

Если удается установить соответствие между объектами указанных моделей на разных стадиях (процессах) жизненного цикла продукта, то они позволяют провести *трассирование требований*, т.е. проследить за последовательной трансформацией требований объектов на этих стадиях. Трассирование заключается в контроле трансформаций объектов при переходе от этапа к этапу с учетом внесения изменений во все наработанные промежуточные продукты разных стадий разработки и ее завершения.

Концептуальное моделирование проблемы системы происходит в терминах взаимодействия объектов:

– онтология домена определяет состав объектов домена, их атрибуты и взаимоотношения, а также оказываемые услуги;

– модель поведения определяет возможные состояния объектов, инциденты, события, инициирующие переходы из одного состояния в другое, а также сообщения, которые объекты посылают друг другу;

– модель процессов определяет действия, которые выполняют объекты.

Все объектные методы имеют в своем составе приведенные модели, отличающиеся своими нотациями и некоторыми другими деталями.

4.2.1. Метод инженерии требований А. Джекобсона

Данный метод является методом с последовательным выявлением объектов, существенных для домена проблемной области [3]. Все методы анализа предметной области в качестве первого шага выполняют выявление объектов. Правильный выбор объектов обуславливает понятность и точность требований.

Рассматриваемый метод Джекобсона дает рекомендации относительно того, с чего начинать путь к пониманию проблемы и поиску существенных для нее объектов.

Автор назвал свой метод как базирующийся на вариантах (примерах или сценариях –use case driven) системы, которую требуется построить. В дальнейшем термин сценарий используется для обозначения варианта представления системы.

Сложность проблемы обычно преодолевается путем декомпозиции ее на отдельные компоненты с меньшей сложностью. Большая система может быть сложной, чтобы представить ее компоненты программными модулями. Поэтому разработка системы с помощью данного метода начинается с осмысления того, для кого и для чего создается система.

Сложность общей цели выражается через отдельные подцели, которым отвечают функциональные или нефункциональные требования и проектные решения. Цели являются источниками требований к системе и способом оценки этих требований, путем выявления противоречий между требованиями и установления зависимостей между целями.

Цели можно рассматривать, как точку зрения отдельного пользователя или заказчика или среды функционирования системы. Цели могут находиться между собою в определенных отношениях, например, конфликтовать, кооперироваться, зависеть одна от другой или быть независимыми.

Следующим шагом после выявления целей является определение носителей интересов, которым отвечает каждая цель, и возможные варианты удовлетворения составных целей в виде сценариев работы системы. Последние помогают пользователю получить представление о назначении и функциях системы. Эти действия соответствуют первой итерации определения требований к разработке.

Таким образом, производится последовательная декомпозиция сложности проблемы в виде совокупности целей, каждая из которых трансформируется в совокупность возможных вариантов использования системы, которые трансформируются в процессе их анализа в совокупность взаимодействующих объектов.

Определенная цепочка трансформации (проблема – цели – сценарии – объекты) отражает степень концептуализации в понимании проблемы, последовательного снижения сложности ее частей и повышения уровня формализации моделей ее представления.

Трансформация обычно выражается в терминах базовых понятий проблемной области, активно используется для представления актеров и сценариев, или создается в процессе построения такого представления.

Каждый сценарий инициируется определенным пользователем, являющимся носителем интересов. Абстракция определенной роли личности пользователя–инициатора запуска определенной работы в системе, представленной сценарием, и обмена информацией с системой – называется *актером*. Это абстрактное понятие обобщает понятие действующего лица системы. Фиксация актеров можно рассматривать, как определенный шаг выявления целей системы через роли, которые являются носителями целей и постановщиками задач, для решения которых создается система.

Актер считается внешним фактором системы, действия которого носят недетерминированный характер. Этим подчеркивается разница между пользователем как конкретным лицом и актером–ролью, которую может играть любое лицо в системе.

В роли актера может выступать и программная система, если она инициирует выполнение определенных работ данной системы. Таким образом, актер – это абстракция внешнего объекта, экземпляр которого может быть человеком или внешней системой. В модели актер представляется классом, а пользователь – экземпляром класса. Одно лицо может быть экземпляром нескольких актеров.

Лицо в роли (экземпляр актера) запускает операции в системе, соотнесенные с поведением последовательности транзакций системы и называется *сценарием*. Когда пользователь, как экземпляр актера, вводит определенный символ для старта экземпляра соответствующего сценария, то это приводит к выполнению ряда действий с помощью транзакций, которые заканчиваются тогда, когда экземпляр

сценария находится в состоянии ожидания входного символа.

Экземпляр сценария существует, пока он выполняется и его можно считать экземпляром класса, в роли которого выступает описание транзакции.

Сценарий определяет протекание событий в системе и обладает состоянием и поведением. Каждое взаимодействие между актером и системой это новый сценарий или объект. Если несколько сценариев системы имеют одинаковое поведение, то их можно рассматривать как класс сценариев. Вызов сценария – это порождение экземпляра класса. Таким образом, сценарии – это транзакции с внутренним состоянием.

Модель системы по данному методу основывается на сценариях и внесение в нее изменений должно осуществляться повторным моделированием актеров и запускаемых ими сценариев. Такая модель отражает требования пользователей и легко изменяется по их предложению. Сценарий – это полная цепочка событий, инициированных актером, и спецификация реакции на них. Совокупность сценариев определяет все возможные варианты использования системы. Каждого актера обслуживает своя совокупность сценариев.

Можно выделить базовую цель событий, существенную для сценария, а также альтернативные, при ошибках пользователя и др.

Для задания модели сценариев используется специальная графическая нотация, со следующими основными правилами:

– актер обозначается иконой человека, под которой указывается название;

– сценарий изображается овалом, в середине которого указывается название иконы;

– актер связывается стрелкой с каждым запускаемым им сценарием.

На рис. 4.1 представлен пример диаграммы сценариев для клиента банка, как актера, который запускает заданный сценарий. Для достижения цели актер обращается не к кассиру или клерку банка, а непосредственно к компьютеру системы.

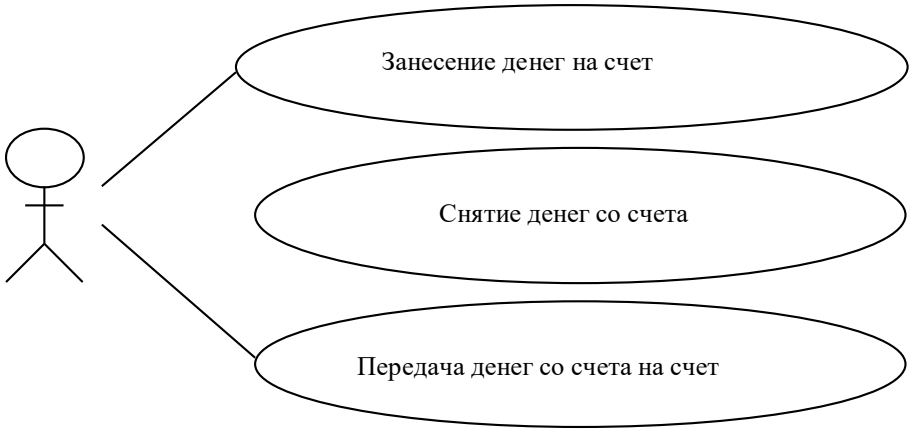


Рис.4.1. Пример диаграммы сценариев для клиента банка

Все сценарии, которые включены в систему, обведены рамкой, определяющей границы системы, а актер находится вне рамки, являясь внешним фактором по отношению к системе.

Отношения между сценариями

Между сценариями можно задавать отношения, которые изображаются на диаграмме сценариев пунктирными стрелками с указанием названия соответствующих отношений.

Для сценариев определены два типа отношений:

Отношение «расширяет» означает, что функции одного сценария является дополнением к функциям другого, и используется когда имеется несколько вариантов одного и того же сценария. Инвариантная его часть изображается как основной сценарий, а отдельные варианты как расширения. При этом основной сценарий является устойчивым, не меняется при расширении вариантов функций и не зависит от них.

Типовой пример отношения расширения: моделирование необязательных фрагментов сценариев приведен на рис.4.2. При выполнении сценария расширение прерывает выполнение основного расширяемого сценария, причем последний не знает, будет ли расширение и какое. После завершения выполнения сценария расширение будет продолжено путем выполнения основного сценария.

Отношение «использует» означает, что некоторый сценарий может быть

использован как расширение несколькими другими сценариями.

Оба отношения – инструменты определения наследования, если сценарии считать объектами. Отличие между ними состоит в том, что при расширении функция рассматривается как дополнение к основной функции и может быть понятной только в паре с ней, тогда как в отношении «использует» дополнительная функция имеет самостоятельное определение и может быть использована всюду.

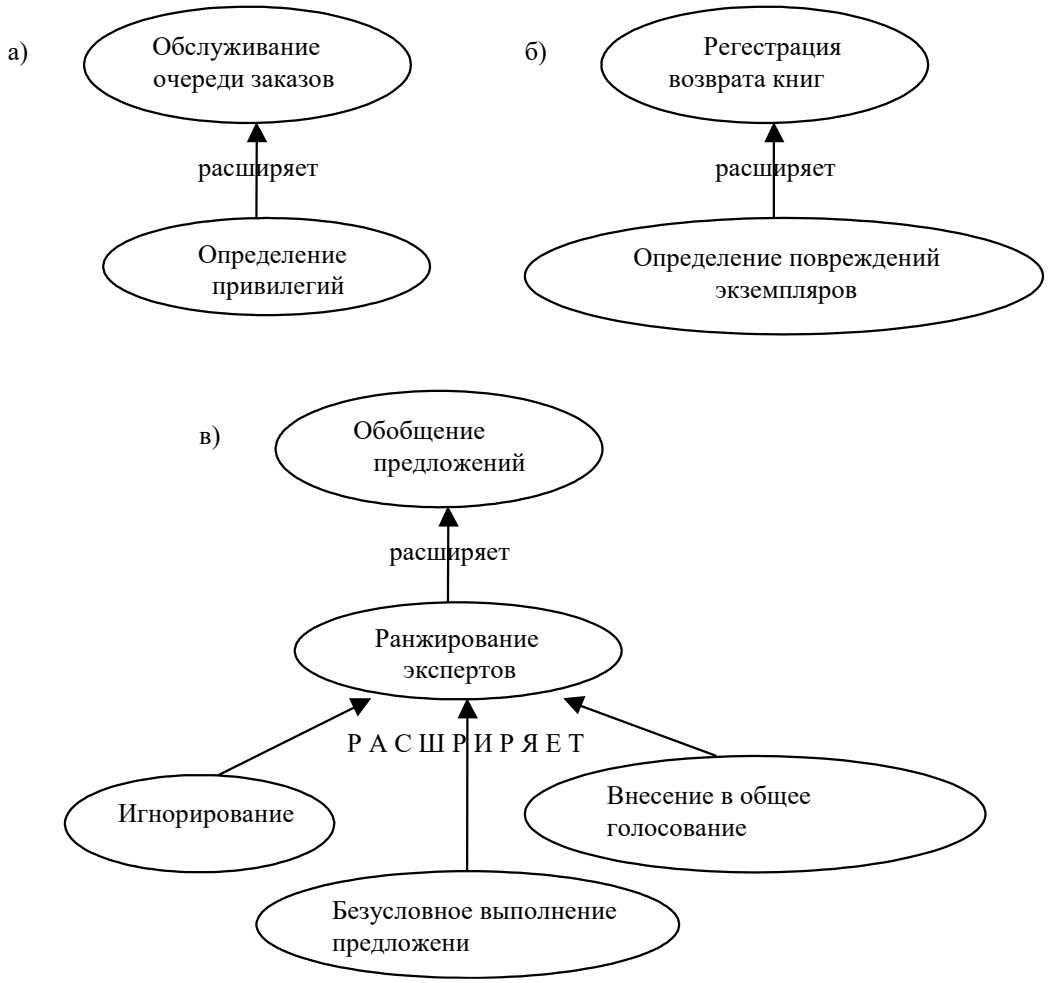


Рис.4.2. Примеры расширения сценариев

На рис. 4.3. показано, что сценарий «сортировать» связан отношением «использует» с несколькими сценариями.

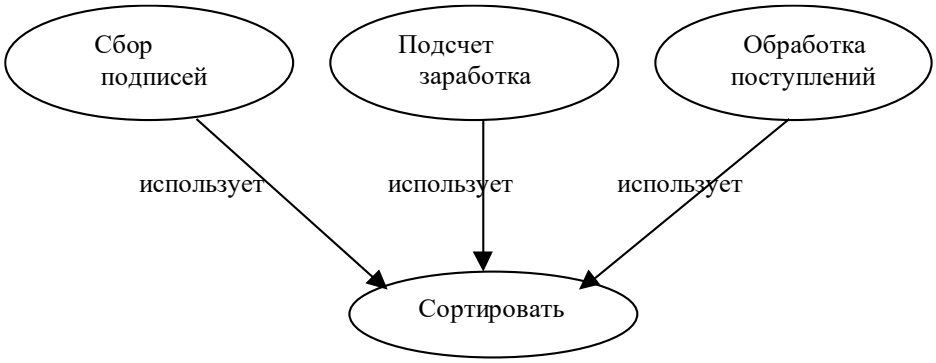


Рис.4.3. Пример отношения «использует».

Таким образом, продуктом первой стадии инженерии требований (сбора требований) является модель требований, которая состоит с трех частей:

1) онтология домена;

2) модель сценариев, называемая диаграммой сценариев;

3) описание интерфейсов сценариев.

Модель сценариев сопровождается неформальным описанием каждого из сценариев. Нотация такого описания не регламентируется. Как один из вариантов, описание сценария может быть представлено последовательностью элементов:

– название, которое помечает сценарий на диаграммах модели требований и служит средством ссылки на сценарий;

– аннотация (краткое содержание в неформальном представлении);

– актеры, которые могут запускать сценарий;

– определение всех аспектов взаимодействия системы с актерами (возможные действия актера и их возможные последствия), а также запрещенных действий актера;

– предусловия, которые определяют начальное состояние на момент запуска сценария, необходимое для успешного выполнения;

– функции, которые реализуются при выполнении сценария и определяют порядок, содержание и альтернативу действий, выполняемых в сценарии;

– исключительные или нестандартные ситуации, которые могут появиться

при выполнении сценария и потребовать специальных действий для их устранения (например, ошибка в действиях актера, которую способна распознать система);

– реакции на предвиденные нестандартные ситуации;

– условия завершения сценария;

– постусловия, которые определяют конечное состояние сценария при его завершении.

На дальнейших стадиях сценарий трансформируется в сценарий поведения системы, т.е. к приведенным элементам модели добавляются элементы, связанные с конструированием решения проблемы и нефункциональными требованиями:

– механизмы запуска сценария (позиции меню);

– механизмы ввода данных;

– поведение при возникновении чрезвычайных ситуаций.

Следующим шагом процесса проектирования является преобразование сценария в описание компонентов системы и проверка их правильности.

Описание интерфейсов делается неформально, они определяют взгляд пользователя на систему, с которым необходимо согласовать требования, собранные на этапе анализа системы и определения требований. Все вопросы взаимодействия отмечаются на панели экрана для каждого из актеров и их элементы (меню, окна ввода, кнопки – индикаторы и т.п.).

Построение прототипа системы, моделирующего действия актера, помогает отработать детали и устранить недоразумения между заказчиком и разработчиком.

4.2.2. Модель анализа требований. Определение объектов

Модель требований дает обобщенное представление о будущих услугах системы для ее клиентов (актерам). Полученное представление является предметом анализа с целью дальнейшего структурирования проблемы. Основу составляет объектная архитектура, результатом структурирования которой должна быть совокупность объектов, полученная путем последовательной декомпозиции каждого из сценариев на объекты с действиями сценария, а также их

взаимодействие, определяемое функциональностью системы.

Таким образом, стратегия выбора объектов в системе базируется на следующих принципах:

– изменение требований неизбежно;

– объект модифицируется вследствие изменения соответствующих требований к системе;

объект должен быть устойчивым к модификации системы (локальные изменения отдельного требования, должны повлечь за собой изменения как можно меньшего количества объектов).

Исходя из этих принципов, в данном методе различаются типы объектов в зависимости от их способности к изменениям, система структурируется согласно следующих критериев:

– наличие информации для обработки системы (для обеспечения ее внутреннего состояния);

– определение поведения системы;

– презентация системы (ее интерфейсов с пользователями и другими системами).

Выбор критериев обусловлен экспертными исследованиями динамики изменений действующих систем.

Для каждого критерия функциональности системы имеется совокупность объектов, с помощью которых локализуются требования к наиболее стабильным фрагментам.

Рассматривается три типа объектов:

– объекты-сущности;

– объекты интерфейса;

– управляющие объекты.

Объекты-сущности моделируют в системе долгоживущую информацию, хранящуюся после выполнения сценария. Обычно, им соответствуют реальные сущности, которые находят свое отображение в БД. Большинство из них может

быть выявлено из анализа онтологии проблемной области, но во внимание берутся только те из них, на которые ссылаются в сценариях.

Объекты интерфейса включают в себя функциональности, зависимые непосредственно от окружения системы и определенных в сценарии. С их помощью актеры взаимодействуют со сценариями системы, их задачей является трансляция информации, которую вводит актер, в события, на которые реагирует система, и обратная трансляция событий, вырабатывающая системой, в вывод для актера. Такие объекты определяются путем анализа описаний интерфейсов сценариев модели требований и анализа действий актеров по запуску каждого из соответствующих ему сценариев.

Управляющие объекты – это объекты, которые превращают информацию, введенную объектами интерфейса и представленную объектами-сущностями, в информацию, что выводится интерфейсными объектами. Они являются своеобразным «клеем» для соединения объектов, связывая цепочки событий и задавая взаимодействие объектов.

Такой распределение служит целям локализации изменений в системе. При преобразовании модели требований в модель анализа каждый сценарий разбивается на эти три типа объектов. При этом один и тот же объект может присутствовать в нескольких сценариях, и важно распознать такие объекты, чтобы унифицировать их функции и определить их как единый объект. Критерий распознавания состоит в том, что если в различных сценариях имеется ссылка на один и тот же экземпляр объекта, то это один и том же объект.

После выделения объектов, формируется базис архитектуры системы как совокупности взаимодействующих объектов, для каждого из которых можно установить связь с соответствующим сценарием модели требований. После чего провести трассирование требований, идя от модели требований к модели анализа.

Атрибуты объектов представлены прямоугольниками, объединенными прямой линией с символом объекта, при этом на линии указывается название атрибута, а в прямоугольнике – его тип.

Между объектами определяются ассоциации, которые изображаются одно- или двунаправленными стрелками, на которых указываются названия ассоциаций типа:

– взаимодействует,

– составляется с,

– выполняет роль,

– наследует,

– расширяет,

– использует.

Эти ассоциации существенно отличаются от ассоциаций в моделях данных. Последние нацелены преимущественно на осуществление навигации в БД, тогда как ассоциации определяют взаимодействие объектов.

Исходя из известного метода анализа требований И. Джекобсона на стадии анализа определяются:

– онтология домена;

– модель сценариев;

– неформальное описание сценариев и актеров;

– описание интерфейсов сценариев и актеров;

– диаграммы взаимодействия объектов сценариев.

Полученные требования трассируются, после чего проводится реализация функций системы на следующих этапах ЖЦ.

4.3. Классификация требований

Формируемые требования к системе разделены на две категории:

– функциональные требования, которые отображают возможности проектируемой системы;

– нефункциональные требования, которые отображают ограничения, определяющие принципы функционирования системы и доступа к данным системы пользователей.

Первая из приведенных категорий дает ответ на вопрос «что делает система»,

а вторая определяет характеристики ее работы, например, что вероятность сбоев системы на некотором промежутке времени, доступ до ресурсов системы разных категорий пользователей и др.

Функциональные требования характеризуют функции системы или ее ПО, а также способы поведения ПО в процессе выполнения функций и методы передачи и преобразования входных данных в результаты. *Нефункциональные требования* определяют условия и среду выполнения функций (например, защита и доступ к БД, секретность и др.). Разработка требований и их локализация завершается на этапе проектирования архитектуры и отражается в специальном документе, по которому проводится окончательное согласование требований для достижения взаимопонимания между заказчиком и разработчиком.

Функциональные требования связаны с семантическими особенностями ПрО, для которой планируется разработка ПС. Важным фактором является проблема использования соответствующей терминологии при описание моделей ПрО и требований к системе. Одним из путей ее решения является стандартизации терминологии для нескольких ПрО (например, для информационных технологий, систем обеспечения качества и др.). Тенденция к созданию стандартизированного понятийного базиса для большинства ПрО отражает важность этой проблемы в плане обеспечения единого понимания терминов, используемых в документах, описывающих требования к системе и к ПО.

Нефункциональные требования могут иметь числовой вид (например, время ожидания ответа, количество обслуживаемых клиентов, БД данных и др.), а также содержать числовые значения показателей надежности и качества работы компонентов системы, период смены версий системы и др. Для большинства ПС, с которыми будут работать много пользователей, требования должны выражать такие ограничения на работу системы:

– конфиденциальность;

– отказоустойчивость;

– одновременность доступа к системе пользователей;

– безопасность;

– время ожидания ответа на обращение к системе;

– ограничения на исполнительские функции системы (ресурсы памяти, скорость реакции на обращение к системе и т.п.);

– регламентации действующих стандартов, упрощающих процессов организации формирования требований и менеджмента.

Иными словами, для каждой системы формулируются нефункциональные требования, относящиеся к защите от несанкционированного доступа, к регистрации событий системы, аудиту, резервному копированию, восстановлению информации. Эти требования на этапе анализа и проектирования структуры системы должны быть определены и формализованы аналитиками. Для обеспечения безопасности системы должны быть определены категории пользователей системы, которые имеют доступ к тем или иным данным и компонентам.

Определяются объекты и субъекты защиты. На этапе анализа системы решается вопрос, какой уровень защиты данных необходимо предоставить для каждого из компонентов системы, выделить критичные данные, доступ к которым строго ограничен. Для этого применяется система мер по регистрации пользователей, т.е. идентификации и аутентификации, а также реализуется защита данных, ориентированная на регламентированный доступ к объектам данных (например, к таблицам, представлениям). Если же требуется сделать ограничение доступа к данным (например, к отдельным записям, полям в таблице), то в системе должны предусматриваться определенные средства (например, мандатная защита).

Для обеспечения восстановления и хранения резервных копий БД, архивов баз данных изучаются возможности, предоставляемые СУБД, и рассматриваются способы обеспечения требуемого уровня бесперебойной работы системы, а также организационные мероприятия, отображаемые в соответствующих документах по описанию требований к проектируемой системе.

В целях описания нефункциональных (защита, безопасность,

аутентификация и др.) требований к системе и фиксации сведений о способности компонента обеспечивать несанкционированный доступ к нему неавторизованных пользователей, языки спецификаций компонентов дополнились средствами описания нефункциональных свойств. Эта группа свойств целиком связана с сервисом среды функционирования компонентов, ее способностью обеспечивать меры борьбы с разными угрозами, которые поступают извне от пользователей, не имеющих прав доступа к некоторым данным или ко всем данным системы.

Процесс формализованного описания функциональных и нефункциональных требований, а также требований к характеристикам качества с учетом стандарта качества ISO/IEC 9126–94 будет уточняться на этапах ЖЦ ПО и специфицироваться. В спецификации требований отражается структура ПО, требования к функциям, качеству и документации, а также задается архитектура системы и ПО, алгоритмы, логика управления и структура данных. Специфицируются также системные, нефункциональные требования и требования к взаимодействию с другими компонентами и платформами (БД, СУБД, сеть и др.).

4.4. Трассирование требований

Одной из главных проблем сбора требований является проблема изменения требований. Требования появляются в процессе общения между группой заказчиков и аналитиками системы в виде интервью, обсуждений, которые не приносят желаемого результата. Это объясняется тем, что составляющих элементов требований последовательно изменяются, благодаря чему их содержание и форма постепенно становятся более точными и полными, т.е. соответствуют действительности [6].

Инструменты трассировки поддерживают развитие и обработку требований с сохранением их описания и внутренних связей между ними. Трассировка помогает проверять особенности системы на спецификациях требований, выявлять источники разнообразных ошибок и управлять изменениями требований. Если требования разрабатывались в объектной ориентации, то объектами трассировки являются классы и суперклассы и поддерживаемые отношения между ними.

Некоторые методы трассировки базируются на формальных спецификациях отношений (фреймы, соглашения сотрудничества и др.), другие ограничиваются описаниями действий, ситуаций, контекста и возможных решений.

Трассировку можно описать исходя из следующего:

1) требования изменяются во время функционирования системы;
2) возникновение требований и их расположение зависит от деталей практической ситуации и контекста их возникновения (требования можно изменить, изменяя эти детали);
3) трассировка требований должна поддерживаться и изменятся на протяжении всего ЖЦ программного продукта (т.к. изменяются сами требования, необходимо проводить изменение и промежуточных результатов, полученных при анализе, спецификации, кодировке и т.д.);
4) для удобства трассировки использовать иерархическую структуру связей между требованиями, основу которой составляет информация об атрибутах требований.

Чтобы принять решение о возможных модификациях, необходимо иметь достаточно информации о частях и связях между ними. Более того, различные аспекты требований могут быть по-разному представлены и изменены их контексты путем персонального вмешательства аналитиков или заказчика.

Механизмы трассировки должны учитывать следующее:

1) вместо простых связей вводить более сложные отношения (например, транзитивное отношение для выделения цепочек связей) или вводить специфические отношения;
2) использовать сложные и гибкие пути трассировки;
3) поддержать базы данных объектов трассировки и отношений между ними.
4) Желательно наличие механизмов поддержки возможностей объектно-ориентированного программирования, операций над классами, а также

механизмов унификации функций и отношений (1:1, 1:N, N:M), уничтожение и изменение связей, установка стандартных связей.

Трассировка может быть выборочной для определенных объектов, беспорядочной проверкой объектов, связанных с другими объектами, а также с возможными переходами от одного объекта к другим. Она проводится с использованием механизмов поддержания контекста и отношений, соответствующих данной ситуации (например, трассировка с регулярными выражениями, когда контекст может быть изменен только при модификации соответствующего регулярного выражения).

Человеческий фактор. Любой процесс постановки требований напрямую связан с умением людей контактировать друг с другом, кооперироваться или эффективно распределять разноплановые производственные функции между собой. Необходимо уметь достигнуть соглашения в спорных вопросах, касающихся дизайна или технических решений. Организационные функции не должны выходить за рамки понимания проблемы. Важно чтобы лица, работающие над проектом на разных уровнях, имели возможность эффективно общаться друг с другом.

Технологии проекта должны обращать внимание на динамику людских отношений в коллективе. Они должны способствовать эффективному достижению согласия и управления разногласиями, давать возможность снижать сложность отношений в группе сотрудников, работающих над проектом, особенно в повседневной работе при создании высококачественного продукта.

Наиболее привычной и результативной моделью отношений между заказчиком и проектировщиком является модель типа «учитель – ученик». На практике заказчик наблюдает за работой проектировщика системы. Когда заказчик его не беспокоит, проектировщик обращает внимание на примеры и сам отвечает себе на возникающие вопросы. В порядке исключения, заказчик может остановить свою работу, обдумать поставленный вопрос для пояснения и удовлетворения проектировщика.

Это даже помогает разработчику разобраться в деталях своей работы и избавляет его от описания излишних абстракций. Проектировщик должен сформировать свои независимые идеи относительно проектировки будущей системы и постоянно консультироваться с заказчиком, что избежать ошибок на самых ранних этапах проектирования системы Использование данной модели на практике возможно только при хороших взаимоотношениях между заказчиком и исполнителем проекта.

4.5. Управление требованиями на базе стандартов

Процесс управления требованиями традиционно считается одним из ключевых при создании автоматизированных систем – наибольшие риски проектов связаны с высокой изменчивостью требований и ошибками в их определении. Организация управления требованиями направлена на снижение таких рисков, причем с помощью методики, основанной на международных и отечественных стандартах.

Проблемы при выполнении проектов создания автоматизированных систем (АС) могут возникать из-за неформального сбора информации, предполагаемой функциональности АС, ошибочных или несогласованных нефункциональных требований к системе, а также нерегламентированной процедуры их изменения. Организация управления требованиями, прежде всего, направлена на дезавуирование таких проблем за счет усовершенствования способов сбора, документирования, согласования и модификации требований к системе, отслеживания требований от заинтересованных лиц и из прочих источников, их порождающих. Регламентация процедур управления требованиями должна обеспечить высокое качество работы с ними в проектах, связанных с разработкой, сопровождением, созданием АС, разработкой их организационно методического обеспечения, а также внедрения готовых систем за счет уменьшения типичных ошибок при работе с требованиями. Существенный момент предлагаемой методики – использование международных и отечественных стандартов в области

управления жизненным циклом (ЖЦ) АС, позволяющее практически без адаптации применять методики в рамках уже существующих, хорошо зарекомендовавших себя на практике процессах разработки. В качестве инструментального средства, поддерживающего моделирование требований на основе UML 2.0., используется продукт Enterprise Architect компании Sparx System.

4.6. Подготовка к управлению требованиями

Под управлением требованиями обычно понимается систематический подход к выявлению, документированию, планированию реализации требований и отслеживанию их изменений [2, 7-9]. Методика призвана регламентировать следующие процедуры работы с требованиями: выявление, документирование, верификация и утверждение требований, планирование реализации и отслеживание изменений требований.

Методика предусматривает две стадии: подготовку управления требованиями и непосредственно управление ими. Процесс подготовки управления требованиями представлен на рис. 2, а его описание в таблице 1.

Таблица 1. Описание процесса подготовки управления требованиями

№	Входная/Выходная информация	Шаг процесса	Участник	Бизнес правила
01	Информация от заинтересованных лиц (заказчики, менеджер проекта)/ Этапы жизненного цикла системы	Определение этапов ЖЦ системы, на котором будет организована работа с требованиями	Специалист по управлению требованиями	ГОСТ 34.601-90, ГОСТ 19.102-77, ГОСТ Р ИСО/МЭК 12207-99, RUP [3]
02	Этапы ЖЦ/Шаблоны	Подготовка шаблонов	Специалист по	ГОСТ 34.602-89,

№	Входная/Выходная информация	Шаг процесса	Участник	Бизнес правила
	документов	документов для каждого этапа	управлению требованиями	ГОСТ 19.201-78, РД 50-34.698-90, RUP [3]
03	Шаблоны документов/Коды типов требований	Кодирование типов требований в документах	Специалист по управлению требованиями	ГОСТ 34.602-89, ГОСТ 19.201-78, РД 50-34.698-90, RUP [3]
04	Коды типов требований/Атрибуты типов требований	Определение атрибутов типов требований	Специалист по управлению требованиями	Типовые атрибуты согласно RUP [3]
05	Коды типов требований/Зависимости между типами требований	Задание зависимостей между типами требований	Специалист по управлению требованиями	Выполняется для каждого конкретного проекта. Шаг процесса может быть пропущен
06	Зависимости между типами требований, типы требований с атрибутами, шаблоны документов/ План управления требованиями	Разработка плана управления требованиями	Специалист по управлению требованиями	ГОСТ 34.601-90, ГОСТ 19.102-77, ГОСТ 2.503-90, RUP [3]
07	Информация от заинтересованных	Разработка процедур	Специалист по	ГОСТ 34.602-89,

№	Входная/Выходная информация	Шаг процесса	Участник	Бизнес правила
	лиц по процедурам работы с требованиями/ Процедура выявления требований, анализа, верификации и утверждения	выявления, верификации, утверждения, изменения требований	управлению требованиями	ГОСТ 19.201-78, РД 50-34.698-90, ГОСТ 2.503-90, RUP [3]
08	ГОСТ 34.602-89, ГОСТ 19. 201 - 78/Шаблон модели и его описание	Создание шаблона модели требований и его описание	Специалист по управлению требованиями	ГОСТ 34.602-89, ГОСТ 19. 201-78

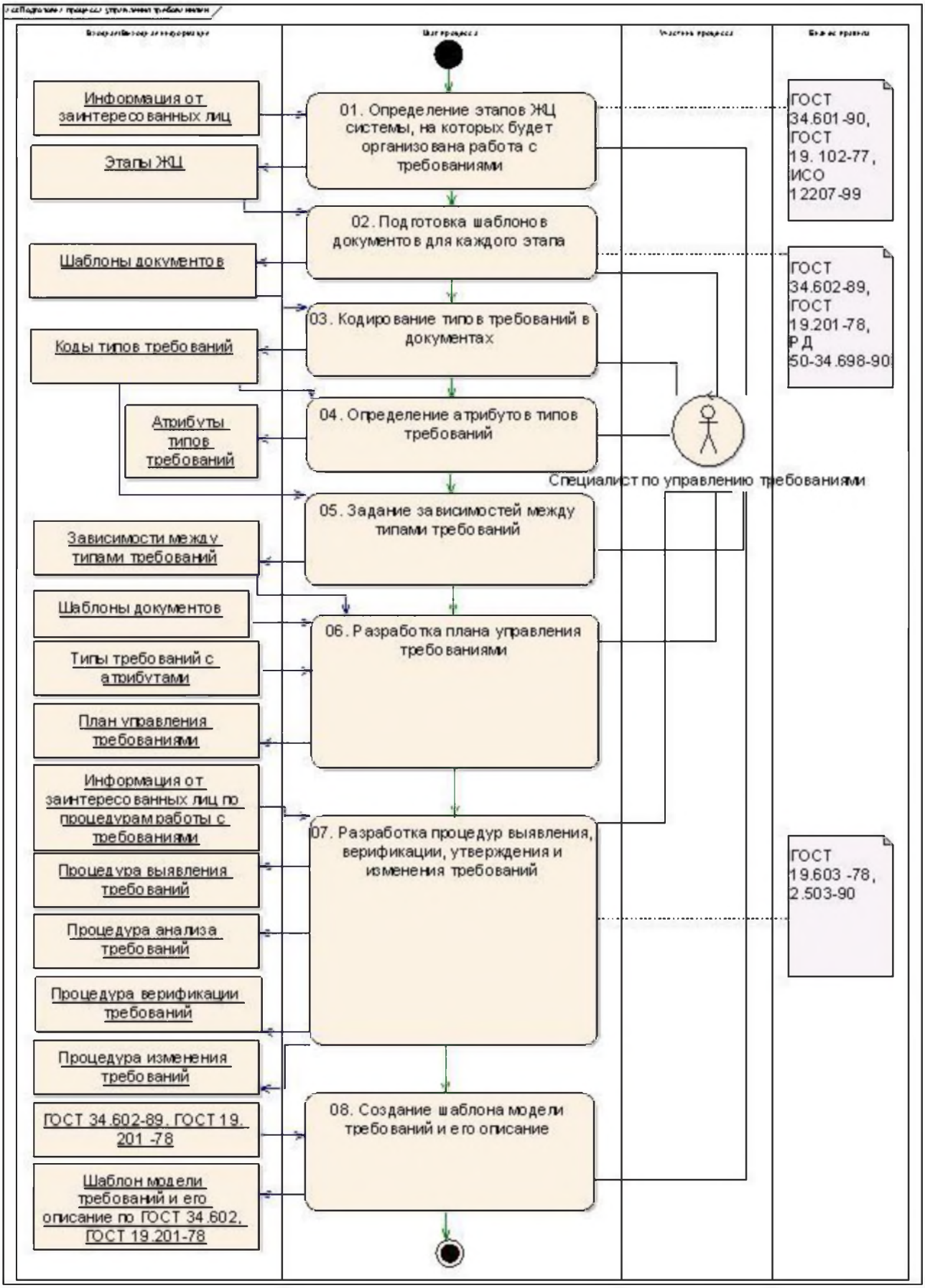


Рис. 4.4. Процесс подготовки управления требованиями

Определение этапов ЖЦ системы, на которых будет организована работа с требованиями, зависит от выбора его модели. Далее в качестве примера мы будем ориентироваться на определение согласно ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания», в котором работа с требованиями в рамках процедур, регламентируемых методикой, осуществляется на следующих стадиях: формирование требований к АС, разработка концепции, техническое задание и сопровождение АС.

Подготовка шаблонов документов выполняется для каждого этапа ЖЦ. Состав документов, оформляемых для выделенных этапов ЖЦ системы, приведен в таблице 2. Часть документов, создаваемых и сопровождаемых при управлении требованиями, определена в соответствии с ГОСТ, а часть готовится по специальным шаблонам, которые созданы на основе обобщения работ [7-9, 10]. В документах, формируемых с использованием подготовленных шаблонов, будут фиксироваться различные типы требований.

Далее выполняется кодирование типов требований в документах. Назначенные коды типов требований используются при определении их атрибутов и при задании зависимостей между ними.

Для каждого типа требований должен быть разработан атрибутный состав, позволяющий данное требование описать , а также дать возможность проектной группе оценить его с точки зрения ключевых аспектов разработки таких, как объем работ, стабильность требования, влияние на архитектуру, приоритетность [9]. Для большинства требований можно выделить общий блок атрибутов. При необходимости дополнительные атрибуты могут быть назначены индивидуально для каждого типа требования.

Различные типы требований могут оказывать взаимное влияние друг на друга, что должно быть учтено при реализации требований зависимых типов. Зависимости между типами требований могут носить различный характер. Например, требования по производительности могут конфликтовать с требованиями к пользовательскому интерфейсу в случае применения web-

интерфейса и его перегруженности графическими объектами. Установка взаимного влияния типов требований является задачей каждого конкретного проекта. При невозможности определить зависимости на начальных этапах разработки данный шаг процесса может быть пропущен.

Итогом работ, выполненных на предыдущих стадиях процесса подготовки управления требованиями, является разработка плана.

Следующим шагом процесса является разработка основных процедур управления требованиями: процедура выявления требований, процедура верификации требований, процедура изменения требований. При этом учитывается информация, поступившая от заинтересованных лиц по данным процедурам, и требования ГОСТ 34.602-89, ГОСТ 19. 201 -78.

В заключении создается шаблон модели требований и его описание. В качестве опорных документов используются ГОСТ 34.602-89, ГОСТ 19. 201 -78.

Подготовительный этап является крайне важным, позволяя регламентировать организационные процедуры, определить шаблоны основных документов, планировать процесс управления требованиями, а также определить роли основных участников процесса. Типовой состав участников проектной группы в рамках процесса управления требованиями выглядит следующим образом: системный аналитик; специалист по требованиям; рецензент требований; архитектор. Кроме того, в процессе задействован эксперт предметной области. В большинстве проектов роль эксперта, как правило, играет представитель заказчика, но в ряде случаев эксперт предметной области входит в состав проектной группы со стороны разработчика.

Часто для серии типовых проектов могут применяться сходные процедуры, шаблоны, роли, однако игнорирование подготовительного этапа в «нетиповом» проекте может привести к использованию неадекватных процедур управления требованиями, что часто сказывается на качестве конечного продукта, либо порождает другие проблемы в проекте, например, отсутствие или неактуальность проектной документации. Рассмотрим пример. Для типовых проектов,

ограниченных по срокам реализации (обычно до 6 мес.), наличие подробной документации не обязательно. В случае краткосрочного проекта, не относящегося к типовому (например, в виду его значимости и перспектив развития), недостаточная документированность изменений требований недопустима, что должно найти свое отражение в соответствующих процедурах процесса управления требованиями.

Если ориентироваться на ГОСТ 34.601-90, то предлагается использовать методику управления требованиями на следующих этапах ЖЦ (таблица 2). В таблице также перечислены документы, рекомендуемые для фиксирования требований. Часть документов, создаваемых и сопровождаемых при управлении требованиями, определена в соответствии с ГОСТ, а часть готовится по специальным шаблонам, ориентированным на применение для широкого класса проектов.

Таблица 2. Стадии и этапы создания АС и оформляемые документы

№ стадии создания АС	Стадии создания АС	Этапы создания АС	Документы
01	Формирование требований к АС	1.1. Обследование объекта и обоснование необходимости создания АС. 1.2. Формирование требований пользователя к АС.	Интервью заказчиков и пользователей о проблемах
02	Разработка концепции	2.1. Изучение объекта. 2.2. Проведение	Концепция

№ стадии создания АС	Стадии создания АС	Этапы создания АС	Документы
		необходимых научно-исследовательских работ. 2.3. Разработка вариантов концепции АС, удовлетворяющего требованиям пользователя.2.4. Оформление отч ета о выполненной работе.	
03	Техническое задание	3.1. Разработка и утверждение технического задания на создание АС.	ТЗ по ГОСТ 34.602 - 89; Описание автоматизируемых функций РД 50-34.698-90 п. 2.5; Словарь терминов; Модель требований; Описание модели требований
…			
08	Сопровождение АС	8.1. Выполнение работ в соответствии с гарантийными	Интервью заказчиков и пользователей о проблемах;

№ стадии создания АС	Стадии создания АС	Этапы создания АС	Документы
		обязательствами. 8.2. Послегарантийное обслуживание	ТЗ по ГОСТ 34.602 - 89; Запрос на изменение требований; Описание автоматизируемых функций РД 50-34.698-90 п. 2.5; Словарь терминов Модель требований Описание модели требований

4.7. Процесс управления требованиями

На основании регламентных процедур реализуется процесс управления требованиями (рис. 3, таблица 3).

Таблица 3. Описание процесса управления требованиями

№	Входная/Выходная информация	Шаг процесса	Участник	Бизнес правила
01	Методы выявления требований, результаты	Выявление требований и их структурирование	Системный аналитик	Процедуры выявления требований

	обследования объекта автоматизации, план управления требованиями/Список выявленных требований			
02	Список выявленных требований/Модель требований и ее описание	Моделирование требований	Системный аналитик	Шаблон модели требований и его описание
03	Список выявленных требований/Документы с требованиями	Документирование требований	Системный аналитик, Специалист по требованиям	Шаблоны документов
04	Документы с требованиями, модель требований и ее описание. Документы и модели с требованиями верифицированные и утвержденные	Верификация и утверждение требований	Эксперт предметной области, Рецензент	Процедура верификации и утверждения требований
05	Документы и модели с требованиями верифицированные и	Определение базовой версии требований	Архитектор	Процедура верификации и

	утвержденные /Базовая версия требований			утверждения требований (раздел, посвященный определению базовой версии требований)
06	Базовая версия требований/ Измененные требования	Изменение требований	Все заинтересованные лица	Процедура управления изменениями

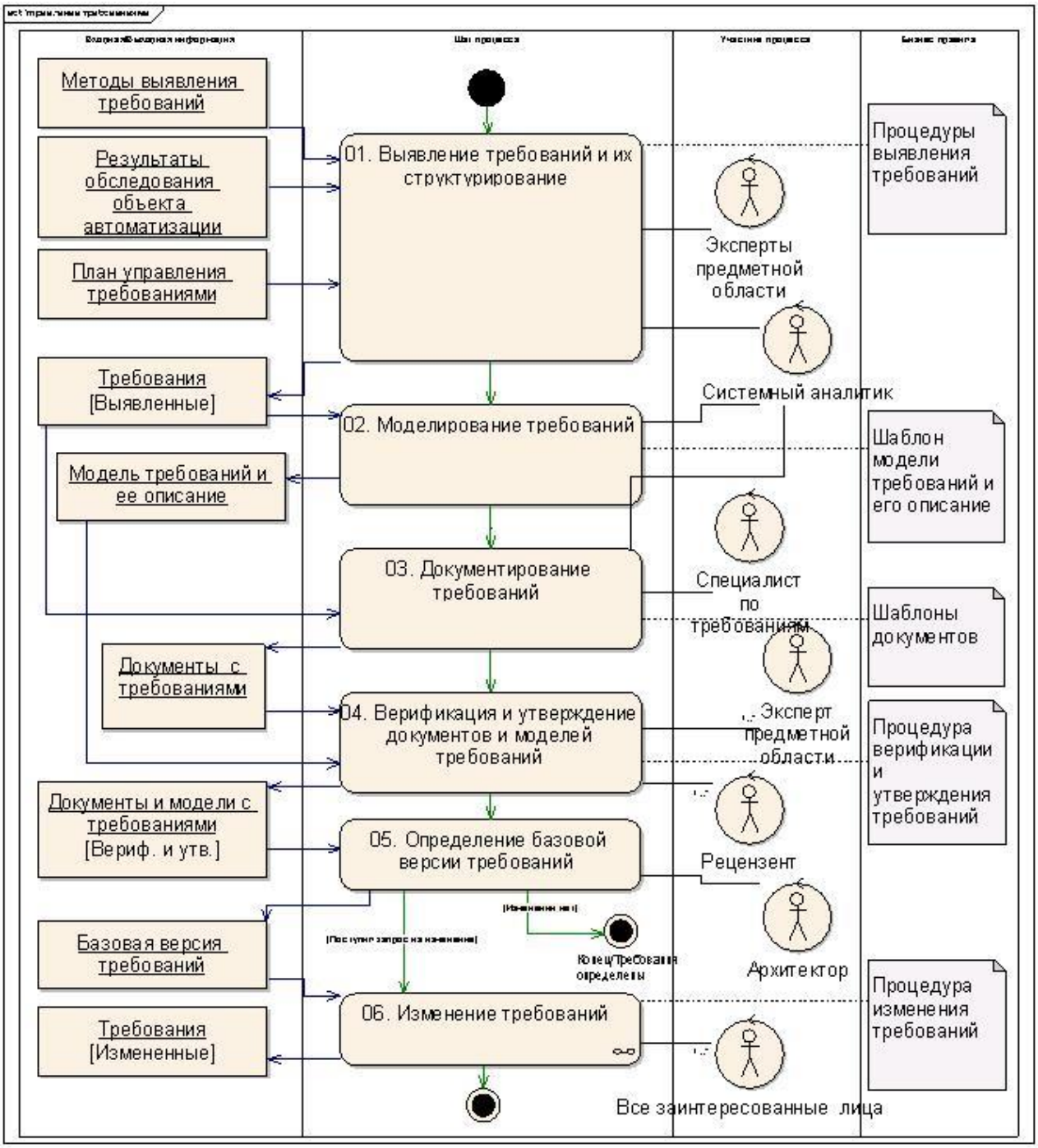


Рис. 4.5. Процесс управления требованиями

4.8. Пример управления требованиями: выявление

Выбор конкретного метода выявления требований (бизнес-моделирование, интервьюирование, создание прототипов) зависит от типа проекта. Выявление требований проводится системным аналитиком с привлечением экспертов по предметной области. Результатом этой деятельности являются требования,

записанные в согласованном формате и структурированные в соответствии со своими типами, как представлено в плане управления требованиями. Например, выявление функциональных требований на основе описания бизнес-процессов проводится следующим образом. Каждому бизнес-процессу ставится в соответствие подсистема в разрабатываемой системе, каждому шагу бизнес-процесса – функциональное требование. На рис. 4.4 представлен состав бизнес-процессов, а на рис. 4.5 – описание конкретного процесса «Зачисление студентов в университет».

Как видно из рис. 4.4 автоматизируемыми процессами являются: зачисление; перевод; отчисление; проведение сессии; подготовка отчетов. Как видно из рис. 4.3, шагами бизнес-процесса зачисления, подлежащими автоматизации являются: формирование списков групп; заполнение личной карточки студента; регистрация выдачи зачетной книжки в журнале.

На основе состава и шагов бизнес-процесса, подлежащих автоматизации, строится матрица трассировки (табл. 4, 5), которая позволяет проследить связи бизнес-процессов с реализующими их подсистемами и конкретных шагов бизнес-процессов с функциональными требованиями, а также контролировать полноту и целостность реализации: каждому автоматизируемому бизнес-процессу должна быть поставлена в соответствие подсистема (подсистемы), а подсистема должна реализовывать какой-либо процесс, соответственно.

Таблица 4. Зависимость подсистем от бизнес-процессов

№	Бизнес-процесс	Подсистема
1	Зачисление студента	Зачисление студента
2	Перевод студента	Перевод студента
3	Отчисление студента	Отчисление студента
4	Подготовка отч етов	Подготовка отч етов

5	Проведение сессии	Проведение сессии

Таблица 5. Зависимость функциональных требований подсистемы «Зачисление студента» от шагов бизнес-процесса «Зачисление студента»

№	Шаг бизнес-процесса	Функциональное требование
1	Формирование списков групп	Формирование списков групп
2	Заполнение личной карточки студента	Ведение личных карточек студентов
3	Регистрация выдачи зачетной книжки	Ведение журнала регистрации выдачи зачетных книжек

На основе данных из таблиц 3, 4 создается модель подсистем и функциональных требований, как представлено на рис. 4.5, 4.6.

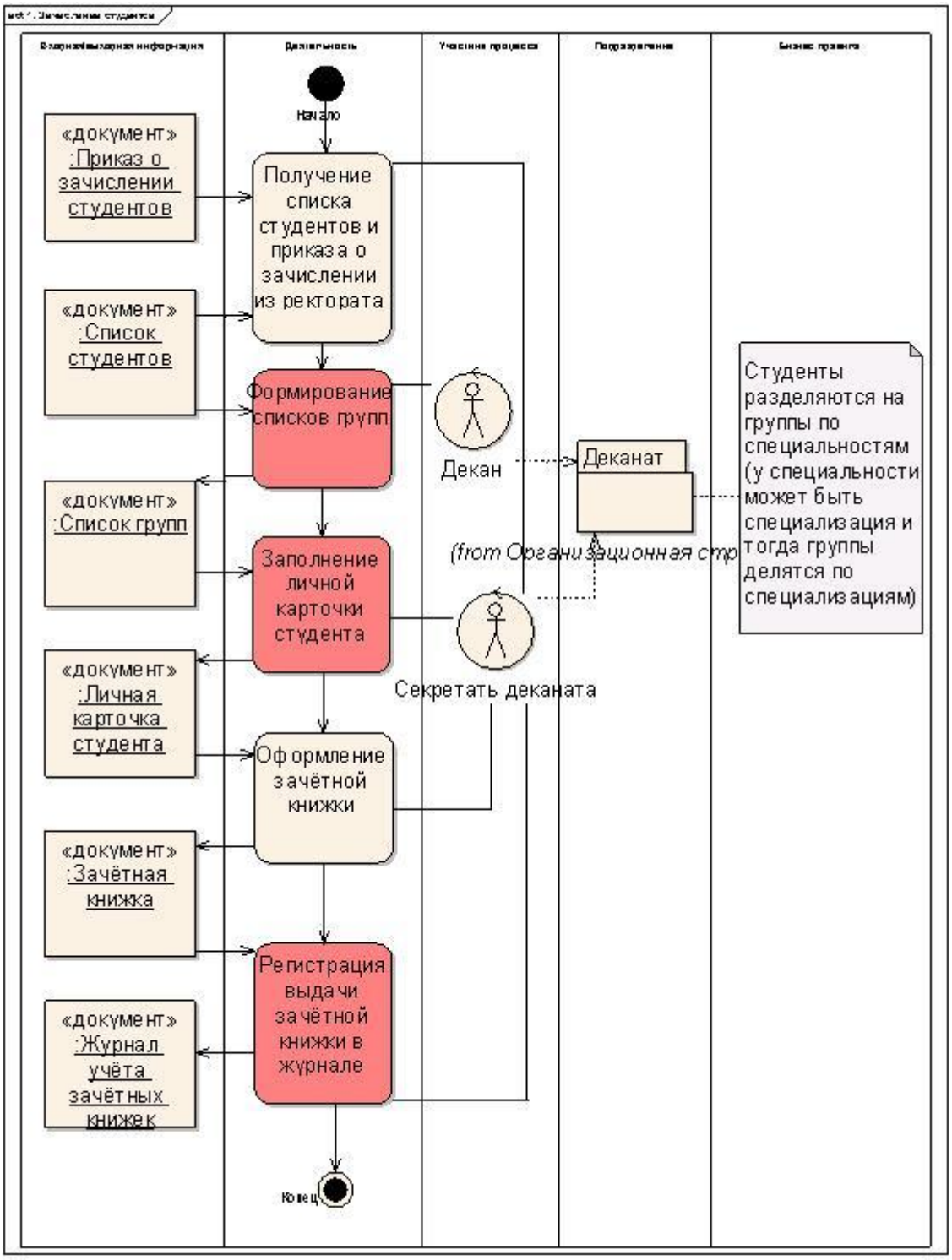


Рис. 4.5. Пример управления требованиями: выявление а) Состав бизнес-процессов; б) Описание бизнес-процесса «Зачисление студентов в университет».

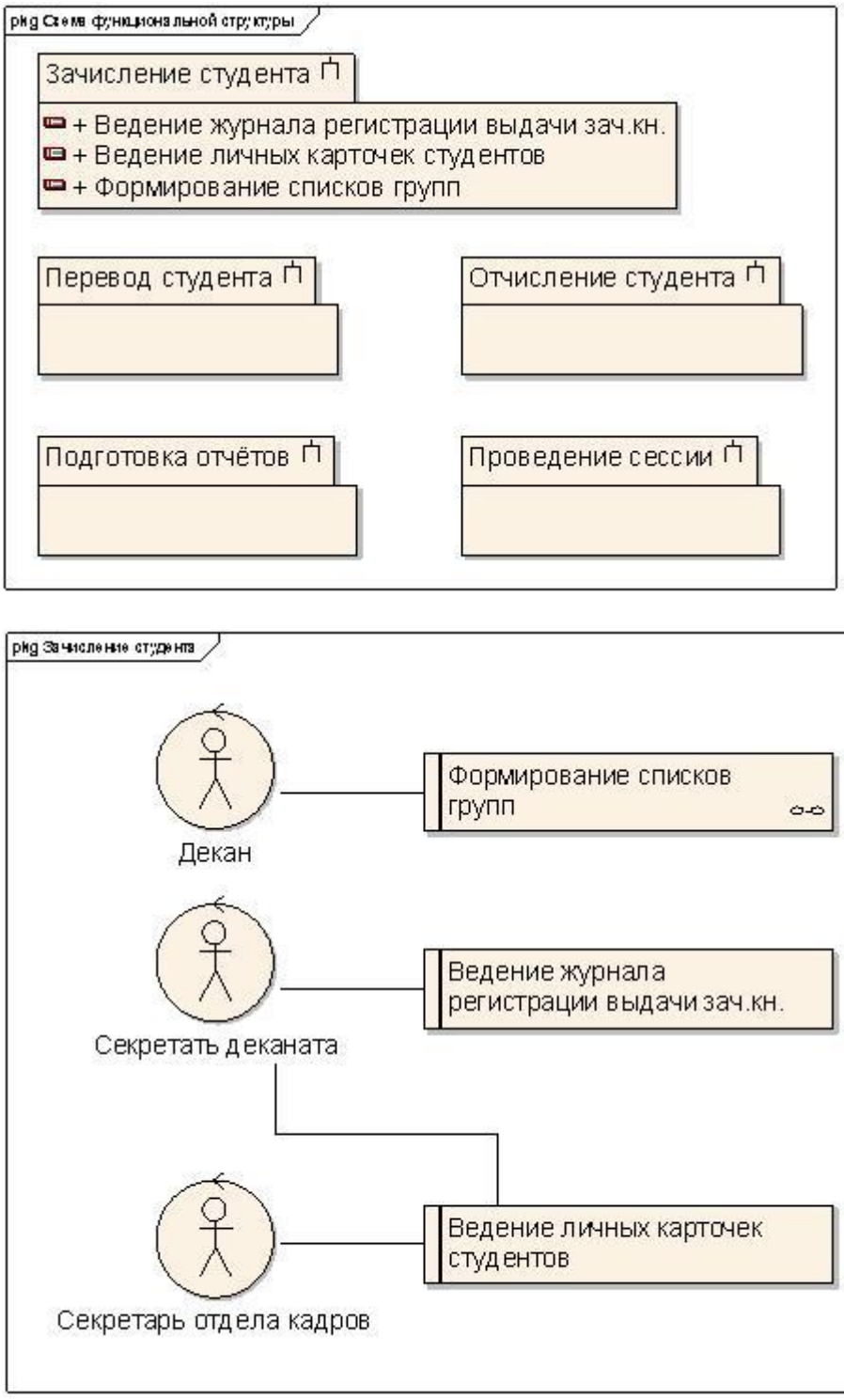


Рис. 4.6. Пример управления требованиями: выявление а) Состав подсистем; б) Состав функциональных требований подсистемы «Зачисление студентов».

Реальное применение методики основывается, прежде всего, на шаблонах проектных документов, методических рекомендациях, опыте ее использования в различных проектах по разработке АС. Сегодня данная методика применяется в проектах газотранспортной и банковской отраслей, а также в автомобильном

бизнесе.

ЛИТЕРАТУРА

1. Леонов И.В. Введение в методологию разработки программного обеспечения при помощи Rational Rose. Требования к системе и способы использования. [Электронный ресурс]. – Режим доступа: http://www.interface.ru/fset.asp?Url=/case/roser.htm.
2. Вигерс К.И. Разработка требований к ПО. – Русская редакция Microsoft. М: 2004. –575с.
3. Pamela Zave, Michael Jackson, «Four Dark Corners of Requirements Engineering», ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 1, 1997, с. 1–30.
4. Jacobson I., Griss M., Jonsson P. Software Reuse. – N.-Y. – Addison-Wesley, 1997. – 497p.
5. Методы и средства инженерии программного обеспечения. [Электронный ресурс]. – Режим доступа: http:/www.rational. com.uml.html.
6. Francisco A. C. Pinheiro, Joseph A. Goguen, «An Object - Oriented tool for Tracing Requirements», «Software», Mach 1996, № 3, с. 31-38.
7. Microsoft Solutions Framework 3.0. [Электронный ресурс]. – Режим доступа: http://www.microsoft.com/rus/ msdn/msf/default.mspx.
8. Методология CDM – метод разработки информационных систем фирмы ORACLE. ORACLE MAGAZINE - №2(4) 1997, с 25-31.
9. Rational Unified Process Version 2003.06.00 – методология разработки программного обеспечения. [Электронный ресурс]. – Режим доступа: http://www.open-ones.com/rup/process/about.htm
10. Липаев В.В., Документирование сложных программных средств. Серия "Информатизация России на пороге XXI века". – М.: СИНТЕГ, 1998. – 220 с.

5. УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЕКТОМ

5.1. Основные понятия и определения

Понятие управление

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

УПРАВЛЕНИЕ – элемент, функция организованных систем различной природы (биологических, социальных, технических), обеспечивающая сохранение их определенной структуры, поддержание режима деятельности, реализацию их программ и целей.

УПРАВЛЕНИЕ – руководство, направление чьей-либо деятельности (www.mega.km.ru).

УПРАВЛЕНИЕ – изменение состояния объекта, системы или процесса, ведущее к достижению поставленной цели (словарь по кибернетике).

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

- наличие цели управления;
- наличие (возможность) управляющего воздействия;
- наличие измерений состояния объекта или процесса;
- ограниченность управления.

Понятие проект

Слово «проект» имеет достаточно много значений. Происходит от латинского projectus, что означает «брошенный вперед». В последнее время слово «проект» употребляется достаточно часто (и часто всуе): проект озеленения улиц города, проект повышения квалификации сотрудников, проект реорганизации деятельности фирмы и т.д.

Под проектом обычно понимается некоторый достаточно сложный вид деятельности, управление которым является также достаточно сложно и при удаче может принести хороший результат. Известны несколько определений проекта:

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

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

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

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

- Цель проекта. Наличие четко выраженного конечного результата, выхода, продукции, определяемых в терминах затрат, качества и времени реализации.
- Уникальность. Проект – это разовое начинание, которое не будет повторяться. Даже “повторяющиеся” проекты, например, по строительству еще одного предприятия по той же проектной документации, значительно отличаются друг от друга использующимися ресурсами и средой реализации.
- Ограниченность во времени. Проект имеет начало и конец. Для его реализация необходима временная концентрация ресурсов. По минованию надобности, ресурсы используются на другие цели.
- Ограниченность ресурсов, выделяемых на выполнение проекта (финансовых, людских, материальных).
- Сложность. Для достижения целей проекта необходимо решить множество задач. Отношения между задачами могут быть довольно сложными, особенно, если в проекте много задач.

- Неопределенность. Возможность достижения цели в указанные сроки с выделенными ресурсами заранее не гарантирована.
- Предсказуемость. По мере реализации проекта, изменяется потребность в тех или иных ресурсах. Это изменение идет в определенной предсказуемой последовательности, определяемой жизненным циклом проекта.

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

5.2. Управление проектами

Известны несколько определений управления проектами:

- Набор проверенных принципов, методов и методик, применяемых для эффективного планирования, составления графика, управления и отслеживания результатов работы (PMI®)
- Планирование, организация, контроль и управление ресурсами компании, выделенными в рамках определенного проекта. (Керцнер – Kerzner)
- Специализация общего менеджмента, определяющего применение стандартных руководящих навыков планирования, организации, комплектования персоналом, продвижения, а также управления и контроля для достижения определенных целей проекта (Фатрелл).

Наиболее полным можно считать определение, сформулированное PMI в Своде знаний по управлению проектами (PMBOK): «Управление проектом (Project Management – PM) – это наука и искусство руководства и координации людских и материальных ресурсов на протяжении жизненного цикла проекта путем

применения современных методов и техники управления для достижения определенных в проекте результатов по составу и объему работ, стоимости, времени, качеству и удовлетворению участников проекта».

Управление проектом основано на двух китах (принципах):

1. Умение – знание принципов и методов управления проектом (планирования, организация, составление графиков, контроль, управление и отслеживание).
2. Навыки – опыт в области управления – применение умения для достижения целей в конкретных условиях.

История управления проектами

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

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

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

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

В настоящее время в США почти не осталось компаний, которые не используют формальные методы управления проектами. В России формальные методы в проектах использует незначительное число предприятий. Большинство из этих инноваторов работают на рынке информационных технологий. Согласно

исследованию, проведенному консалтинговой компанией Interthink, 97,5 % компаний в США и Канаде используют формализованные подходы к управлению проектами, а 22,5 % компаний используют полностью проектно-ориентированный подход для всех своих проектов.

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

5.3. Управление проектами. Определения и концепции

Классическое управление проектами [1] выделяет два вида организации человеческой деятельности: операционная и проектная.

Операционная деятельность применяется, когда внешние условия хорошо известны и стабильны, когда производственные операции хорошо изучены и неоднократно испытаны, а функции исполнителей определены и постоянны. В этом случае основой эффективности служат узкая специализация и повышение компетенции. «Если водитель трамвая начнет искать новые пути, жди беды».

Там, где разрабатывается новый продукт, внешние условия и требования к которому постоянно меняются, где применяемые производственные технологии используются впервые, где постоянно требуются поиск новых возможностей, интеллектуальные усилия и творчество, там требуются проекты.

Проект – временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов.

У операционной и проектной деятельности есть ряд общих характеристик: выполняются людьми, ограничены доступностью ресурсов, планируются, исполняются и управляются. Операционная деятельность и проекты различаются, главным образом, тем, что операционная деятельность – это продолжающийся во времени и повторяющийся процесс, в то время как проекты являются временными и уникальными.

Ограничение по срокам означает, что у любого проекта есть четкое начало и четкое завершение. Завершение наступает, когда достигнуты цели проекта; или осознано, что цели проекта не будут или не могут быть достигнуты; или исчезла необходимость в проекте, и он прекращается.

Уникальность также важное отличие проектной деятельности от операционной. Если бы результаты проекта не носили уникальный характер, работу по их достижению можно было бы четко регламентировать, установить производственные нормативы и реализовывать в рамках операционной деятельности (конвейер). Задача проекта – достижение конкретной бизнес-цели. Задача операционной деятельности – обеспечение нормального течения бизнеса.

Проект – это средство стратегического развития (рис. 5.1). Цель – описание того, что мы хотим достичь. Стратегия – констатация того, каким образом мы собираемся эти цели достигать. Проекты преобразуют стратегии в действия, а цели в реальность.

Рисунок 5.1. Проект – средство стратегического развития.

Таким образом, каждая работа, которую выполняет конкретный сотрудник, привязывается к достижению стратегических целей организации.

Проекты объединяются в программы. *Программа* – ряд связанных друг с другом проектов, управление которыми координируется для достижения преимуществ и степени управляемости, недоступных при управлении ими по отдельности.

Проекты и программы объединяются в портфели. *Портфель* – набор проектов или программ и других работ, объединенных вместе с целью эффективного управления данными работами для достижения стратегических целей.

Проекты и управление ими существовали всегда. В качестве самостоятельной области знаний управление проектами начало формироваться в начале XX века. В этой дисциплине пока нет единых международных стандартов. Наиболее известные центры компетенции:

■ PMI, Project Management Institute, PMBOK – американский национальный стандарт ANSI/PMI 99-001-2004.

■ IPMA, International Project Management Association. В России – СОВНЕТ.

Примерно 50 лет назад человечество начало жить в новой общественно-экономической формации, которая называется информационное или постиндустриальное общество. Мы живем в эпоху перемен, глобализации и интеллектуального капитала.

Эпоха перемен. Все в мире стало непрерывно и стремительно изменяться. Изобилие стало причиной острейшей конкуренции. Инновации – неотъемлемый атрибут нашего времени. «Если у вас медленный доступ в Интернет, вы можете навсегда отстать от развития информационных технологий». Практика должна постоянно перестраиваться применительно к новым и новым условиям.

Глобализация. Всеобщая взаимозависимость и взаимосвязанность. Транснациональные компании. Бизнес идет туда, где дешевле рабочая сила. Интернет. Конкуренция без границ. Пример. Google. За ночь любой из нас в принципе может создать многомиллионную компанию у себя в гараже. С помощью Интернета вы можете выйти на рынок, на котором более 100 млн. потребителей.

Все решают таланты. Простая мобилизация средств и усилий уже не может обеспечить прогресс. Вспомним Ф. Брукса [3], «Если проект не укладывается в сроки, то добавление рабочей силы задержит его еще больше». Идею богатства теперь связывают не с деньгами, а с людьми, не с финансовым капиталом, а с «человеческим». Рынок труда превращается в рынок независимых специалистов и его участникам все больше известно о возможных вариантах выбора. Работники интеллектуального труда начинают самостоятельно определять себе цену.

Человечеству известны два вида деятельности. *Репродуктивная* деятельность (труд) является слепком, копией с деятельности другого человека либо копией своей собственной деятельности, освоенной в предшествующем опыте. Такая деятельность, как, например, труд токаря в любом механическом цеху, или рутинная повседневная деятельность менеджера-управленца на уровне раз и навсегда усвоенных технологий. *Продуктивная* деятельность (творчество) – деятельность, направленная на получение объективно нового или субъективно нового (для данного работника) результата.

Репродуктивная деятельность уходит в прошлое. В постиндустриальном обществе интеллект – основная производственная сила. Сегодня от 70 до 80 % всего, что сегодня делается людьми, производится при помощи их интеллекта [4]. В любом товаре, сделанном в США, доля зарплаты составляет 70 процентов. Но это в среднем по всем товарам. Что касается разработки ПО, то почти все, что в этой отрасли производится, создается при помощи интеллекта.

Все меньший объем человеческой деятельности может быть организован в виде повторяющихся операций. Традиционный пример операционной деятельности – это работа бухгалтерии. Но жизнь так стремительно изменяется, что сегодня, по утверждению сведущих людей, подготовка и сдача годового финансового отчета каждый раз реализуется как самостоятельный проект.

Проект это основа инноваций. Сделать то, до чего другие компании еще не додумались, сделать это как можно быстрее, иначе это сделают другие.

Предложить потребителю более качественный продукт или такой продукт, потребность в котором потребитель даже не может пока осознать.

5.2.2. Критерии успешности проекта

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

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

Задача проекта – достижение конкретной бизнес-цели, при соблюдении ограничений «железного треугольника» (рис. 5.2). Это означает, что ни один из углов треугольника не может быть изменен без оказания влияния на другие. Например, чтобы уменьшить время, потребуется увеличить стоимость и/или сократить содержание.

Рисунок 5.2. «Железный треугольник» ограничений проекта.

Согласно текущей редакции стандарта PMBOK [1], проект считается успешным, если удовлетворены все требования заказчика и *участников проекта*. Поэтому у проекта разработки ПО сегодня не три, а четыре фактора успеха:

1. Выполнен в соответствие со спецификациями.
2. Выполнен в срок.
3. Выполнен в пределах бюджета.
4. *Каждый участник команды уходил с работы в 18:00 с чувством успеха.*

Этот четвертый фактор успеха должен стать воспроизводимым, если предприятие хочет быть эффективным. Для успешного проекта характерно постоянное ощущение его участниками чувства удовлетворения и гордости за результаты своей работы, чувства оптимизма. *Нет ничего более гибельного для проекта, чем равнодушие или уныние его участников.*

Эффективность это отношение полученного результата к произведенным затратам. Нельзя рассматривать эффективность, исходя только из результативности: чем больше ты производишь, чем больше делаешь, тем выше твоя эффективность. С таким подходом можно «зарезать на ужин курицу, несущую золотые яйца». Затраты не следует путать с инвестициями. Оплата аренды, электроэнергии, коммунальные платежи – затраты. Создание и закрепление эффективной команды – это стратегическое приобретение компании. Обучение участников проекта – инвестиции. Вложение в людей – это увеличение числителя в формуле эффективности. Уход из компании всех профессионалов после проекта, выполненного по принципу «любой ценой», – затраты, причем очень тяжело восполняемые. Нарастающая конкуренция указывает на совершенно четкий тренд в мировой экономике – персонал – это форма инвестиций, активов, которые нужно уметь наращивать, управлять и сохранять. *Сегодня люди – это капитал.*

Современное предприятие обязано относиться к своим работникам так же, как к своим лучшим клиентам. Главный капитал современной компании – это знания. Большая часть этих знаний неотъемлема от их носителя – человека. Те предприятия, которые этого не поняли, не выживут потому, что не смогут быть эффективными. Сегодня эффективное предприятие – это сервис. Предприятие, с одной стороны, предоставляет услуги и продукты своим клиентам, а с другой, – рабочие места для профессионального персонала. Принципы «Одно предприятие на всю жизнь», «Работай продуктивно, а предприятие о тебе позаботится» – уходят в прошлое. Посмотрите на рынок рабочей силы в ИТ – правила устанавливают профессионалы.

Категории управления проектами

Категории (от греч. kategoria высказывание; признак), в философии наиболее общие и фундаментальные понятия, отражающие существенные, всеобщие свойства и отношения явлений действительности и познания. Категории образовались как результат обобщения исторического развития познания и

практики: материя и сознание, пространство и время, причинность, необходимость и случайность, возможность и действительность, и др.

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

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

Не проекты – это …

К непроектам относят те виды деятельности, прямое управление которыми невозможно или достаточно просто. К непроектам можно отнести:

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

  Но программа может разбиваться на отдельные конкретные цели, для которых устанавливаются сроки и выделяются ресурсы. Т.е. программа может разбиваться на отдельные проекты: проект Апполон, проект Союз, проект строительства канала Волга – Амударья, проект (или программа?) поворота северных рек.
- **Выполнение установившегося процесса** – деятельность, которая выполняется многократно и постоянно: конвейерное производство, обработка заказов, ведение бухгалтерии. Имеет конкретную цель (выпуск запланированного количества продукции, получение установленной прибыли, ведение отчетности) и выделенные ресурсы, но не является

уникальной или сложной и не связана с конкретными сроками. Управление такими повторяющимися процессами относительно простое.

Изменение параметров установившихся процессов может превратиться в проект (повышение прибыльности фирмы) с конкретными целями (до 45 %), сроками и выделенными ресурсами.

- **Решение творческой задачи** (научной или художественной). Здесь есть конкретная цель, уникальность и сложность, но, как правило, нет ограничений по времени и ресурсам (объединим усилия нашего коллектива и докажем теорему к Новому году!). Слишком велика степень неопределенности.

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

5.3. PMBOK: 9 областей управленческих знаний

PMBOK (Project Management Body of Knowledge - Свод знаний по управлению проектами) – международный стандарт состава знаний по управлению проектами, который разработан и развивается Институтом Проектного Менеджмента (Project Management Institute – PMI). Известны версии этого стандарта от 1996, 2000 и 2004 гг. PMBOK содержит описания состава знаний по следующим 9 разделам (областям знаний) управления проектами:

1. Управление интеграцией проекта (Integration).
   - Создание плана проекта (Project Plan Development) .
   - Исполнение плана проекта (Project Plan Execution).
   - Контроль изменений в проекте (Integrated Change Control).
2. Управление объемом работ (Scope).
   - Инициирование (Initiation) – формальное принятие решения о начале проекта (следующей фазы проекта);

- Планирование объема работ (Scope Planning) – разработка документа, описывающего объем работ;
- Формализация объема работ (Scope Definition) – декомпозиция всего объема работ (основных необходимых результатов) на мелкие, измеримые задачи;
- Верификация (Scope Verification) – подтверждение объема работ – формальная проверка приемлемости результатов работы;
- Управление изменениями объема работ (Scope Change Control) – контроль и утверждение изменений.

3. Управление временем выполнения (Time).
   - Определение состава работ (Activity Definition) ;
   - Определение взаимосвязей работ (Activity Sequencing);
   - Оценка длительностей работ (Activity Duration Estimating) ;
   - Составление расписания проекта (Schedule Development);
4. Управление стоимостью (Cost).
   - Планирование ресурсов (Resource Planning) – какие ресурсы в каком количестве нужны для работ;
   - Оценка стоимостей (Cost Estimating) – ресурсов, необходимых для работ;
   - Разработка бюджета (Cost Budgeting) – бюджетирование – распределение затрат по компонентам проекта;
   - Контроль стоимости (Cost Control) – управление изменениями бюджета
5. Управление качеством (Quality).
   - Планирование качества (Quality Planning) – определение стандартов качества и средств для их достижения;
   - Обеспечение качества процесса (Quality Assurance) – плановая, регулярная оценка исполнения – проверка производственных процессов;
   - Контроль качества результатов (Quality Control) – мониторинг результатов проекта, определение их соответствия стандартам, выявление и устранение причин несоответствия качества;

6. Управление персоналом (Human Resource).
    - Организационное планирование (Organizational Planning) – идентификация, документирование, и назначение проектных ролей, обязанностей и структуры отчетности;
    - Подбор кадров (Staff Acquisition) – получение необходимых для проекта человеческих ресурсов, назначение персонала в команду проекта;
    - Развитие команды проекта (Team Development) – повышение производительности труда: индивидуальной и команды в целом (улучшение взаимодействия);
7. Управление коммуникациями (Communications).
    - Планирование взаимодействия (Communications Planning) – определение потребностей участников проекта в информации и планирование информационных потоков;
    - Распределение информации (Information Distribution) – регулярное и своевременное обеспечение участников проекта необходимой информацией;
    - Оценка исполнения (Performance Reporting) – сбор и распространение отчетности о текущем состоянии проекта, достигнутом прогрессе и ожидаемых результатах;
    - Административное завершение (Administrative Closure) – создание, распространение (уничтожение) информации, необходимые для формального завершения проекта/фазы.
8. Управление рисками (Risk).
    - Планирование управления рисками (Risk Management Planning);
    - Идентификация рисков (Risk Identification);
    - Качественный анализ рисков (Qualitative Risk Analysis);
    - Количественный анализ рисков (Quantitative Risk Analysis);
    - Планирование реагирования на риски (Risk Response Planning);
    - Мониторинг и контроль рисков (Risk Monitoring and Control);

9. Управление закупками и поставками (Procurement).
   - Планирование закупок (Procurement Planning) – определение какие продуктов и услуг нужны извне;
   - Планирование предложений (Solicitation Planning) – документирование требований к продуктам и услугам от внешних поставщиков;
   - Получение предложений (Solicitation);
   - Выбор поставщиков (Source Selection);
   - Управление контрактами (Contract Administration) – регулирование отношений с поставщиками;
   - Завершение контрактов (Contract Closeout) – подтверждение выполнения, разрешение споров;

5.4. SQI: 34 компетенции IT менеджера

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

Институтом качества ПО (SQI – Software Quality Institute) разработан руководящей документ (Body of Knowledge) для сертификации менеджеров программных проектов (SWPM – SoftWare Project Management). В этом документе содержится список 34 компетенций, которыми должен обладать менеджер программного проекта. Список разделен на три основные категории:

- Методика разработки продукта;
- Навыки управления проектов;
- Навыки управления персоналом.

Методика разработки продукта

1. Процессы оценивания – определение критериев для отбора.
2. Знание стандартов процесса.
3. Определение продукта – идентификация клиентской среды и требований, выдвигаемых к продукту.

4. Оценка альтернативных процессов.
5. Управление требованиями – мониторинг изменения требований.
6. Управление субподрядчиками – планирование, управление и осуществление контроля.
7. Выполнение начальной оценки – оценка степени трудности, рисков, затрат и создание графиков.
8. Отбор методов и инструментов – определение процессов отбора.
9. Подгонка процессов – модификация стандартных процессов с целью удовлетворения требований проектов.
10. Отслеживание качества продукта – контроль качества в процессе разработки продукта.
11. Понимание действий по разработке продукта – изучение цикла разработки ПО.

Навыки управления проектов

12. Создание структуры пооперационного перечня работ.
13. Документирование планов – идентификация ключевых компонент.
14. Оценка стоимости – стоимости завершения проекта.
15. Оценка трудозатрат – необходимых для завершения проекта.
16. Менеджмент рисков – идентификация, определение воздействия, обработка рисков.
17. Отслеживание процесса разработки – контроль процесса разработки.
18. Составление графика – разработка графика и ключевых стадий проекта.
19. Выбор метрических показателей.
20. Отбор инструментов менеджмента проекта – выбор методик и инструментов.
21. Отслеживание процессов – мониторинг совместимости членов команды.
22. Отслеживание хода разработки продукта – мониторинг хода разработки по выбранным метрическим показателям.

Навыки управления персоналом

23. Оценка производительности – оценка действий команды, направленных на повышение ее производительности.
24. Вопросы интеллектуальной собственности – понимание степени влияния критических проблем.
25. Организация эффективных встреч – планирование и проведение.
26. Взаимодействие и общение – с разработчиками, руководством и другими командами.
27. Лидерство – обучение проектных команд для получения оптимальных результатов.
28. Управление изменениями – обеспечение эффективного управления изменениями.
29. Успешное ведение переговоров – разрешение конфликтов и ведение переговоров.
30. Планирование карьерного роста – структурирование и управление ходом реализации карьеры.
31. Эффективное представление – использование письменных и устных навыков.
32. Набор персонала – вербовка и собеседование с членами команды.
33. Отбор команды – высококомпетентных специалистов.
34. Создание команды – формирование, руководство и поддержка эффективной команды.

5.5. Управление командой проекта

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

Джон Макдоналд

Управление программным проектом включает решение трех основных задач:

1. Подбор и управление командой.
2. Выбор процесса.
3. Выбор инструментальных средств.

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

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

Ролевая модель команды

Состав команды определяется опытом и уровнем коллектива, особенностями проекта, применяемыми технологиями и уровнем этих технологий. Выделенные позиции на обязательно представлены конкретными людьми. Это список основных функциональных ролей в команде (ролевая модель команды). В малых командах роли могут совмещаться. В больших – выделяться группы или отделы (отдел проектирования, отдел тестирования, отдел контроля качества, отдел подготовки документации, …).

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

Выделяют следующие основные роли:

- **Менеджер проекта** – главное действующее лицо, обладающее знаниями и навыками, необходимыми для успешного управления проектом. Его основные функции:

- ✓ Подбор и управление кадрами;
- ✓ Подготовка и исполнение плана проекта;
- ✓ Руководство командой;
- ✓ Обеспечение связи между подразделениями;
- ✓ Обеспечение готовности продукта.

- **Проектировщик** – это функция проектирования архитектуры высокого уровня и контроля ее выполнения. В небольших командах функция распределяется между менеджером и разработчиками. В больших проектах это может быть целый отдел. Основными функциями проектирования являются:
  - ✓ Анализ требований;
  - ✓ Разработка архитектуры и основных интерфейсов;
  - ✓ Участие в планировании проекта;
  - ✓ Контроль выполнения проекта;
  - ✓ Участие в подборе кадров.
- **Разработчик** – роль, ответственная за непосредственное создание конечного продукта. Помимо собственно программирования (кодирования) в его функции входит:
  - ✓ Контроль архитектурных и технических спецификаций продукта;
  - ✓ Подбор технологических инструментов и стандартов;
  - ✓ Диагностика и разрешение всех технических проблем;
  - ✓ Контроль за работой разработчиков документации, тестирования, технологов;
  - ✓ Мониторинг состояния продукта (ведение списка обнаруженных ошибок);
  - ✓ Подбор инструментов разработки, метрик и стандартов. Контроль их использования.

- **Тестировщик** – роль, ответственная за удовлетворение требований к продукту (функциональных и нефункциональных). В функции тестировщика входит:
  - ✓ Составление плана тестирования. План тестирования составляет один из элементов проекта и составляется до начала реализации (разработки) проекта. Время, отводимое в плане на тестирование может быть сопоставимо с временем разработки.
  - ✓ Контроль выполнения плана. Важнейшая функция контроля – поддержка целостности базы данных зарегистрированных ошибок. В этой базе регистрируется:
    - кто, когда и где обнаружил, описание ошибки, описание состояния среды;
    - статус ошибки: приоритет, кто разрешает
    - состояние ошибки: висит, в разработке, разрешена, проблемы

    Эта база должна быть доступна всем, т.к. в тестировании принимают участие все члены команды.
  - ✓ Разработка тестов. Самая трудоемкая часть в работе тестировщика. Тестирование должно обеспечить полную проверку функциональности при всех режимах работы продукта.
  - ✓ Автоматизация тестирования включает автоматизацию составления тестов, автоматизацию пропуска тестов и автоматизацию обработки результатов тестирования. В виду важности автоматизации тестирования, иногда вводят нового участника – инженера по автоматизации.
  - ✓ Выбор инструментов, метрик, стандартов для организации процесса тестирования.Организация Бета тестирования – тестирования почти готового продукта внешними тестерами (пользователями). Эту важную

процедуру надо продумать и организовать в случае разработки коробочного продукта.

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

В некоторых случаях функции инженера по качеству возлагаются на тестировщика. На самом деле они шире – два следующих уровня качества. Здесь приведены функции, отличные от функции тестировщика:

- ✓ Составление плана качества. План качества включает все мероприятия по повышению качества (на всех уровнях). Имеет долговременный характер. План тестирования – его оперативная составляющая.
- ✓ Описание процессов. Описание процессов является их формализацией. При описании вводятся метрики процесса, влияющие на качество продукта.
- ✓ Оценка процессов включает регистрацию хода выполнения процессов и оценку значений установленных метрик процессов. Выявление «слабых» мест и выработка рекомендаций по улучшению процессов.
- ✓ Улучшение процессов – переопределение процесса, автоматизация части работ, обучение персонала.

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

организации работ выделяются ключевые процессы и выполняются работы по улучшению этих процессов.

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

Подводя итог, следует отметить, что ролевые модели проектных команд могут быть самыми разнообразными.

Модели организации команд

> Если бы люди обладали последовательностью и постоянством, они могли бы убирать бумаги с рабочего стола, предотвращать кариес, избавляться от лишнего веса, бросать курить, и может быть, даже разрабатывать программное обеспечение, укладываясь в рабочий график.
>
> *Алистэр Коуберн Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения*

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

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

Практически любую методологию можно с успехом применять в каком-нибудь проекте.

Любая методология может привести к провалу проекта.

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

Исследованию вопросов человеческого фактора (Peopleware) уделяется достаточно много внимания. Наиболее известными работами являются:

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

- Все разные – по характеру, темпераменту, активности, целям – нет двух одинаковых людей.
- Все похожие – участие в проекте объединяет людей общностью целей, поиском путей достижения этих целей.
- Различаются по типу:
  - ✓ Индивидуалисты или члены команды;
  - ✓ Генераторы идей или исполнители;
  - ✓ Ответственные или безответственные.
- Постоянны и изменчивы – люди, как правило, проявляют постоянство своих привычек и свойств характера, но при этом способны проявлять «противоположные» качества: индивидуалист – командные качества, исполнитель – генерировать идеи, …
- Многообразны – надо понимать, что многообразие людей является основной гарантией выживания человечества вообще и возможности выполнять ИТ проекты в частности. Если бы все были индивидуалисты или все командники, все генераторы идей или все исполнители, то вряд ли удалось выполнить хотя бы один проект, а мир стал бы ужасен.

  Как же управлять такими людьми?

5.2.1.1. Административная модель (теория X)

Это традиционный стиль управления, связанный с иерархической административно-командной моделью, которую используют военные организации. В основе лежит теория X, которая утверждает, что такой подход необходим, поскольку большинство людей по своей природе не любит работу и будет стремиться избежать ее, если у них есть такая возможность. Однако менеджеры

должны принуждать, контролировать, направлять сотрудников и угрожать им, чтобы получить от них максимальную отдачу. Девиз теории и модели: «Люди делают только то, что вы контролируете». Или в более мягком варианте: «Люди делают то, что они не хотят делать, только если вы их контролируете». В конце концов, теория утверждает, что большинство людей предпочитают, чтобы им говорили, что следует делать и им не придется ничего решать самим.

Характерные черты модели:

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

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

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

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

5.2.1.2. Модель хаоса (теория Y)

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

Характерными чертами модели хаоса являются:

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

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

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

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

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

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

5.2.1.3. Открытая архитектура (теория Z)

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

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

- Адаптация к условиям работы – если делаем независимые модули, то расходимся и делаем, если нужна архитектура базы данных, то собираемся вместе и обсуждаем идеи.

- Коллективное обсуждение проблем, выработка консенсуса и принятие решения – не все могут согласиться, но принятое решение является коллективным и в силу этого – обязательным для всех.
- Распределенная ответственность – отвечают все, кто обсуждал, вырабатывал, принимал.
- Динамика состава рабочих групп в зависимости от текущих задач.
- Отсутствие специализации – участники меняются ролями и функциями и могут при необходимости заменить друг друга.
- Задача менеджера – активное (но рядовое, не руководящее) участие в процессе, контроль конструктивности обсуждений, обеспечение возможности активного участия всех.

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

Общение в команде

5.2.1.4. Коммуникации

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

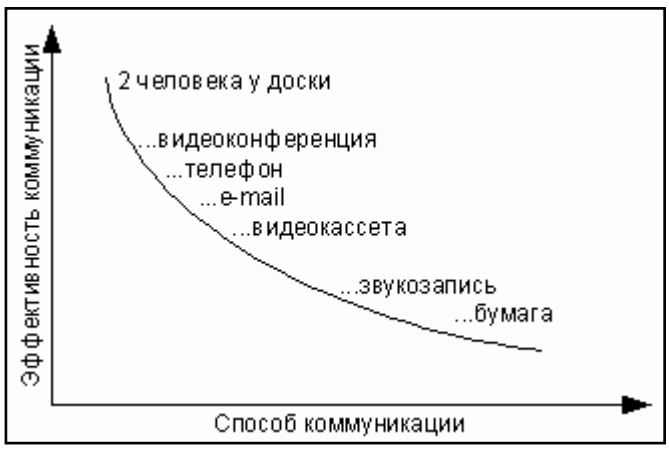


Рис. 5.3 Кривая эффективности.

На рис. 5.3 изображена некая кривая, иллюстрирующая эффективность различных способов общения. Кривая является обобщением ряда исследований в этой области. Видно, что эффективность общения падает по мере возрастания степени его формализованности. С одной стороны, в этом нет ничего удивительного – старый принцип бюрократа гласит: хочешь получить отказ – пиши письмо, хочешь получить обещание – звони по телефону, хочешь добиться результата – езжай сам.

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

5.2.1.5. Принятие решений – компромисс и консенсус

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

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

Начнем с определений.

Компромисс – соглашение, достигнутое посредством взаимных уступок.

Консенсус (коллективное мнение) – общее для конкретной группы мнение

В чем же разница?

Компромисс:

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

Консенсус:

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

Л. Константин [3] приводит следующий пример компромисса и консенсуса. При обсуждении вопроса о размещении кнопок панели инструментов выдвинуты два варианта: горизонтально и вертикально. Компромисс – по диагонали (нелепое решение). Консенсус – настраиваемая пользователем панель (лучшее решение, включающее оба варианта на основе новой идеи – настраиваемая панель).

5.2.1.6. Как добиться консенсуса?

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

- Вера в достижение консенсуса – каждый член команды должен доверять другим в том, что обсуждение приведет к поиску оптимального решения, а не к борьбе личностных мнений. Создание такой атмосферы взаимного доверия является важнейшим в создании эффективной команды. Следует понимать, что взаимное доверие появляется не само по себе, а является результатом:
  - ✓ Нескольких удачных консенсусов;
  - ✓ Участием всех в выработке и принятии оптимальных решений;
  - ✓ Созданием у каждого осознания причастности к принятым решениям.

- Не позиция, а варианты решений – на обсуждение люди должны приходить не со сформированной позицией (ни шагу назад), а с вариантами возможных решений.
- Объективность принимаемых решений как попытка ограничить проявления чувств и эмоций при обсуждении вопросов. Чувства и эмоции являются неотъемлемым свойством человеческой природы. Избежать их полностью вряд ли удастся, но для приведения их «в норму» можно использовать следующие правила:
  - ✓ Критерии оценки вариантов – для объективности обсуждения крайне важно заранее договориться о критериях оценки – установить список критериев и выполнить их ранжировку по степени важности.
  - ✓ Разделение фактов и мнений. Факты – объективные показатели, выраженные в большинстве случаев количественно (но не обязательно): быстродействие, время отклика. Мнения – то, что не основано на фактах. Мнениями не следует пренебрегать, т.к. они часто основаны на опыте, интуиции.
- Замена позиций – в случае, когда обсуждение все же заходит в тупик, бывает полезно предложить участникам изменить точку зрения: «перечислите, пожалуйста, сильные стороны варианта Вашего оппонента и слабые стороны Вашего варианта».
- Слегка управляя – роль руководителя в достижении консенсуса состоит в том, чтобы дать всем возможность высказаться и предложить свои варианты, оставляя свое мнение напоследок или не высказывать его совсем. Руководитель должен быть нейтрален. Руководитель (лидер) может принимать активное участие в обсуждении, но только на правах равного и поручить в этом случае руководство собранием другому человеку.

Корпоративная политика

Представим себе ситуацию, когда одна команда разрабатывает коробочный проект. Проект идет успешно. Даже блестяще: подобралась слаженная команда

профессионалов, было найдено красивое архитектурное решение, учитывающее возможность широкого изменения требований, разработан оригинальный интерфейс, успешно использовано большое количество ранее созданных компонент и т.д. и т.д. Но финансирование проекта было прекращено руководством фирмы. Блестящий проект был признан бесперспективным. Попытки выяснить «истинные» причины успеха не имели. Активным «выяснятелям» намекнули, что они могут попасть под очередное сокращение кадров.

Что делать в такой ситуации?

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

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

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

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

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

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

Корпоративная политика – это не только умение лидера проекта ладить с начальством. Это еще умение всей команды взаимодействовать с другими подразделениями фирмы. Ларри Константин [3] выделяет три измерения, в которых происходят эти взаимодействия: во властной структуре, в структуре задач и в информационной структуре.

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

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

библиотек компонентов, договариваются с другими группами, выторговывая ресурсы и услуги.

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

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

5.6. Планирование и контроль

> Срок завершения небрежно сверстанного проекта в три раза превышает запланированный.
>
> Срок реализации тщательно спланированного проекта превышает установленный в два раза.
>
> *Законы управления проектами. http://www.nwsta.com/Soft/proj/pm01.php*

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

- Вы должны убедить Заказчика в том, что с вами можно иметь дело. Как?

- «Да мы все кандидаты и доктора наук!» – малоубедительно
- «Мы уже делали что-то подобное» – несколько лучше
- «У нас есть план!» – это уже предмет для дальнейшего разговора

- Проект должен быть предсказуемым. Как сделать его предсказуемым?
  - План – один из основных элементов предсказуемости проекта. Контроль хода выполнения плана позволяет оценить возможность продолжения проекта.
- Проект имеет элемент неопределенности.
  - Т.е. заранее всего предусмотреть нельзя, в силу чего первоначальные планы обычно и не выполняются. Но при возникновении ранее неучтенных обстоятельств, планы можно и нужно корректировать. Для сохранения предсказуемости проекта.
- План – ничто. Планирование – все!

5.2.1.7. Задачи планирования

Основными функциями планирования являются:

- Преобразование потребностей в управляемые задачи
  - Изначально проект выступает в виде требований, разработанных и согласованных с Заказчиком. Цель планирования – представить его в виде совокупности отдельных задач, выполнение которых можно контролировать.
- Определение необходимых ресурсов
  - Детальные планы позволят Вам определить количество людей, необходимого оборудования и рабочие условия, которые понадобятся для выполнения проекта.
- Координация командной работы над проектом
  - Очень часто выполнение проекта разбивается на отдельные работы, которые можно выполнять параллельно. Планы делают возможной координацию путем определения того кто, что и когда делает.
- Оценка потенциальных рисков

- Хотя некоторые риски могут быть выявлены во время формулировки требований, гораздо больше их обнаруживается после осуществления детального планирования. Знание о существовании этих рисков позволит Вам раньше их заметить (если они осуществились) и приготовиться к их адресации.

- Сигнализация о возникновении проблем
  - Отклонение от плана – сигнал о возникновении проблемы. Планы – это не догма, которой необходимо безоговорочно следовать. Для менеджера проекта они скорее являются предположениями и основой для сравнения. Если выполнение проекта не оправдывает ожиданий, то необходимо провести соответствующую корректировку плана.

Что надо планировать?

При планировании выполнения проекта надо найти ответы на следующие вопросы:

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

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

5.2.1.8. Как проверять и оценивать?

Прежде всего, определим, что надо проверять и оценивать:

- Общий ход выполнения проекта;
- Выполнение отдельных видов работ;
- Работу отдельных исполнителей;
- И т.д.

Проверять и оценивать можно по-разному:

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

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

5.2.1.9. Метрики проекта

Объективно оценить и проконтролировать можно только то, что можно измерить. Для объективной оценки необходимо вводить метрики проекта – количественные показатели оценки различных характеристик проекта и процесса

его выполнения. Метрики могут вводиться как для всего проекта в целом, так и для отдельных видов работ. Общими метриками проекта являются:

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

Как надо планировать?

5.2.1.10. Когда начинать планировать?

1. В самом начале проекта?
2. Когда сформулированы требования и ясен объем работ?
3. Когда выполнение проекта выходит из под контроля и проект надо «ввести в берега»?

Начнем с того, что «правильного» ответа на этот вопрос не существует: есть проект и есть проект. Если у вас небольшая, слаженная команда профессионалов, то … (см. технологию XP, где планирование сведено к разумному минимуму). Если у вас большой проект, большая команда, ограниченные сроки, то планировать придется. И когда начинать?

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

Если проект сложный, много распределенных ресурсов, то планировать надо начинать с самого начала проекта. Могут спросить: как планировать то, что пока еще не известно? Ведь в начале проекта мы еще не знаем даже того, что нам предстоит сделать? Здесь следует вспомнить, что план ИТ-проекта – это не догма.

Планирование – циклический это процесс составления, оценки и корректировки плана. В сложных случаях этот процесс надо запускать как можно раньше.

5.2.1.11. Структурная декомпозиция работ

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

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

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

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

5.2.1.12. Создание СДР

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

1. Определите основные цели проекта.
2. Определите функциональные требования, которые удовлетворяют целям проекта.

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

5.2.1.13. Критерии СДР

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

- Целенаправленность
    - Все деятельности должны быть направлены на достижение единой цели проекта и вести к конечному результату.
- Независимость
    - Между деятельностями в рамках проекта очень часто существует множество зависимостей. Управлять и контролировать такие деятельности достаточно сложно. Для повышения управляемости

проектом любая деятельность должна быть определена до такого уровня детальности, что она будет завершена без необходимости активной координации с результатами других деятельностей. Такой ситуацией лучше всего управлять путем разбиения работы на равнозависимые подмножества.

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

Стандарты планирования

В настоящее время вышли несколько международных стандартов по планированию проектов:

1. IEEE Std 1058-1998 «IEEE Standard for Software Project Management Plans».
2. IEEE Std. 1228-1994. IEEE Standard for Software Safety Plans.
3. IEEE Std. 1059-1993. IEEE Guide for Software Verification and Validation Plans.
4. IEEE Std. 730-2002. IEEE Standard for Software Quality Assurance Plans.
5. IEEE Std. 828-1998. IEEE Standard for Software Configuration Management Plans.

5.5. Средства управления проектом

> Система календарного планирования позволяет руководителю компании понять, как эффективно используются ресурсы, а также помогает при планировании видеть ясную картину происходящего.
>
> *Stephen Fulkerson, Process Architect, Planview. Austin, Texas, USA*

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

Если менеджер проекта не силен в теории управления проектами, то обязательно возникнет вопрос целесообразности использования системы для управления проектами. «Программные продукты не помогут неопытным менеджерам продуктов успешно управлять проектом» говорит Robert A. Edwards, technical vice president for Welcom, Houston, Texas, USA. Он также предостерегает, что программный продукт не разрешит всех проблем. Программный продукт может только помочь, но решать проблемы будут люди.

«Система не заменит практический опыт. Вы можете иметь отличный программный продукт, но если вы не владеете методологией планирования, то можете и не уложиться в сроки» говорит Pedro Contreas, planning and reporting supervisor for production project management department of SINCOR, Caracas, Venezuela. «Хорошая команда проекта может добиться успеха и с менее полной и дорогой системой. В то время как плохая команда проекта может провалить проект,

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

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

Функции систем управления проектами

Инструментальные средства управления проектом должны поддерживать следующие основные функции:

- Комплекс работ, связей и временных характеристик. Средства описания комплекса работ проекта, связей между работами и их временных характеристик должны включать:
    - Описания глобальных параметров планирования проекта;
    - Описание логической структуры комплекса работ;
    - Многоуровневое представление проекта;
    - Назначение временных параметров планирования задач;
    - Поддержка календарей отдельных задач и проекта в целом.
- Информация о ресурсах и затратах. Средства поддержки информации о ресурсах и затратах по проекту и назначения ресурсов и затрат отдельным работам проекта должны обеспечивать решение следующих задач:
    - Организационная структура исполнителей;

- Ведение списка наличных ресурсов, номенклатуры материалов и статей затрат;
- Поддержка календарей ресурсов;
- Назначение ресурсов работам;
- Календарное планирование при ограниченных ресурсах.

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

- настроить универсальное программное обеспечение на специфику управления проектами в конкретной предметной области (например, интеграция со сметными системами для строительных проектов).

Обзор систем управления проектами

К числу наиболее известных систем для управления проектами относятся:

- **MS Excel**. Хорошо подходит для недельного планирования и отчетности.
- **MS Project 2002**. Microsoft Project является на сегодня самой распространенной в мире системой управления проектами. Во многих западных компаниях MS Project стал привычной добавкой к Microsoft Office даже для рядовых сотрудников, которые используют его для планирования графиков несложных комплексов работ. Отличительной особенностью пакета является его простота. Разработчики MS Project не стремятся вложить в пакет сложные алгоритмы календарного или ресурсного планирования.

  Семейство 2002 состоит из следующих продуктов:

  - **MS Project Standard 2002** – рус. Легкая, универсальная система для управления проектами, в том числе для планирования и формирования графиков выполнения проектов. В сочетании с сервером MS Project Server 2002 позволяет наладить коллективную работу над проектом в масштабах рабочей группы на предприятиях разной величины.
  - **MS Project Professional 2002**. Новое приложение. Содержит всю функциональность Microsoft Project Standard 2002 и в сочетании с Microsoft Project Server 2002 обеспечивает поддержку коллективной работы над проектами и предоставляет средства анализа и управления проектами и ресурсами в масштабах крупного предприятия.
  - **MS Project Server 2002** – рус. Очередное пополнение в семействе Microsoft .NET Server, которое в сочетании с Microsoft Project Professional и Microsoft Project Standard обеспечивает полноценную поддержку

коллективной работы над проектами, а также содержит средства анализа и управления ресурсами в масштабах всего предприятия.

- **MS Project Web Access 2002**. Web-интерфейс, предоставляющий доступ к информации о проектах и средствам анализа для руководителей и членов групп, которым не нужен полный набор функций управления в Microsoft Project. Эти пользователи могут обращаться к информации о проектах через Web-браузер. Поставляется в составе Microsoft Project Server 2002.

- **Open Plan**. Производитель Welcom Corp. (США). Дистрибьютор в России ЛАНИТ. Open Plan – полностью руссифицированная система планирования и контроля крупных проектов и программ. Основные отличия системы:
  - мощные средства ресурсного и стоимостного планирования,
  - эффективная организация многопользовательской работы и
  - возможность создания открытого, масштабируемого решения для всего предприятия.

  Open Plan поставляется в двух вариантах – Professional и Desktop – каждый из которых отвечает различным потребностям исполнителей, менеджеров и других участников проекта.

Продукты Primavera Systems, Inc. (США). Дистрибьютор в России ПМСОФТ:

- **Primavera Project Planner**. Центральный программный продукт семейства Primavera, Primavera Project Planner (P3) применяется для календарно-сетевого планирования и управления с учетом потребностей в материальных, трудовых и финансовых ресурсах средними и крупными проектами в самых различных областях, хотя наибольшее распространение данный продукт получил в сфере управления строительными и инженерными проектами.

- **SureTrak Project Manager**. Кроме P3, компанией Primavera Systems поставляется облегченная система для УП – SureTrak. Этот полностью русифицированный продукт ориентирован на контроль выполнения небольших проектов или/и фрагментов крупных проектов. Может работать как самостоятельно, так и совместно с P3 в корпоративной системе управления проектами.

Spider Project. Производитель Spider Technologies Group (Россия). Российская разработка Spider Project отличается мощными алгоритмами планирования использования ограниченных ресурсов и большим количеством дополнительных функций. Система спроектирована с учетом большого практического опыта, потребностей, особенностей и приоритетов Российского рынка. Spider Project поставляется в двух вариантах – Professional и Desktop.

- **Project Expert**. Производитель Про-Инвест Консалтинг (Россия). Российская разработка Project Expert обеспечивает построение финансовой модели предприятия, анализ финансовой эффективности бизнес-проектов, разработку стратегического плана развития и подготовку бизнес-плана.
- **1С-Рарус: Управление проектами**. 1С-Рарус (Россия). Российская разработка на платформе бухгалтерской системы «1С: Предприятие» версии 7.7 служит для планирования, организации, координации и контроля проектных работ и ресурсов. Типовое решение разработано только средствами и методами программы «1С: Предприятие» и представляет собой дополнение к компоненте «Бухгалтерский учет» программы «1С: Предприятие» версии 7.7. 1С-Рарус: Управление проектами интегрируется с любыми конфигурациями, которые используют компоненту «1С: Бухгалтерский учет».

Контрольные вопросы

1. Что такое проект?
2. Назовите 7 основных характеристик проекта.
3. Примеры непроектов и их связь с проектами.

4. Что такое управление и управление проектами?
5. Что такое категории управления проектами?
6. Что за треугольник ограничений проекта?
7. Какие из девяти областей управленческих знаний вы запомнили?
8. Попробуйте дать краткую характеристику каждой из них.
9. На какие три категории разбиты 34 компетенции менеджера IT проекта и почему?
10. Попробуйте дать характеристику каждой из них.
11. Зачем нужны роли в команде?
12. Роли и ответственности команды проекта.
13. Что такое модель управления командой и каковы критерии выбора модели?
14. Какие модели управления командой вы запомнили?
15. В чем их преимущества и в чем недостатки?
16. Какова роль общения в команде?
17. Какие возможны способы общения в команде?
18. Преимущества и недостатки различных способов общения?
19. Чем компромисс отличается от консенсуса?
20. Как достичь компромисса?
21. Как добиться консенсуса?
22. Что такое корпоративная политика?

ЛИТЕРАТУРА

1. Шафер Д, Фатрел Р., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат: Пер. с англ. – М.: Вильямс, 2003. – 1136 с.
2. Салливан Эд. Время – деньги. Создание команды разработчиков программного обеспечения/ Пер.с англ. – М.: Русская редакция, 2002. – 364с.
3. Константин Л. Человеческий фактор в программировании. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 384 с.
4. Коуберн А. Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения. [Электронный ресурс]. – Режим доступа: http://www.optim.ru/cs/ 2002/3/cobern/people.asp.
5. Лапланте Ф. Человеческий фактор в управлении ИТ-проектом. [Электронный ресурс]. – Режим доступа: http://www.info-system.ru/pj_managment/article/pj_people_factor.html.
6. Демарко Т., Листер Т. Человеческий фактор: эффективные проекты и команды. – Пер. с англ. – СПб: Символ-Плюс, 2004 г. – 259 с.

6. КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Понятие «качество» на самом деле не столь очевидно и просто, как это может показаться на первый взгляд.

Качество программного обеспечения – характеристика программного обеспечения (ПО) как степени его соответствия требованиям. При этом требования могут трактоваться довольно широко, что порождает целый ряд независимых определений понятия. Чаще всего используется определение ISO 9001, согласно которому качество есть «степень соответствия присущих характеристик требованиям».

Качество кода может определяться различными критериями. Некоторые из них имеют значение только с точки зрения человека. Например, то, как отформатирован текст программы, совершенно не важно для компьютера, но может иметь серьезное значение для последующего сопровождения. Многие из имеющихся стандартов оформления кода, определяющих специфичные для используемого языка соглашения и задающие ряд правил, улучшающих читаемость кода, имеют своей целью облегчить будущее сопровождение ПО, включающее отладку и обновление. Существуют и другие критерии, определяющие, «хорошо» ли написан код, например, такие, как структурированность – степень логического разбиения кода на ряд управляемых блоков.

- Читаемость кода;
- Легкость поддержки, тестирования, отладки, исправления ошибок, адаптации;
- Низкая сложность кода;
- Низкое использование ресурсов: памяти и процессорного времени;
- Корректная обработка исключительных ситуаций;
- Малое число предупреждений при компиляции и линковке.

6.1. Факторы качества

Фактор качества ПО – это нефункциональное требование к программе, которое обычно не описывается в договоре с заказчиком, но, тем не менее, является желательным требованием, повышающим качество программы.

Некоторые из факторов качества:

1.Понятность. Назначение ПО должно быть понятным, из самой программы и документации.

2.Полнота. Все необходимые части программы должны быть представлены и полностью реализованы.

3. Краткость. Отсутствие лишней, дублирующейся информации. Повторяющиеся части кода должны быть преобразованы в вызов общей процедуры. То же касается и документации.

4. Портируемость. Легкость в адаптации программы к другому окружению: другой архитектуре, платформе, операционной системе или ее версии.

5. Согласованность. По всей программе и в документации должны использоваться одни и те же соглашения, форматы и обозначения.

6. Сопровождаемость. Насколько сложно изменить программу для удовлетворения новых требований. Это требование также указывает, что программа должна быть хорошо документирована, не слишком запутана, и иметь резерв роста по использованию ресурсов (память, процессор).

7. Тестируемость. Позволяет ли программа выполнить проверку приемочных характеристик, поддерживается ли возможность измерения производительности.

8. Удобство использования. Простота и удобство использования программы. Это требование относится прежде всего к интерфейсу пользователя.

9. Надежность. Отсутствие отказов и сбоев в работе программ, а также простота исправления дефектов и ошибок:

10. Структурированность.

11. Эффективность. Насколько рационально программа относится к ресурсам (память, процессор) при выполнении своих задач.

12. Безопасность

6.1.1. Что такое качество?

В различных источниках можно найти различные определения качества:

- Качество продукции – совокупность свойств продукции, обусловливающих ее способность удовлетворять определенные потребности в соответствии с ее назначением. (БСЭ)
- Степень соответствия присущих характеристик требованиям (ISO 9001)
- Качество товара – совокупность потребительских свойств товара. (ГОСТ Р 51303-99)

Качество – это свойство товара (услуги) наиболее полно удовлетворять требованиям и пожеланиям потребителя.

Качество ПО характеризуется тремя главными аспектами: качество программного продукта, качество процессов ЖЦ и качество сопровождения или внедрения. Далее мы будем рассматривать качество процессов создания ПО как главный определяющий фактор для качества ПО.

Одна из глав (область знаний) SWEBOK рассматривает вопросы качества программного обеспечения, выходя за рамки отдельных процессов жизненного цикла. Качество программного обеспечения является постоянным объектом заботы программной инженерии и обсуждается во многих областях знаний (что вполне обосновано, если учесть поистине катастрофический уровень проваленных проектов и неудовлетворенность пользователей программных продуктов, ставшая притчей во языцех для программной индустрии). В общем случае, SWEBOK описывает ряд путей достижения качества программного обеспечения (рис.6.1)

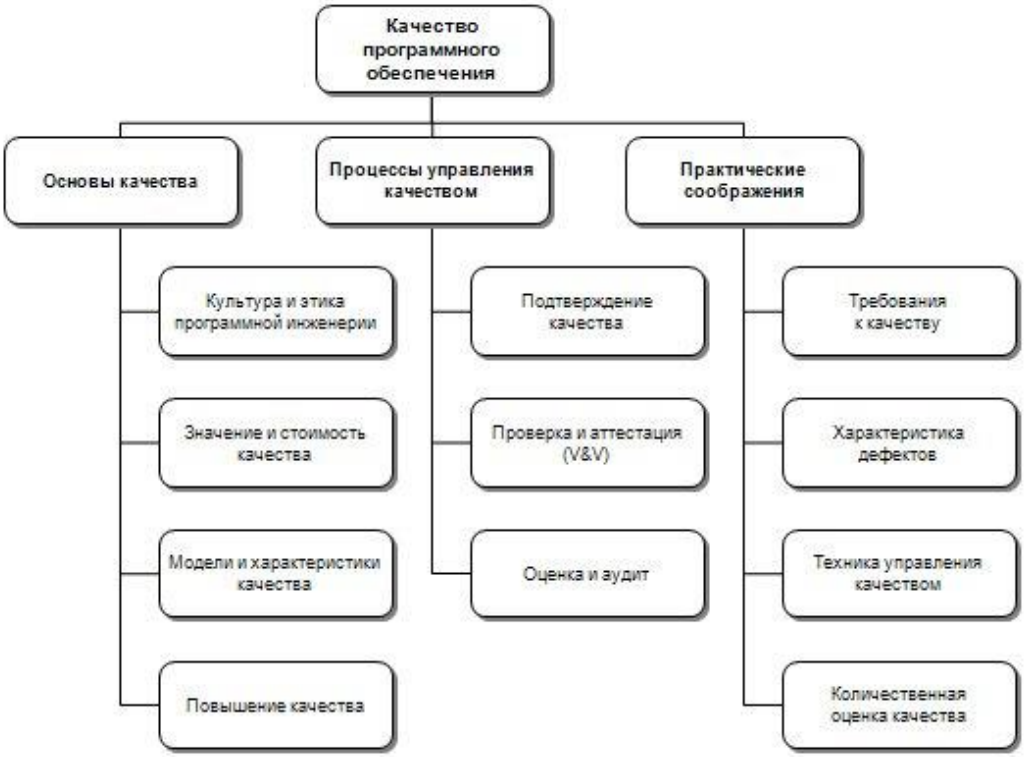


Рисунок 6.1. Область знаний «Качество программного обеспечения».

6.1.2. Мера качества: ценность и стоимость

Введенное понятие качества есть «качественный» показатель продукта. Можно ли качество измерить? Наиболее общим является подход, при котором вводятся понятия:

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

При этом оказывается, что для производителя и потребителя эти показатели имеют различные значения. Для производителя вся продукция, не содержащая дефектов, которые препятствовали бы продаже этой продукции, имеет ценность. Для потребителя же ценность имеют только те свойства продукции, которые соответствуют его ожиданиям. Важными являются три основные соотношения между ценностью и стоимостью:

- Мера качества для потребителя: $\boldsymbol{Q_u = C_u / S_u}$

- Мера качества для производителя: $Q_d = C_d / S_d$
- Конкурентоспособность продукта: $K = C_u / C_d$

6.1.3. Эволюция методов обеспечения качества

На разных этапах развития товарного производства применялись различные методы обеспечения качества. Выделяют три основные фазы эволюции методов обеспечения качества:

- Фаза отбраковки
- Фаза управления качеством
- Фаза планирования качества

Фаза отбраковки.

Началась вместе с зарождением ремесленного производства. Отдельные мастера проверяли свою собственную работу Цеховые организации средневековых городов, которые, если выражаться современным языком, сертифицировали мастеров – присуждали звание мастера после серьезных испытаний качества изделия. При этом каждое изделий было индивидуальным.

Следующий шаг этой фазы был связан с применением стандартов (калибров). В 70-х гг. XIX века в оружейном производстве (заводы Сэмюеля Кольта) родилась идея стандартного качества – изделия собирались не из подогнанных друг к другу деталей, а из случайно выбранных из партии, то есть взаимозаменяемых деталей. Перед сборкой эти детали проверялись с помощью калибров, и негодные отбраковывались. Контроль и отбраковку осуществляли специально обученные контролеры.

В конце XIX века Генри Мартин Леланд («Кадиллак») впервые применил в автомобильном производстве работу по калибрам и придумал пару «проходной» и «непроходной» калибр. В марте 1908 г. эксперты Британского автомотоклуба отобрали случайным образом 3 экземпляра из экспортной партии автомобилей «Кадиллак», прибывшей в Англию, и разобрал их до последнего винтика. Все детали свалили в кучу, а затем кое- какие детали из этой кучи изъяли и заменили запчастями, позаимствованными опять же наугад в местном агентстве по продаже

и обслуживанию автомобилей «Кадиллак». Потом группа механиков, вооруженная только отвертками и гаечными ключами, собрала машины заново и запустила моторы. Две машины завелись с первой попытки, а одна – со второй, и все они отправились на длительную обкатку по только что сданному в эксплуатацию автодрому Бруклэндс. И когда вновь собранные машины подтвердили полную идентичность своих ходовых характеристик параметрам автомобилей заводской сборки, Британский автомотоклуб выдал фирме «Кадиллак» диплом и серебряный кубок с надписью «За стандартизацию". После этого на табличке с гербом фирмы на автомобилях "Кадиллак" появилась надпись «Standart of the world» – образец для подражания для всего мира.

Выходной контроль вместо входного. В начале XX века Форд впервые ввел вместо входного контроля комплектующих на сборке выходной контроль на тех производствах, где эти комплектующие изготавливались, то есть на сборку стали поступать только годные, качественные изделия. Он также создал отдельную службу технического контроля, независимую от производства.

Замена контроля составлял один из элементов производственной системы Форда–Тейлора, которая была разработана Тейлором и внедрена на заводах Форда. Эта система основана на концепции научного менеджмента, включившая системный подход, кадровый менеджмент, идею разделения ответственности между работниками и управленцами в обеспечении качественной и эффективной работы организации, идею научного нормирования труда. В основных чертах просуществовала до настоящего времени и является моделью организации производства большинства современных предприятий. Только в 70-е годы ей на смену стала приходить другая концепция (*производственная система Тойота*).

Концепция фазы отбраковки состоит в том, что потребитель должен получать только годные изделия, т.е. изделия, соответствующие стандартам. Основные усилия должны быть направлены на то, чтобы не годные изделия (брак) были бы отсечены от потребителя.

Результат фазы состоял в том, что численность контролеров стала составлять до 30 – 40 % от численности производственных рабочих, иногда и более. Повышение качества всегда сопровождается ростом затрат на его обеспечение. Т.е. цели повышения эффективности производства и повышения качества изделий являются противоречивыми (не могут быть достигнуты одновременно).

Фаза управления качеством

Цель фазы управления качеством – сосредоточить усилия не на том, как обнаружить и изъять негодные изделия до их отгрузки покупателю, а на том, как увеличить выход годных изделий в техпроцессе. В этой фазе выделяют два этапа:

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

Старт первого этапа: май 1924г. Вестерн Электрик, США. Точкой отсчета считаются работы, выполненные в Отделе технического контроля фирмы Вестерн Электрик, США. В мае 1924 г. сотрудник отдела доктор Шухарт передал своему начальнику короткую записку, которая содержала метод построения диаграмм, известных нынче по всему миру как контрольные карты Шухарта. Контрольные карты основаны на статистических методах оценки стабильности протекания различных технологических процессов. На основе выборок – замеров контрольных показателей процессов (количество брака в контрольной партии) – строится среднее значение и допустимые верхнее и нижнее отклонения. Процесс не должен выходить за допустимые значения. Статистические методы, предложенные Шухартом, дали в руки управленцев инструмент, который позволил контролировать качество производства комплектующих.

Для первого этапа было характерно создание аудиторских служб по качеству, которые в отличие от отделов технического контроля занималась не разбраковкой продукции, а путем контроля небольших выборок из партий изделий проверяла работоспособность системы обеспечения качества на производстве.

Внедрение таких служб контроля качества отдельных процессов значительно повысило эффективность производства, но есть предел, определяемый системой. Каждый производственный процесс имеет определенный предел выхода годных изделий, и это предел определяется не процессом самим по себе, а системой, то есть всей совокупностью деятельности предприятия, организации труда, управления, в которой этот процесс протекает.

Второй этап управления качеством (фаза менеджмента качества) связан с повышением качества путем управления предприятием. Составными элементами этого этапа является:

- Совершенствование системы в целом, а не только отдельных производственных процессов
- Непосредственное участии высшего руководства компаний в проблемах качества
- Обучение всех сотрудников компаний сверху донизу основным методам обеспечения качества
- Упор на мотивацию сотрудников на высококачественный труд
- Концепция «0 дефектов на всех участках".

Начало этапа – 1950 г., Япония. В 1950 доктор Эдвардс Деминг получил приглашение от японского союза ученых и инженеров принять участие в программе восстановления японской промышленности. Там он и предложил программу менеджмента качества из 14 пунктов, разработал принцип постоянного улучшения качества. За 12 лекций доктор Деминг встретился с сотнями ведущих менеджеров японских фирм. Основная идея программы: "Основа качества продукции – качество труда и качественный менеджмент на всех уровнях, то есть такая организация работы коллективов людей, когда каждый работник получает удовольствие от своей работы". Программа была активно воспринята и произвела революцию в японской промышленности.

Основные идеи этой программы были разработаны и пытались применяться в США, но там вначале не нашли должного отклика. Только в Японии они нашли

благодатную почву. После прочтения лекций в Японии, Деминг предрек: «Через несколько лет мир содрогнется от обилия качественных японских товаров». По-видимому, он оказался прав.

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

Второй этап – менеджмент качества отражает современное представление о системе управления качеством. Противоречие между повышением качества и ростом эффективности производства в его прежних формах было преодолено – применение новых идей управления позволило одновременно повышать качество и снижать затраты на производство. Потребитель практически во всех странах стал получать товары и услуги высочайшего качества по доступной цене – идея «общества потребления» воплотилась в жизнь.

Сложилась концепция стандартизованного качества – качество определяет производитель, а покупатель берет товар или не берет. Внутренним противоречием этой фазы является вопрос: что делать при ошибке определения запросов, когда годные товары не находят спроса?

Фаза планирования качества

Цель – планирование запросов. Эта фаза стала зарождаться в середине 60х гг. как развитие идей предыдущей фазы в направлении более полного удовлетворения запросов потребителей.

Предпосылками возникновения фазы планирования качества являются:

- Развитие мирового рынка товаров и услуг;
- Резкое обострение конкуренции на этом рынке;
- Политика государственной защиты интересов потребителей;
- Развитие теории надежности изделий;
- Внедрение вычислительной техники и САПР.

Основы концепции новой фазы:

– Большая часть дефектов изделий закладывается на стадии разработки из-за недостаточного качества проектных работ;
– Математическое моделирование свойств и процессов. Перенос центра тяжести работ по созданию изделия с натурных испытаний опытных образцов или партий на математическое моделирование свойств изделий, а также моделирование процессов производства изделий, что позволяет обнаружить и устранить конструкторские и технологическое дефекты еще до начала стадии производства;
– Снижение цены – высокое качество необходимо предоставить потребителю за приемлемую цену, которая постоянно снижается, т.к. конкуренция на рынках очень высока.

В настоящее время эта фаза только зарождается, и ее концепция еще окончательно не сформировалась.

6.1. ISO9000: система управления качеством

ISO9000 – серия международных стандартов, регламентирующих организацию системы управления качеством. Стандарты серии ISO-9000 универсальны, т.е. применимы к любым предприятиям независимо от сферы деятельности, формы собственности, размеров предприятия.

В этом разделе мы рассмотрим следующие вопросы:

– TQM – фундаментальные требования ISO9000;
– Структура документов ISO9000;
– Как работает система управления качеством?
– Немного об истории ISO9000.

6.2.1. ISO9000. Фундаментальные требования

Фундаментальными требованиями ISO9000 для построения систем управления качеством являются 8 принципов TQM – Total Quality Management:

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

Организации зависят от своих потребителей и, таким образом, должны понимать текущие и будущие потребности потребителей, удовлетворять их требования и стремиться превзойти их ожидания [3].

Применение принципа ориентации организации на потребителя приводит к следующим действиям:

- понимание всего спектра потребностей и ожиданий потребителей относительно продуктов, поставок, цен, надежности и т. д.;
- обеспечение взвешенного подхода к потребностям и ожиданиям потребителей и других участвующих сторон (владельцев, персонала, поставщиков, местных сообществ и общества в целом);
- распространение информации об этих потребностях и ожиданиях во всей организации;
- измерение степени удовлетворенности потребителей и влияние на результат;
- управление взаимоотношениями с потребителями.
- Полезные применения данного принципа включают:
- для формулировки политики и стратегии (policy and strategy formulation) обеспечение того, что потребности потребителей и других участвующих сторон осознаются всей организацией;
- для установления целей и плановых показателей (goal and target setting) обеспечение того, что соответствующие цели и плановые показатели непосредственно связаны с потребностями и ожиданиями потребителей;
- для управления операциями (operational management) повышение производительности организации для удовлетворения потребностей потребителей;
- для управления людскими ресурсами (human resource management) обеспечение того, что персонал обладает знаниями и опытом, требующимися для удовлетворения потребителей организации.

2. Лидерство (Leadership)

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

Применение принципа лидерства приводит к следующим действиям:

- действенность и личный пример;
- понимание изменений во внешней окружающей среде и реагирование на них;
- внимание к потребностям всех участвующих сторон, включая потребителей, владельцев, персонала, поставщиков, местные сообщества и общество в целом;
- выработка ясного видения будущего организации;
- выработка общих ценностей и этики на всех уровнях организации;
- установление доверия и искоренение страха;
- обеспечение персонала требуемыми ресурсами и свободой, необходимыми для того, чтобы действовать ответственно и обоснованно;
- воодушевление, поощрение и признание вклада персонала;
- содействие открытому и честному общению;
- обучение, подготовка и инструктирование персонала;
- выработка достойных целей и плановых показателей;
- реализация стратегии по достижению этих целей и плановых показателей.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии выработка и распространение ясного видения будущего организации;

- для установления целей и плановых показателей преобразование видения организации в измеримые цели и плановые показатели;
- для управления операциями стратегические цели организации достигаются полномочным и вовлеченным персоналом;
- для управления людскими ресурсами наличие полномочной, мотивированной, хорошо информированный и стабильной рабочей силы.

3. Вовлечение персонала (Involvement of People).

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

Применение принципа вовлечения персонала приводит к следующим действиям персонала:

- принятие на себя задач и ответственности за их решение;
- активный поиск возможностей усовершенствования;
- активный поиск возможностей повышения собственных квалификации, знаний и опыта;
- свободный обмен знаниями и опытом в группах и коллективах;
- концентрация на создании ценностей для потребителя;
- новаторство и творчество в дальнейшем продвижении стратегических целей организации;
- олицетворение организации перед лицом потребителей, местных сообществ и общества в целом;
- получение удовольствия от своей работы;
- гордость и удовлетворение быть частью организации.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии персонал, эффективно участвующий в усовершенствовании политики и стратегии организации;

- для установления целей и плановых показателей персонал, принимающий на себя задачи и разделяющий ответственность за их решение;
- для управления операциями персонал, вовлеченный в соответствующие усовершенствования решений и процессов;
- для управления людскими ресурсами персонал, в большей степени удовлетворенный своей работой и активно вовлеченный в собственный рост и развитие на благо организации.

4. Процессный подход (Process Approach).

Применение принципа процессного подхода приводит к следующим действиям персонала:

- определение процесса достижения желаемого результата;
- выявление и измерение входов и выходов процесса;
- выявление интерфейсов процесса с функциями организации;
- оценка возможного риска, его последствий и влияния процесса на потребителей, поставщиков и другие участвующие в процессе стороны;
- четкое распределение ответственности, полномочий и подотчетности при управлении процессом;
- выявление внутренних и внешних потребителей, поставщиков и других участвующих в процессе сторон;
- при проектировании процессов уделяется внимание шагам процессов, видам деятельности, потокам, контрольным величинам, потребностям в подготовке персонала, оборудовании, методах, информации, материалах и других ресурсах, необходимых для достижения желаемого результата.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии использование определенных процессов во всей организации приведет к более предсказуемым

результатам, лучшему использованию ресурсов, сокращению времени цикла и снижению затрат;

- для установления целей и плановых показателей понимание зрелости процессов способствует выработки достойных целей и плановых показателей;
- для управления операциями принятие процессного подхода ко всем операциям приводит к снижению затрат, предотвращению ошибок, контролю за отклонениями, сокращению времени цикла и более предсказуемым выходам;
- для управления людскими ресурсами установление эффективных по затратам процессов для управления людскими ресурсами, таких, как найм, образование и подготовка, способствует приведению этих процессов в соответствие потребностям организации и создает более зрелую рабочую силу.

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

Применение принципа системного подхода к административному управлению приводит к следующим действиям:

- определение системы путем выявления или разработки процессов, влияющих на достижение заданной стратегической цели;
- структурирование системы так, чтобы достичь заданную стратегическую цель наиболее эффективным способом;
- понимание взаимозависимостей между процессами системы;
- непрерывное усовершенствование системы посредством измерения и оценки;
- предварительное установление ограничений по ресурсам.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии, создание всеобъемлющих и достойных планов, связывающих входы функций и процессов;

- для установления целей и плановых показателей, цели и плановые показатели конкретных процессов приведены в соответствие с ключевыми стратегическими целями организации;
- для управления операциями, более широкий взгляд на эффективность процессов, что приводит к пониманию причин проблем и своевременным действиям по усовершенствованию;
- для управления людскими ресурсами, дает лучшее понимание распределения ролей и ответственности при достижении общих стратегических целей, уменьшая таким образом межфункциональные барьеры и улучшая коллективную работу.

6. Непрерывное усовершенствование (Continual Improvement)

Применение принципа непрерывного усовершенствования приводит к следующим действиям:

- превращение непрерывного усовершенствования продуктов, процессов и систем в стратегическую цель каждого сотрудника организации;
- применение базовых понятий последовательного (инкрементного) и скачкообразного усовершенствования;
- проведение периодических аттестаций степени достижения установленных критериев высшего качества для выявления областей потенциального усовершенствования;
- непрерывное повышение эффективности и результативности всех процессов;
- поощрение действий, основанных на предотвращении проблем;
- предоставление каждому члену организации необходимых образования и подготовки по методам и инструментальным средствам непрерывного усовершенствования, таким, как:
- цикл Планирование – Исполнение – Проверка – Корректирующие действия (Plan – Do – Check – Act);

- решение проблем;
- реинженеринг процессов;
- нововведение в процессах.
- установление мер и целей для направления и отслеживания усовершенствований;
- признание усовершенствований.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии создание и осуществление более конкурентоспособных бизнес-планов путем объединения непрерывного усовершенствования со стратегическим и бизнес-планированием;
- для установления целей и плановых показателей установление реалистичных и достойных целей усовершенствования и предоставление ресурсов для их достижения;
- для управления операциями вовлечение персонала организации в непрерывное усовершенствование процессов;
- для управления людскими ресурсами обеспечение всего персонала организации инструментальными средствами, возможностями и мотивацией для совершенствования продуктов, процессов и систем.

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

Применение принципа основанного на фактах подхода к принятию решений приводит к следующим действиям:

- осуществление измерений и сбор данных и информации, относящихся к стратегической цели;
- обеспечение существенной точности, надежности и доступности данных и информации;
- анализ данных и информации с применением обоснованных методов;

- понимание значения соответствующих статистических методик;
- принятие решений и осуществление действий на базе логического анализа, уравновешенного опытом и интуицией.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии, стратегии, основанные на соответствующих данных и информации более реалистичны и достижимы с большей вероятностью;
- для установления целей и плановых показателей, применение соответствующих сравнительных данных и информации для установления реалистичных и достойных целей и плановых показателей;
- для управления операциями, данные и информация являются базисом для понимания производительности процессов и систем для направления усовершенствований и предотвращения будущих проблем;
- для управления людскими ресурсами, анализ данных и информации из таких источников, как опросы персонала, предложения сотрудников, целевых групп для направления формулировки политики управления людскими ресурсами.

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

Применение принципа взаимовыгодных отношений с поставщиками приводит к следующим действиям:

- выявление и выбор ключевых поставщиков;
- установление с поставщиками отношений, которые бы уравновешивали краткосрочные выгоды и долгосрочные соображения для организации и общества в целом;
- создание ясного и открытого общения;

- инициация совместных разработок и усовершенствования продуктов и процессов;
- совместная установление ясного понимания потребностей потребителей;
- обмен информацией и будущими планами;
- признание усовершенствований и достижений поставщиков.

Полезные применения данного принципа включают:

- для формулировки политики и стратегии, создание конкурентоспособных преимуществ путем создания стратегических союзов или партнерских отношений с поставщиками;
- для установления целей и плановых показателей, установление более достойных целей и плановых показателей с помощью вовлечения и участия поставщиков на ранних этапах;
- для управления операциями, создание отношений с поставщиками и управление этими отношениями для обеспечения надежных, своевременных и свободных от дефектов поставок;
- для управления людскими ресурсами, выработка и повышение зрелости поставщиков посредством проведения подготовки поставщиков и совместных усилий по усовершенствованию.

6.3. ISO9000. Структура документов СК

Система управления качеством поддерживается следующей структурой документов:

- Заявление о политике и целях в области качества.
- Руководство по качеству.
- Документированные процедуры, требуемые настоящим международным стандартом.
- Документы, необходимые организации для:
  - обеспечения эффективного планирования;

  - осуществления процессов и управления ими (положения о подразделениях, должностные инструкции, регламенты, технологические инструкции).
- Записи о качестве.

ISO9000. Заявление о политике и целях в области качества

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

- Общие намерения и направления деятельности организации в области качества, официально сформулированные высшим руководством. Как правило, цели относятся к таким областям как:
  - новая техника / технология;
  - совершенствование продукции /новая продукция;
  - удовлетворенность потребителей / завоевание рынков;
  - социальные вопросы / удовлетворенность персонала;
  - поставщики / снижение отходов / экономия ресурсов;
- Не должна быть просто декларацией – цели должны быть конкретными, достижимыми, проверяемыми.
- Определять приоритеты, поставить 3-4 цели. Не надо все перечисленные направления включать в «Политику». Высшее руководство должно определить приоритеты, поставить 3-4 цели. Это не значит, что по остальным направлениям ничего не будет делаться, просто заявленные в «Политике» цели будут первоочередными, наиболее важными.
- Добиваться поставленных целей. При этом высшее руководство не только должно подписать «Политику», что является обязательным, но действительно добиваться этих целей.

ISO9000. Руководство по качеству

Руководство по качеству – более детальны документ (20-50 листов), содержащий описание всей системы качества в целом. Структура этого документа

должна соответствовать структуре стандарта ISO-9001 или должна быть представлена таблица соответствия разделов руководства и стандарта. Структура стандарта ISO-9001 включает:

- Система менеджмента качества.

  Общие требования. Требования к документации.

- Ответственность руководства.

Обязательства руководства. Ориентация на потребителя. Политика в области качества. Планирование. Ответственность, полномочия и обмен информацией. Анализ со стороны руководства.

- Менеджмент ресурсов.

Обеспечение ресурсами. Человеческие ресурсы. Инфраструктура. Производственная среда.

- Процессы жизненного цикла продукции

Планирование процессов жизненного цикла продукции. Процессы, связанные с потребителями. Проектирование и разработка. Закупки. Производство и обслуживание. Управление устройствами для мониторинга и измерений.

- Измерение, анализ и улучшение.

Общие положения. Мониторинг и измерение. Управление несоответствующей продукцией. Анализ данных. Улучшение.

ISO9000. Документированные процедуры

При внедрении систем качества значительная роль отводится документации. Значимость документации проявляется в нескольких критических случаях, таких как:

- достижение требуемого уровня качества продукта/услуги и непрерывного улучшения качества;
- обеспечение повторяемости процессов, протекающих в организации;
- осуществление требуемого обучения персонала;
- оценка эффективности системы;
- проведение аудита и сертификации системы качества.

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

- Обязательными для документирования:

- процедура по управлению документацией;процедура по управлению записями о качестве;
- процедура по проведению внутренних проверок;
- процедура по управлению несоответствующей продукцией;
- процедура по корректирующим действиям;
- процедура по предупреждающим действиям.

6.1.1.1. ISO9000. Записи о качестве

Состав записей учитывает специфику предприятия, сложившуюся практику, т.е. предприятие само определяет в каком виде вести и хранить эти записи. Ниже приведены некоторые (но не все) возможные варианты для всех типов записей, требуемых ISO-9000:

- Анализ со стороны руководства.
  - акты анализа, протоколы "дня качества", протоколы совещания ….
- Образование, подготовка, навыки и опыт персонала.
  - личные дела сотрудников, журналы, карточки в отделе кадров …
- Соответствие процессов требованиям.
  - протокол испытаний, журнал процесса, акт испытаний….
- Результаты оценивания поставщиков.
  - Реестр надежных поставщиков.
- Валидация процессов обеспечения производства.
  - Протокол проверки спец.процессов, акт исследования, журнал пооперационного контроля.
- Анализ требований к продукции.
  - Протокол о намерениях.

- Входные данные проектирования и разработки.
    - ТЗ на разработку
- Анализ проекта и разработки
    - Заключение по проекту.
- Согласование (рассмотрение) проекта и разработки.
    - Акт приемки проекта, отзыв рецензента.
- Утверждение проекта и разработки.
    - Утверждающая подпись на акте.
- Изменения проекта и разработки.
    - Извещение об изменении.
- Идентификация продукции.
    - Бирки, шильдики, наклейки.
- Собственность потребителя утеряна, повреждена или признана непригодной для использования.
    - Извещение о браке, дефектная ведомость, акт.
- Результаты калибровки и поверки контрольных и измерительных приборов.
    - Свидетельство о калибровке, график поверки, паспорт на прибор.
- Результаты внутренних проверок.
    - График внутренних проверок, отчеты и акты внутренних проверок.
- Соответствие продукции критериям приемки; лицо, санкционировавшее выпуск продукции.
    - Накладные приемки, акт приемки ОТК, сертификат соответствия.
- Характер несоответствий, предпринятые действия.
    - Классификатор дефектов, карточки разрешений на отклонения, акт списания в брак.
- Результаты предпринятых корректирующих действий.
    - Отметка в контрольной карточке, отметка в плане корректирующих действий.

- Результаты предпринятых предупреждающих действий.
    - Отметка в контрольной карточке, отметка в плане корректирующих действий, протокол совещания.

6.2.3. ISO9000. Как работает система управления качеством

Схема работы системы управления качеством представлена на слайде:

- Объектом управления является «производственная линейка»: Требования заказчика – Создание продукции (оказание услуги) – Выпуск продукции (оказание услуги)
- Обеспечение качества составляет элемент общей стратегии организации, опирается на требуемые для этого ресурсы и адекватное управление, организованными в соответствии со стандартом ISO9001.2000.
- В соответствии с этим стандартом, адекватное управление включает:
    - Выпускаемая продукция контролируется на удовлетворение требований заказчика.
    - Этот контроль проводится путем измерений количественных показателей, на основе чего проводится анализ и даются рекомендации по улучшению процессов производства.
    - Для выполнения этих рекомендаций подключается руководство, которое достигает результата путем управления соответствующими ресурсами.
- Все это вместе способствует повышению имиджа организации и сертификации на соответствие стандарту.

Главным в этой схеме является то, что она работает постоянно и непрерывно. Действует принцип CPI: Continuous Process Improvement – Постоянное Улучшение Процессов.

6.2.4. ISO9000. Немного истории

1979 г. BS-5750 (British Standards Institution, BSI) Первые стандарты на системы качества, носившие название BS-5750, были разработаны Британским институтом стандартов (British Standards Institution, BSI) и утверждены в 1979 году. Были взяты за основу ISO 9000.

В марте 1987 г. была принята первая версия серии стандартов ISO 9000, разработанная на основе BS-5750. Как и все стандарты ISO имеет рекомендательный характер, однако более чем в 90 странах мира они приняты в качестве национальных стандартов. В России некоторые стандарты ISO утверждены в настоящее время в качестве государственных стандартов (ГОСТ).

В 1994 г. была принята вторая редакция (версия) стандартов. Эта редакция включала 24 стандарта (номера с 9000 и 10000). Такое большое количество стандартов версии 1994 года объясняется тем, что они создавались независимо от специфики промышленности, но при практическом применении потребовалась разработка рекомендаций, уточняющих их применение в деятельности, связанной с перспективным управлением, непрерывным улучшением, проверками, подготовкой и обучением персонала и т.д.

Декабрь 2000 г. – третья редакция: ISO 9000:2000, содержащая 5 базовых стандартов.

ISO 9000. Версия 1994 г.

Стандарты ISO серии 9000 версии 1994 могут быть условно разделены на три отдельные группы:

- Базовые стандарты;
- Стандарты поддержки;
- Методические руководства.

ISO9000.94. Базовые стандарты

Данная группа включает 4 стандарта ISO (9001, 9002, 9003, 9004).

- **ISO 9001:1994 –** Системы качества. Модель для обеспечения качества при проектировании, разработке, производстве, монтаже и обслуживании.
- **ISO 9002:1994 –** Системы качества. Модель для обеспечения качества при производстве, монтаже и обслуживании.
- **ISO 9003:1994 –** Системы качества. Модель для обеспечения качества при контроле и испытаниях готовой продукции.
- **ISO 9004:1993 –** Общее руководство качеством и элементы системы качества.

ISO9000.94. Стандарты поддержки

Данная группа содержит стандарты, предназначенные для оказания помощи:

- **ISO 10011-1:1990 –** планирование, подготовка и проверка системы качества
- **ISO 10011-2:1991 –** подбор и обучение экспертов для проверок системы качества
- **ISO 10011-3:1991 –** подготовка и руководство программой проверок системы качества
- **ISO 8402:1994 -** определение терминов, наиболее часто встречающихся в стандартах и технических условиях
- **ISO 9000-1:1994 -** определение областей применения стандартов
- **ISO 10012-1:1992 –** определение базовых характеристик системы метрологического обеспечения качества, необходимых для измерительной системы поставщика

ISO9000.94. Методические руководства

Данная группа содержит методические рекомендации, представляющие собой документы по оказанию помощи:

- **ISO 9000-2:1993, ISO 9000-3:1991; ISO 9000-4:1993** – в практическом применении ISO 9001, 9002 и 9003.
- **ISO 9004-2:1991 –** внедрения системы качества в сфере услуг.
- **ISO 9004-3:1993 –** управления качеством перерабатываемых материалов.

- **ISO 9004-4:1993 –** непрерывного улучшения качества внутри организации.
- **ISO 10013 –** в подготовке Руководств по качеству.
- **ISO 1005 –** в подготовке и применении планов по качеству.
- **ISO 1006 –** в подготовке и применении обеспечения качества в перспективном управлении.
- **ISO 1007 –** в подготовке и применении конфигурации управления;
- **ISO 10014 –** персональной ответственности за изучение потребностей потребителя и последующее их удовлетворение.
- **ISO 10015 –** планов непрерывного обучения и подготовки персонала.

ISO 9000. Версия 2000г.

В результате пересмотра комплекс стандартов ISO 9000:2000 теперь содержит 5 базовых стандартов:

- **ISO 9000:2000**. Система менеджмента качества. Основные принципы и словарь.
- **ISO 9001:2000**. Система менеджмента качества. Требования.
    - устанавливает минимальный набор требований к системам качества и применяется для целей сертификации.
- **ISO 9004:2000**. Система менеджмента качества. Руководящие указания по улучшению качества.
    - содержит методические указания по созданию систем менеджмента качества, которые ориентированы на высокую эффективность деятельности.
- **ISO 19011:2000**. Руководящие указания по проверке системы менеджмента качества и охраны окружающей среды.
- **ISO 10012**. Обеспечение качества измерительного оборудования.

Позже стали выходить различные методические руководства по применению ISO9000:2000 к различным видам деятельности. В частности:

- **ISO/IEC 90003**, Guidelines for the Application of ISO 9001:2000 to Computer Software, 2004.

6.3. ISO12207: процессы качества ПО

Как отмечалось, серия стандартов ISO9000 имеет универсальный характер, т.е. применима для любого вида деятельности, т.е. формально и для разработки программных продуктов. Универсальный характер стандартов этой серии обычно вызывает проблемы при их применении в той или иной области. Поэтому серия ISO9000 включает достаточно большое количество методических руководств по применению стандартов в том или ином виде деятельности. Разработка программных продуктов является достаточно специфической областью деятельности, но для нее никаких методических указаний в серии нет. Вызвано это, видимо, тем, что в программной инженерии есть свои, специфические для этой области стандарты качества. Все эти стандарты не противоречат ISO9000, а дополняют и конкретизируют эту серию для программной инженерии.

Одним из таких стандартов является ISO12207 – Процессы жизненного цикла программного обеспечения. В этом стандарте описаны два процесса, относящиеся к управлению качеством ПО:

- Процесс обеспечения качества
- Процесс верификации
- Процесс аттестации
- Процесс усовершенствования

Качество ПО было предметом стандартизации и нашей стране. Был создан стандарт ГОСТ 2844–94, в котором дано определение качества ПО, как совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика, в соответствии с назначением. Этот стандарт регламентирует базовую модель качества и его показатели, главным среди них является надежность.

6.3.1. ISO12207. Процесс обеспечения качества

Цель процесса – обеспечение гарантий того, что программные продукты и процессы в жизненном цикле проекта соответствуют установленным требованиям и утвержденным планам. При обеспечении качества могут использоваться результаты других вспомогательных процессов, таких как верификация, аттестация, совместные анализы, аудит и решение проблем.

Процесс состоит из следующих работ:

- Подготовка процесса;
- Обеспечение продукта;
- Обеспечение процесса;
- Обеспечение систем качества.

Подготовка процесса обеспечения качества

Подготовка процесса среди прочих включает решение следующих основных задач:

1. Адаптация процесса обеспечения качества к условиям конкретного проекта. Цели процесса обеспечения качества должны быть определены так, чтобы гарантировать, что программные продукты и процессы, используемые при создании данных программных продуктов, соответствуют установленным требованиям и утвержденным планам.
2. Координация процесса с процессами верификации, аттестации, совместного анализа и аудита.
3. Разработка плана выполнения работ и задач процесса обеспечения качества. План должен устанавливать:
    - Применяемые стандарты качества
    - Процедуры проведения анализов качества
    - Процедуры сбора, регистрации, сопровождения и распространения информации о качестве
    - Ресурсы, графики и обязанности при проведении работ по обеспечению качества

– Выбранные работы и задачи из вспомогательных процессов (верификация, аттестация, совместный анализ, аудит и решение проблем).

4. Обеспечение доступности заказчику отчетов по обеспечению качества.
5. Обеспечение лиц, ответственных за качество организационной независимостью, ресурсами и полномочиями.

Обеспечение продукта

Данная работа состоит из следующих задач:

1. Обеспечение документального оформления, взаимного согласования и выполнения всех планов проекта.
2. Обеспечение разработки программных продуктов и документации по условиям договора и в рамках утвержденных планов.
3. Обеспечение соответствия программных продуктов требованиям и пожеланиям заказчика при их подготовке к поставке.

Обеспечение процесса

Обеспечение процесса состоит из следующих задач:

1. Процессы жизненного цикла, связанные с реализацией проекта (поставка, разработка, эксплуатация, сопровождение и т.д.), должны выполняться в соответствии с условиями договора и в рамках утвержденных планов.
2. Используемые технологии программирования, условия разработки, … должны соответствовать условиям договора.
3. Требования основного договора должны быть доведены до субподрядчика, а разработанные им программные продукты удовлетворяли этим требованиям.
4. Заказчик и другие участники должны обеспечивать взаимную поддержку и кооперацию
5. Характеристики программного продукта и процессов должны соответствовать установленным стандартам и процедурам.

6. Персонал, участвующий в реализации проекта, должен обладать достаточным опытом и знаниями.

Обеспечение систем качества

Данная работа состоит из одной задачи:

1. Должно быть обеспечено проведение дополнительных работ по управлению качеством в соответствии с разделами ГОСТ Р ИСО 9001, указанными в договоре.

6.3.2. ISO12207. Процесс верификации

Цель процесса верификации – определение того, что программные продукты функционируют в полном соответствии с требованиями. Процесс может включать анализ, проверку и испытание (тестирование). Процесс может выполняться с различной степенью независимости исполнителей процесса от разработчиков программного продукта. Независимая верификация выполняется независимой от разработчика организацией.

Процесс верификации состоит из следующих работ:

- Подготовка процесса;
- Верификация.

Подготовка процесса верификации

Основными задачами подготовки процесса верификации являются:

1. Определение необходимости верификации и степени организационной независимости исполнителей. Анализ критичности проектных требований с точки зрения необходимости верификации.
2. Установление процесса верификации. Выбор (при необходимости) независимой организации.
3. Определение работ и программных продуктов, нуждающиеся в верификации
4. Разработка плана верификации на основе установленных задач верификации

5. Выполнение плана верификации. Устранение обнаруженных проблем через процесс решения проблем.

Верификация

Основными задачами верификации являются:

1. Верификация договора по критериям:
   - возможности удовлетворить установленным требованиям;
   - непротиворечивости и полноты требований;
   - наличия процедур внесения изменений в требования и решения проблем;
   - наличия процедур по взаимодействию и кооперации между участниками;
2. Верификация процесса по критериям:
   - соответствие и своевременность установления проектных требований;
   - пригодность, реализуемость и выполнимость выбранных для проекта процессов;
   - применимость выбранных стандартов проектирования;
   - укомплектованность и обученность персонала.
3. Верификация требований по критериям:
   - непротиворечивость, выполнимость, тестируемость и точность;
   - распределение требований к аппаратным, программным и ручным операциям;
   - правильность требований по безопасности, защите и критичности.
4. Верификация проекта по критериям:
   - соответствие и учет требований в проекте;
   - реализуемость проекта по времени, требованиям, ресурсам

- возможность выбора проекта, исходя из установленных требований;
- правильность реализации в проекте требований безопасности, защиты и других критических требований.

5. Верификация программы по критериям:
   - тестируемость, правильность и соответствие установленным требованиям и стандартам программирования;
   - реализуемость событий и интерфейсов, обнаружения, локализации и восстановления ошибок;
   - возможность выбора программы, исходя из проекта или установленных требований;
   - правильность реализации в программе требований безопасности, защиты и других критических требований.
6. Верификация сборки по критериям:
   - полнота и правильность сборки компонентов и модулей;
   - полнота и правильность сборки технических и программных объектов и ручных операций в систему.
7. Верификация документации по критериям:
   - соответствие, полнота и непротиворечивость документации;
   - своевременность подготовки документации;
   - управление конфигурацией документов.

6.3.3. ISO12207. Процесс аттестации

Цель процесса аттестации – определение полноты установленных требований, созданного программного продукта их функциональному назначению. Процесс может выполняться с различными степенями независимости исполнителей. Независимой называют аттестацию, если организация-исполнитель не зависит от поставщика, разработчика, оператора или персонала сопровождения.

Процесс состоит из следующих работ:

- Подготовка процесса
- Аттестация.

Подготовка процесса аттестации

Основными задачами подготовки процесса аттестации являются:

1. Определение необходимости аттестации и степень организационной независимости исполнителей.
2. Определение задач аттестации и установление процесса аттестации.
3. Разработка плана аттестации, определяющего объекты, задачи, ресурсы и процедуры аттестации.
4. Выполнение плана аттестации. Устранение обнаруженных проблем через процесс решения проблем.

Аттестация

Основными задачами аттестации являются:

1. Подготовка требований к тестированию, контрольных примеров и технических условий испытаний.
2. Обеспечение соответствия требований, контрольных примеров и технических условий испытаний конкретным требованиям и объектам.
3. Проведение испытаний, включая:
    - испытания при критических, граничных и особых значениях исходных данных;
    - испытание на ошибкоустойчивость;
    - испытание при участии репрезентативно выбранных пользователей.

6.3.4. ISO12207. Процесс усовершенствования

Процесс усовершенствования является процессом установления, оценки, измерения, контроля и улучшения любого процесса жизненного цикла программных средств.

Процесс состоит из следующих работ:

- Создание процесса;
- Оценка процесса;
- Усовершенствование процесса.

Создание процесса усовершенствования

Работа состоит из одной задачи:

Определить набор организационных процессов для всех процессов жизненного цикла в соответствии с имеющимся практическим опытом.

При этом:

- Эти организационные процессы и их применение в конкретных ситуациях должны быть задокументированы;
- Должен быть определен механизм управления процессом усовершенствования при разработке, контроле, управлении и улучшении улучшаемых процессов.

Оценка процесса усовершенствования

Оценка процесса (улучшаемого) состоит из следующих задач:

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

Усовершенствование процесса

Усовершенствование (выполняемых) процессов включает:

3. По результатам анализа и оценки внести соответствующие улучшения в выполняемый процесс, при этом должны быть внесены соответствующие изменения в документацию выполняемого процесса.
4. Для выявления сильных и слабых сторон выполняемых процессов должны быть собраны и проанализированы архивные, технические и оценочные данные. Результаты анализов должны быть использованы для

усовершенствования данных процессов, выработки рекомендаций по внесению изменений в реализуемые или планируемые проекты и определения потребности в передовых технологиях.

5. Для усовершенствования организационных процессов административной деятельности должны быть собраны, обновлены и использованы данные о расходах. Эти данные должны быть использованы при определении стоимости работ по предотвращению и решению обнаруженных проблем и несоответствий в программных продуктах и услугах.

6.3.5. ISO12207. Некоторые выводы

В части управления качеством стандарт ISO12207 явно следует принципам TQM:

- Процессный подход, как основа стандарта
- Системной подход к управлению
- Ориентация на потребителя
- Непрерывное усовершенствование (процесс усовершенствования)

Стандарт ISO12207 также соответствует (и явно ссылается в задаче построения системы управления качеством) стандарту ISO9000.

Недостатками стандарта ISO12207 являются:

- Дается некоторая детализация построения системы управления качеством для разработки ПО (процессы аттестации и верификации), но в целом он просто ссылается на ISO9000. Это не очень удивительно: ISO12207 был принят в 1995 году сразу вслед за второй версией ISO9000.94.
- Организация процессов управления качеством дается в самом общем виде и не всегда ясно, как их применять на практике.
- Непонятно, в чем разница между верификацией и аттестацией.

Причины недостатков ISO12207 связаны с тем, что этот стандарт имеет не «сертификационный», а рекомендательный характер.

6.4. CMM: зрелость организаций и процессов

CMM SW – Capability Maturity Model for Software – американский стандарт в области качества ПО, разработанный SEI по заказу министерства обороны США. Этот стандарт появился в 1993 году и быстро получил широкое международное признание. Главным образом потому, что:

- Этот стандарт предназначен только для разработки ПО;
- По отношению к остальным стандартам, управление качеством для разработки ПО в нем прописаны достаточно подробно и детально.

В этом разделе мы рассмотрим следующие вопросы:

- Причины и история создания стандарта CMM;
- Модель технологической зрелости CMM;
- Пять уровней зрелости.
- Как работает стандарт CMM.

6.4.1. CMM. Причины и история создания

Как отмечалось, появившийся в 1987г. стандарт ISO 9000 универсален. Его основными недостатками являлись:

- Недостаточная подробность, порождающая возможность самых различных его толкований в зависимости от представлений аудитора
- Неточность оценки качества процессов, задействованных при создании и внедрении программного обеспечения
- Отсутствие механизмов улучшения процессов.

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

- Рост сложности задач, вызванный развитием аппаратной базы. Появление и широкое внедрение интегральных схем существенно повысило производительность вычислительной техники, что позволило переходить к решению качественно более сложных задач.

взрывоподобным Рост объема и сложности задач, возлагаемых на программное обеспечение, имел взрывоподобный характер;

- Хронические срывы сроков и качества, как следствие роста сложности задач. Сроки выполнения проектов постоянно срывались, качество ПО (соответствие ожиданиям заказчика) оставалось на неприемлемо низком уровне, и Министерство обороны США начало всерьез беспокоиться об эффективности расходования бюджетных средств;
- Безуспешный поиск методик и инструментов. Усилия были направлены на поиск эффективных методологий и инструментов для разрешения «сугубо технических» (как тогда казалось) проблем программного обеспечения. Почти два десятилетия обещаний поднять производительность и качество работ за счет новых методов и средств разработки ПО ушло на осознание того, что корень зла – не в технике;
- Неспособность организаций управлять процессом разработки ПО как основная причина сложившейся ситуации. В конце концов, был сделан вывод, что фундаментальная проблема «хронического кризиса ПО» состоит в неспособности организаций управлять технологическим процессом разработки программного обеспечения;
- Поиск методов оценки способности организаций. И тогда военные приступили к поиску формальных и объективных методов оценки способности организации-разработчика произвести ПО требуемой сложности в установленные сроки и с требуемым уровнем качества. SEI (Software Engineering Institute) получило заказ от министерства обороны США на проведение исследований в этой области.

В 1993 году выходит отчет SEI: CMM SW – Capability Maturity Model for Software – Модель технологической зрелости организации-разработчика ПО.

6.4.2. CMM. Модель технологической зрелости

Зрелые и незрелые организации

Исследователи SEI пошли достаточно простым путем. Следуя TQM, оценку зрелости организаций они решили проводить на основе анализа выполняемых этой организацией процессов по разработке ПО. При этом считалось (а это – в духе ISO9000), что организация является тем более зрелой (более предсказуемой), чем более установленными являются применяемые процессы.

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

Модель технологической зрелости

В CMM дается следующее определение: Модель технологической зрелости – это описание стадий эволюции, которые проходят организации-разработчики по мере того, как они (организации) определяют, реализуют, измеряют, контролируют и совершенствуют процессы создания ПО. Модель зрелости процессов разработки ПО предоставляет организации-разработчику руководящие принципы управления своими процессами разработки и сопровождения ПО, а также развития культуры управления и программной инженерии. CMM предназначена помогать организациям в выборе стратегий усовершенствования процессов путем определения текущего уровня зрелости производственного процесса и выявления некоторых вопросов, наиболее значимых для повышения качества создаваемого ПО и усовершенствования процессов. [Paulk, 1.3]

Основу модели CMM составляют следующие фундаментальные понятия:

- Process (технология, технологический процесс, процесс) – последовательность шагов (действий), предпринимаемых с заданной целью. Более точно, процесс определяется так: Производственный процесс – набор операций, методов, практик и преобразований, используемых разработчиками для создания и сопровождения ПО и связанных с ним

продуктов (например, планов проекта, проектных документов, кодов, сценариев тестирования и руководств пользователя).

- Process Capability (продуктивность, совершенство технологии/процесса) – диапазон результатов, которые можно ожидать от организации, соблюдающей данный технологический процесс. Это понятие имеет отношение к будущим проектам, но базируется на фактических характеристиках технологии, достигнутых на предыдущих проектах.
- Process Performance (производительность технологии/процесса) – фактические результаты, достигнутые организацией, соблюдающей данную технологию/процесс. Это понятие ассоциируется с уже выполненными проектами.

  Таким образом, производительность фокусируется на достигаемых результатах, в то время как его продуктивность опирается на ожидаемые результаты.
- Process Maturity (зрелость технологии) – степень определенности, управляемости, наблюдаемости, контролируемости и эффективности процесса, технологии. Фактически, это индикатор полноты технологии и степени последовательности (настойчивости) организации в ее применении на всех проектах. Зрелость определяет потенциал дальнейшего роста совершенства технологии/процесса.

Модель зрелости процессов разработки ПО предоставляет организации-разработчику руководящие принципы управления своими процессами разработки и сопровождения ПО, а также развития культуры управления и программной инженерии. CMM предназначена помогать организациям в выборе стратегий усовершенствования процессов путем определения текущего уровня зрелости производственного процесса и выявления некоторых вопросов, наиболее значимых для повышения качества создаваемого ПО и усовершенствования процессов. Концентрируя свое внимание на конкретном перечне работ и активно добиваясь их выполнения, организация может планомерно совершенствовать свой

производственный процесс, обеспечивая устойчивый и постоянный рост его продуктивности.

6.4.3. CMM. Пять уровней зрелости

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

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

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

6.4.4. CMM. Определение модели зрелости

CMM. Структура модели зрелости

Модель зрелости должна дать ответ на два вопроса:

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

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

Структура (схема) модели CMM содержит следующие основные элементы:

- **Группы ключевых процессов**. Каждый уровень зрелости содержит описание группы ключевых процессов, которые должны выполняться на этом уровне.
- **Цели**. Для каждого ключевого процесса определены цели, которые нужно достигнуть для перехода на следующий уровень. Цели (целевые установки):

- служат критерием эффективной реализации группы ключевых процессов в организации;
- выражают объем, границы и смысл каждой группы ключевых процессов;
- после того, как цели будут реализованы на постоянной основе для всех проектов, можно будет сказать, что организация установила уровень продуктивности своего производственного процесса, характеризующийся данной группой ключевых процессов.

Кроме того, определяется блок связанных работ, после выполнения которых достигаются цели, и круг проблем, которые необходимо решить для достижения следующего уровня зрелости.

- **Разделы**. Описания ключевых процессов организованы по разделам, которые представляют собой атрибуты, указывающие, являются ли эффективными, повторяемыми и устойчивыми реализация и установление групп ключевых процессов. Ниже перечислены пять основных разделов:

– *Обязательства по выполнению*. Описывают действия, которые должна выполнить организация, чтобы обеспечить установление и стабильность процесса. Обязательства по выполнению обычно касаются установления организационных политик и поддержки со стороны высшего руководства.

– *Необходимые предпосылки*. Описывают предварительные условия, которые должны выполняться в проекте или организации для компетентного внедрения производственного процесса, обычно касаются ресурсов, организационных структур и требуемого обучения.

– *Выполняемые операции*. В разделе «Выполняемые операции» описаны роли и процедуры, необходимые для внедрения группы ключевых процессов. Выполняемые операции обычно включают в себя создание планов и реализацию процедур, выполнение и отслеживание работ, а также, по мере необходимости, выполнение корректирующих действий.

– *Практики* раздела «Выполняемые операции» описывают, что должно быть реализовано для получения продуктивного процесса. Все остальные практики вместе формируют базис, с помощью которого организация может внедрить практики, описанные в разделе «Выполняемые операции».

– *Измерения и анализ*. Раздел «Измерения и анализ» описывает, что необходимо для измерения процесса и анализа результатов измерений. В этом разделе обычно приводятся примеры измерений, с помощью которых можно определить статус и эффективность выполняемых операций.

– *Проверка внедрения*. В разделе «Проверка внедрения» описываются шаги, позволяющие убедиться в том, что операции выполняются в соответствии с установленным процессом. В этот раздел обычно входят проверки и аудиты со стороны руководства и работы по обеспечению качества ПО.

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

CMM. Группы ключевых процессов

Уровни зрелости включают следующие группы ключевых процессов:

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

- Управление изменениями в технологиях.
- Управление изменениями в процессах.

6.4.5. CMM. Критерии оценки уровня зрелости

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

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

6.4.6. CMM. Резюме: CMM в тезисах

Схематично идею CMM можно представить в виде следующих тезисов:

1. Зрелость организации есть возможность выполнять сложные проекты.
2. Зрелость организации определяется через зрелость ее технологических процессов.
3. Можно выделить уровни зрелости организаций (процессов). В CMM их пять.
4. Модель зрелости – описание способа оценки уровня зрелости и путей перехода на следующий уровень.
5. Модель зрелости описывается:

- Ключевыми процессами, которые должны выполняться на каждом уровне зрелости.
- Каждый ключевой процесс описывается целями и набором разделов – атрибутов, определяющих различные аспекты выполнения процесса.
- Каждый атрибут описывается в виде ключевых практик – отдельных действий и условий, которые должны выполняться.

6.Достижение уровня зрелости определяется по критерию:

- уровень достигнут, если удовлетворены все ключевые процессы этого уровня.
- ключевой процесс удовлетворен, если достигнуты все его цели
- цели процесса достигнуты, если выполняются все ключевые практики всех разделов или их аналоги.

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

ВОПРОСЫ

1. Что такое качество?
2. Что такое мера качества?
3. Какова мера качества программного продукта?
4. Каковы основные фазы эволюции методов обеспечения качества?
5. Роль стандартов в обеспечении качества?
6. Что такое система управления качеством?
7. Что такое прогнозирование качества?
8. Что такое стандарты на программное обеспечение?
9. Каковы 8 принципов TQM?
10. Краткая характеристика этих принципов
11. Почему ISO9000 предписывает некоторую структуру документов?
12. Каков состав этой структуры документов?
13. Как работает система качества?

14. Применим ли ISO9000 к разработке ПО?

ЛИТЕРАТУРА

1. ГОСТ Р ИСО/МЭК 12207-99. ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ. Информационная технология. ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ.
2. Марк Паулк и др. Модель зрелости процессов разработки программного обеспечения – Capability Maturity Model for Software (CMM) Интерфейс-Пресс. 2002 г. · 256 с.
3. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 CMM) / Пер.с англ. А.С. Агапов, С.В. Зенин, Н.Э. Михайловский, А.А. Мкртумян А.А. – М.: Книга и бизнес, 2001. – 348 с.
4. Терехов А.А., Туньон В. Современные модели качества программного обеспечения (обзор ISO9000, CMM SPICE) [Электронный ресурс]. – Режим доступа: http://www.interface.ru/fset.asp?Url=/misc/qs.htm.
5. Назаренко Ю.А. Технологическая зрелость IT организаций. [Электронный ресурс]. – Режим доступа: http://www.noumen.ru /go/company/obj1041600305/obj1043060989.

7. УРОВНИ ЗРЕЛОСТИ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

7.1. История

Мысль о зависимости бизнеса от ИТ уже весьма не нова, и не требует дополнительной аргументации. С увеличением этой зависимости руководство бизнеса предъявляет все возрастающие требования к эффективности управления ИТ. Один из показателей эффективности управления – зрелость системы управления.

В ноябре 1986 года американский институт Software Engineering Institute (SEI) совместно с Mitre Corporation начали разработку обзора зрелости процессов разработки программного обеспечения, который был предназначен для помощи в улучшении их внутренних процессов.

Разработка такого обзора была вызвана запросом американского федерального правительства на предоставление метода оценки субподрядчиков для разработки ПО. Реальная же проблема состояла в неспособности управлять большими проектами. Во многих компаниях проекты выполнялись со значительным опозданием и с превышением запланированного бюджета. Необходимо было найти решение данной проблемы.

В сентябре 1987 года SEI выпустил краткий обзор процессов разработки ПО с описанием их уровней зрелости, а также опросник, предназначавшийся для выявления областей в компании, в которых были необходимы улучшения. Однако, большинство компаний рассматривало данный опросник в качестве готовой модели, вследствие чего через 4 года вопросник был преобразован в реальную модель, Capability Maturity Model for Software (CMM). Первая версия CMM (Version 1.0), вышедшая в 1991 году, в 1992 году была пересмотрена участниками рабочей встречи, в которой принимали участие около 200 специалистов в области ПО, и членами общества разработчиков.

В настоящее время существует множество разнообразных методологий построения процесса разработки ПО, и у каждого из них есть свои плюсы и минусы,

области эффективного применения. Все эти методологии преследуют своей первой целью улучшение производственного процесса, который позволил бы наиболее эффективно и качественно производить программные продукты. Кроме того, некоторые из них предоставляют методику оценки уже существующего технологического процесса, для того чтобы объективно сравнивать разные компании-разработчики по их уровню и производительности. Такие методики оценки используются компаниями-заказчиками для определения уровня исполнителей для своих проектов при принятии решения о заключении контракта.

Одной из наиболее популярных, востребованных и весомых методик на сегодняшний день является модель построения зрелых процессов разработки программного обеспечения SW-CMM (Capability Maturity Model for Software). До сих пор эта модель, разработанная Институтом программной инженерии при Университете Карнеги-Меллон (США), была почти неизвестна в России. Основной причиной этого было отсутствие материалов по этому стандарту на русском языке.

7.2. Фундаментальные концепции, лежащие в основе понятия зрелости производственных процессов

Согласно словарю Вебстера, процесс является «системой операций для производства чего-либо… последовательностью действий, изменений или функций, предназначенных для достижения окончания или результата». Комитет IEEE определяет процесс как «последовательность шагов, выполняемых для достижения заданной цели» [IEEE-STD-610].

В общем, **производственный процесс** может быть определен как набор операций, методов, практик и преобразований, используемых разработчиками для создания и сопровождения ПО и связанных с ним продуктов (например, планов проекта, проектных документов, кодов, сценариев тестирования и руководств пользователя). По мере того, как организация становится более зрелой, ее производственный процесс становится все более четко определенным и последовательно применяемым в рамках всей организации.

Продуктивность производственного процесса описывает совокупность ожидаемых результатов, которые могут быть достигнуты при следовании производственному процессу. Эта концепция позволяет организации-разработчику прогнозировать наиболее вероятные результаты будущего проекта разработки ПО.

Производительность производственного процесса представляет реальные результаты, достигаемые при соблюдении требований производственного процесса. Таким образом, производительность фокусируется на достигаемых результатах, в то время как его продуктивность опирается на ожидаемые результаты.

В зависимости от атрибутов конкретного проекта и его контекста, фактическая производительность выполнения проекта может не отражать полную продуктивность производственного процесса организации, т. е. потенциал проекта ограничивается его средой. Например, радикальные изменения в разрабатываемом приложении или в используемой технологии могут потребовать длительного обучения сотрудников, что снизит продуктивность и производительность выполнения данного проекта в сравнении с полной продуктивностью производственного процесса организации.

Уровень зрелости производственного процесса – это степень, до которой тот или иной процесс определен, управляем, измеряем, контролируем и эффективен. Зрелость подразумевает потенциал для роста продуктивности и отражает как полноту производственного процесса организации, так и постоянство, с которым организация применяет этот процесс во всех своих проектах. Производственный процесс достаточно хорошо понимается персоналом зрелой организации, обычно благодаря разработанной документации и проведенному обучению, и этот процесс постоянно контролируется и улучшается участвующими в нем сотрудниками. Продуктивность зрелого процесса разработки всегда хорошо известна. Зрелый производственный процесс подразумевает возможность постепенного улучшения качества своих результатов и производительности за счет стабильного повышения дисциплины своего выполнения.

По мере роста зрелости своего производственного процесса, организация-разработчик институционализирует производственные процессы с помощью политик, стандартов и организационных структур. Институционализация подразумевает создание инфраструктуры и корпоративной культуры, которые поддерживают методы, практики и бизнес-процедуры, сохраняя эти достижения после того, как разработавшие их сотрудники покинут организацию.

7.3. Обзор модели зрелости процессов разработки

Хотя зачастую инженеры-разработчики и менеджеры хорошо осведомлены о своих проблемах, их взгляды на то, какие усовершенствования являются наиболее важными, могут быть различными. Без организованной стратегии усовершенствования трудно достичь согласия между профессионалами-разработчиками и руководством по вопросу, какие именно работы по усовершенствованию следует выполнять первыми. Для того чтобы усилия по усовершенствованию процессов принесли долговременные результаты, необходимо разработать эволюционный путь развития, поэтапно увеличивающий зрелость производственного процесса организации. Концептуальная структура зрелости производственного процесса упорядочивает эти стадии таким образом, что усовершенствования на каждой предшествующей стадии являются фундаментом усовершенствований последующей стадии. Таким образом, стратегия усовершенствования, предлагаемая концептуальной структурой зрелости производственного процесса, обеспечивает наиболее прямой путь постоянного улучшения производственного процесса. Эта стратегия призвана руководить усовершенствованиями и выявлять недостатки организации; она не предназначена для быстрого «латания дыр» неудачного проекта.

Модель зрелости процессов разработки ПО предоставляет организации-разработчику руководящие принципы управления своими процессами разработки и сопровождения ПО, а также развития культуры управления и программной инженерии. CMM предназначена помогать организациям в выборе стратегий усовершенствования процессов путем определения текущего уровня зрелости

производственного процесса и выявления некоторых вопросов, наиболее значимых для повышения качества создаваемого ПО и усовершенствования процессов. Концентрируя свое внимание на конкретном перечне работ и активно добиваясь их выполнения, организация может планомерно совершенствовать свой производственный процесс, обеспечивая устойчивый и постоянный рост его продуктивности.

Многоуровневая структура CMM основывается на принципах обеспечения качества продукта. Для получения характеристик зрелости программного процесса в 1987 г. были разработаны методы внутренней и внешней оценки производственного процесса. Начиная с 1990 г., институт SEI с помощью многих энтузиастов из правительственных и отраслевых структур еще более развил и усовершенствовал эту модель, основываясь на опыте нескольких лет совершенствования производственных процессов.

7.4. Пять уровней зрелости производственного процесса

Постоянное совершенствование производственного процесса основано на многих небольших эволюционных шагах, а не на революционных нововведениях . CMM предоставляет концептуальную структуру, организующую эти эволюционные шаги в пять уровней зрелости, формирующих последовательные основания для постоянного совершенствования процесса. Эти пять уровней зрелости определяют порядковую шкалу для измерения зрелости и оценки продуктивности производственного процесса. Эти уровни также помогают организации расставить приоритеты среди своих мероприятий по улучшению процесса разработки.

Уровень зрелости представляет собой точно определенное эволюционное плато на пути к достижению полной зрелости производственного процесса. Каждый уровень зрелости формирует отдельный слой фундамента для постоянного совершенствования производственного процесса, включает в себя набор целей процесса, которые, по мере их достижения, приводят к стабилизации значимых компонентов производственного процесса. Достижение каждого уровня структуры

зрелости характеризуется внедрением различных составляющих производственного процесса, повышающих его продуктивность.

Последующие характеристики **пяти** уровней зрелости раскрывают основные изменения процессов, проводимые на каждом из них.

Начальный. Производственный процесс характеризуется как создаваемый каждый раз под конкретный проект, а иногда даже как хаотический. Определены лишь некоторые процессы и успех проекта зависит от усилий индивидуумов.

Повторяемый. Установлены основные процессы управления проектом, позволяющие отслеживать затраты, следить за графиком работ и функциональностью создаваемого программного решения. Установлена дисциплина процесса, необходимая для повторения достигнутых ранее успехов в проектах разработки подобных приложений.

Определенный. Производственный процесс документирован и стандартизован как для управленческих работ, так и для проектирования. Этот процесс интегрирован в стандартный производственный процесс организации. Во всех проектах используется утвержденная адаптированная версия стандартного производственного процесса организации.

Управляемый. Собираются подробные количественные показатели производственного процесса и качества создаваемого продукта. Как производственный процесс, так и продукты оцениваются и контролируются с количественной точки зрения.

Оптимизирующий. Постоянное совершенствование процесса достигается благодаря количественной обратной связи с процессом и реализации передовых идей и технологий.

7.5. Поведенческие характеристики уровней зрелости

Уровни зрелости от 2 до 5 могут характеризоваться работами, выполняемыми организацией в целях установления или совершенствования производственного процесса, работами по каждому проекту, а также итоговой продуктивности процессов во всех выполняющихся проектах. Поведенческая характеристика уровня 1 включена в целях создания основы для сравнения усовершенствований процессов на более высоких уровнях зрелости.

7.5.1 Уровень 1 – начальный уровень

Находясь на начальном уровне, организация обычно не может обеспечить устойчивый процесс разработки и сопровождения ПО. Когда в организации отсутствует культура управления, преимущества применения хороших решений в процессе проектирования исчезают из-за неэффективного планирования и плохой работы систем согласования.

Во время кризисных ситуаций в проектах зачастую отбрасываются запланированные процедуры и все усилия фокусируются на написании кода и тестировании. Успех целиком зависит от наличия исключительно эффективного менеджера и наличия опытного и квалифицированного коллектива разработчиков. Иногда талантливые и влиятельные менеджеры могут противостоять соблазну игнорировать стандартные плановые процедуры производственного процесса; но если такие менеджеры покидают проект, то они уносят вместе с собой и свое стабилизирующее влияние. Даже самый устойчивый процесс проектирования не сможет противостоять нестабильности, вызванной отсутствием над ежных практик управления.

Продуктивность производственного процесса организаций уровня 1 непредсказуема, что вызывается постоянными изменениями или модификациями производственного процесса по мере выполнения работ (т. е. процесс специально создается для каждого проекта). Графики работ, бюджеты, функциональность и качество продукта, как правило, непредсказуемы. Производительность зависит от возможностей отдельных сотрудников и изменяется в зависимости от присущих им навыков, знаний и мотивации. Существует лишь несколько стабильных

производственных процессов, а производительность можно прогнозировать только на уровне отдельных сотрудников, но не для организации в целом.

7.5.2. Уровень 2 – повторяемый уровень

На повторяемом уровне установлены политики управления проектом разработки и процедуры их применения. Планирование и управление новым проектом базируется на опыте работы с подобными проектами. Целью достижения уровня 2 является институционализация таких процессов эффективного управления проектами разработки, которые позволяют организациям воспроизводить успешные практики прежних проектов, хотя конкретные процессы различных проектов могут различаться. Эффективный процесс может быть охарактеризован как проверенный на практике, документированный, обязательный к выполнению, обучаемый, измеряемый и открытый для дальнейшего усовершенствования.

В проектах организаций второго уровня устанавливаются основные средства управления программным проектом. Реалистичные обязательства по проекту базируются на результатах прежних проектов и на требованиях текущего. Менеджеры проекта отслеживают производственные затраты, выполнение графиков и функциональность продукта; проблемы выполнения обязательств выявляются сразу после их возникновения. Требования к ПО и созданные на их основе рабочие продукты отслеживаются в системе управления конфигурацией, а их целостность контролируется. Определены стандарты проекта разработки и обеспечено их строгое соблюдение в рамках организации. В ходе проекта разработки проводится работа с субподрядчиками (при их наличии) по налаживанию надежных связей между заказчиком и субподрядчиком.

Продуктивность производственного процесса организаций уровня 2 может быть охарактеризована как дисциплинированная, так как планирование и отслеживание проекта разработки стабильно и возможно воспроизведение прежних достижений. Производственный процесс проекта находится под

эффективным управлением системы управления проектами и следует реалистичным планам, основанным на результатах прежних проектов.

7.5.3. Уровень 3 – определенный уровень

На определенном уровне, стандартный процесс разработки и сопровождения ПО в рамках организации надежно документирован, включая как процессы программной инженерии, так и управления, и эти процессы интегрированы в единое целое. Этот стандартный процесс в материалах CMM называется стандартным производственным процессом организации. Процессы, установленные на уровне 3, используются (и, по мере необходимости, изменяются) для помощи менеджерам и техническому персоналу в более эффективном выполнении своих задач. При стандартизации своих производственных процессов организация использует эффективные практики программной инженерии. Существует группа, которая ответственна за работы по координации производственного процесса организации, т. е. группа инженерии производственного процесса (SEPG). Реализована общая для организации программа обучения, гарантирующая, что персонал и руководящее звено обладают знаниями и навыками, требующимися для выполнения назначенных им ролей.

Во время работы над проектами выполняется адаптация стандартного производственного процесса организации с целью разработки производственного процесса, учитывающего уникальные характеристики проекта. Этот адаптированный процесс в материалах CMM называется производственным процессом, определенным для проекта. Определенный производственный процесс содержит взаимосвязанный, интегрированный набор четко определенных процессов управления и программной инженерии. В четко определенный процесс должны входить критерии готовности, входные данные, стандарты и процедуры выполнения работы, механизмы контроля (например, экспертные оценки), выходные данные и критерии завершения. Вследствие того, что производственный процесс ясно определен, руководство получает точную картину технического прогресса по всем проектам.

Продуктивность производственного процесса организаций третьего уровня может быть охарактеризована как стандартная и согласованная, поскольку и работы по управлению и программной инженерии стабильны и воспроизводимы. В пределах установленных линий продуктов затраты, график работ и реализуемые функциональные возможности находятся под строгим контролем, а качество создаваемого ПО непрерывно отслеживается. Продуктивность определенного производственного процесса основана на общем для всей организации понимании работ, ролей и сфер ответственности.

7.5.4. Уровень 4 – управляемый уровень

На управляемом уровне организация устанавливает количественные показатели качества, как для программных продуктов, так и для процессов их разработки. Для важных работ производственных процессов всех проектов выполняются измерения продуктивности и качества, как часть организационной программы измерений. Для сбора и анализа данных, получаемых от производственных процессов отдельных проектов, используется корпоративная база данных по производственным процессам. Производственные процессы уровня 4 оснащены инструментальными средствами для проведения точно определенных и согласованных измерений.

Эти измерения формируют количественную основу для оценки продуктов и производственных процессов проектов.

В ходе проектов контроль над процессами и создаваемыми продуктами достигается путем сужения разброса производительности процессов до приемлемых количественных пределов. Значимые расхождения в производительности процессов можно отличить от случайных расхождений (шумов), особенно внутри установленных линий продуктов. Риски, связанные с обучением персонала работе в новой прикладной области, известны и управляемы.

Продуктивность производственного процесса организаций уровня 4 может быть охарактеризована как предсказуемая, так как процесс функционирует в заданных и измеряемых пределах. Этот уровень продуктивности процесса позволяет организации прогнозировать тенденции развития процесса и качества продукта в пределах заданных количественных ограничений. При превышении этих пределов предпринимаются меры по коррекции ситуации. Создаваемые программные продукты имеют предсказуемо высокий уровень качества.

7.5.1. Уровень 5 – оптимизирующий уровень

Находясь на оптимизирующем уровне, вся организация полностью сосредоточена на непрерывном усовершенствовании производственного процесса. Организация обладает средствами профилактического выявления слабых мест процесса и его улучшения с целью предотвращения появления дефектов. Данные по эффективности производственного процесса используются для выполнения стоимостного анализа новых технологий и предлагаемых изменений производственного процесса организации. Выявляются новшества, использующие наилучшие методы программной инженерии, которые затем распространяются на всю организацию в целом.

В организациях пятого уровня группы проекта разработки анализируют обнаруженные дефекты и определяют причины их возникновения. Производственные процессы анализируются в целях предотвращения повторения известных типов дефектов, а полученный опыт распространяются на другие проекты.

Продуктивность процесса разработки ПО организаций 5-го уровня может быть охарактеризована как постоянно улучшающаяся, так как организации 5-го уровня зрелости постоянно стремятся улучшить диапазон продуктивности своего производственного процесса, повышая, таким образом, производительность процессов своих проектов. Улучшения происходят как за счет последовательного усовершенствования существующего процесса, так и за счет использования новых технологий и методов.

7.6. Понимание концепций уровней зрелости

CMM представляет собой описательную модель, поскольку она описывает существенные (или ключевые) атрибуты, которые, как предполагается, должны характеризовать организацию определенного уровня зрелости. Это модель является нормативной в том смысле, что подробные практики характеризуют обычные виды поведения, ожидаемого от организации, ведущей крупномасштабные проекты по государственным заказам. Обратите внимание, что CMM представляет собой достаточно высокий уровень абстракции и не выдвигает чрезмерных ограничений к способам реализации производственного процесса организации – эта модель просто описывает необходимые существенные атрибуты производственного процесса.

В любом контексте применения CMM следует разумно интерпретировать рекомендуемые практики. В том случае, когда бизнес-среда организации значительно отличается от среды крупной контрактной компании, модель CMM должна быть корректно интерпретирована с использованием информированных профессиональных оценок. CMM не является предписанием, в этой модели нет советов по способам усовершенствования организации. CMM описывает организацию на каждом уровне зрелости без предписания конкретных способов их достижения. Переход с уровня 1 на уровень 2 может занять несколько лет, а переход по остальным уровням обычно занимает около двух лет.

Усовершенствование производственного процесса происходит в контексте бизнес-целей и стратегических планов организации, ее организационной структуры, используемых технологий, социальной культуры и системы управления. CMM фокусируется на аспектах процессов тотального управления качеством (TQM). Успешное усовершенствование процессов подразумевает также учет вопросов вне рамок производственного процесса (например, проблемы с персоналом, которые возникают из-за изменений организационной культуры, предпринимаемых с целью внедрения и институционализации усовершенствований процессов).

7.6.1. Понимание концепции начального уровня

Хотя для организаций уровня 1 обычно характерны специально создаваемые и даже хаотические процессы, они, несмотря на выход за рамки бюджета и графика, часто разрабатывают вполне функциональные продукты. Успех для организаций уровня 1 зависит от компетентности и энтузиазма отдельных сотрудников. Подбор, наем, постоянное повышение квалификации и/или удержание компетентных сотрудников в коллективе представляют собой серьезные задачи для организаций всех уровней зрелости, но в основном эти задачи находятся за пределами рассмотрения CMM.

7.6.2. Понимание повторяемого и определенного уровней

По мере роста объема и сложности проекта, внимание постепенно смещается от технических вопросов к организационным и управленческим – т. е. к вопросам, которые находятся в фокусе рассмотрения модели зрелости процессов. Производственный процесс на этих уровнях позволяет сотрудникам работать более эффективно благодаря созданию документированных процессов, включающих в себя опыт наилучших работников, обретения навыков, необходимых для эффективного выполнения процессов (обычно с помощью обучения), и непрерывного совершенствования за счет обучения у сотрудников, реально выполняющих работу.

Для достижения уровня 2 руководство должно сфокусировать внимание на своих собственных процессах, добиваясь установления дисциплинированного производственного процесса. Уровень 2 формирует основу для уровня 3, так как внимание сосредоточено на руководстве, которое стремится усовершенствовать свои процессы прежде, чем приступить к техническим и организационным вопросам на уровне 3. На уровне 2 руководство занимает лидирующее положение путем документирования и следования процессам управления проектом.

Процессы организаций второго уровня могут различаться от проекта к проекту. Организационные требования для достижения уровня 2 заключаются в формировании политик, помогающих установить в проектах соответствующие

процессы управления. Документированные процедуры обеспечивают основу для согласованных процессов, которые могут быть установлены в рамках всей организации с помощью обучения и мероприятий по обеспечению качества ПО.

На этом фундаменте управления проектом строится уровень 3, который определяет, интегрирует и документирует весь производственный процесс организации. Под интеграцией в этом случае понимается процесс автоматической передачи выходных данных одной задачи на вход другой. Несоответствия между задачами выявляются и разрешаются на стадии планирования процесса до их фактического воздействия на производственный процесс. Одной из проблем уровня 3 является построение таких процессов, которые не содержат излишних ограничений на выполнение сотрудниками своей работы.

7.6.3. Понимание управляющего и оптимизированного уровней

Четвертый и пятый уровни зрелости относительно незнакомы для программной отрасли. Существуют лишь несколько примеров проектов разработки и организаций уровня 4 и 5. Для создания общей картины характеристик организаций этих уровней имеется слишком мало данных. Эти характеристики были определены по аналогии с другими отраслями промышленности и по нескольким примерам из программной отрасли, представляющими данный уровень продуктивности процессов.

Многие характеристики уровней 4 и 5 основаны на концепциях статистического управления процессами. Основные цели управления процессами иллюстрируются трехфазной диаграммой Джурана.

7.6.4. Зачем мерить зрелость?

Нужен ли нам формальный способ измерения зрелости? И если нужен, то чем он может быть полезен?

Если в наших планах проект по внедрению или развитию процессов, то, прежде, чем его затевать, нужно понять, в каком состоянии сейчас наше процессное управление. И, исходя из текущего состояния, определить направления развития.

При этом приоритетные направления определяется исходя из двух критериев: критичность процесса для предоставления услуг и текущий уровень зрелости.

Использование, общепринятых в отрасли методик определения зрелости позволяет определить наше место по отношению к конкурентам.

Следующее, для чего это может быть полезно – определить риски связанные с теми или иными процессами. И, следовательно, риски в предоставлении услуг, поддерживаемых этими процессами. Чем более зрелый процесс, тем меньше рисков.

Важно понимать, что высокая зрелость процесса отнюдь не дает гарантии его эффективной работы в любой ситуации, но дает определенную, измеримую степень уверенности. Т.е. чем выше зрелость, тем больше *вероятность* того, что процесс эффективен. Оценка рисков может быть особенно полезна при планировании внедрения новых услуг. Например, компания планирует внедрять ERP систему. Встает вопрос: сможет ли ИТ служба оказать адекватную поддержку? Оценка зрелости – один из инструментов, позволяющих ответить на этот вопрос.

Некоторые заказчики предъявляют формальные требования к уровню зрелости процессов поставщика услуг. Но даже если таких требований нет, высокий уровень зрелости, подтвержденный независимым уважаемым аудитором, поднимает имидж поставщика услуг.

7.6.5. Как мерить зрелость?

Существуют различные методики оценки уровня зрелости. Например, в CobiT для каждого процесса предложена своя модель зрелости.

Такие методики, как правило, представляют собой набор критериев. По количеству выполненных критериев определяется зрелость процесса.

Оценка проводится путем опроса ключевых сотрудников. В некоторых случаях могут требоваться документальные подтверждения выполнения критериев.

Оценку зрелости можно проводить собственными силами организации, либо привлекать внешнюю компанию. Первое, очевидно дешевле, второе объективней.

Сама по себе измеренная цифра ни о чем не говорит. После того как мы определили текущую зрелость, надо ответить на вопрос

7.6.6. Какой уровень зрелости нам нужен?

Эффективность работы процесса, конечно, зависит от уровня его зрелости, однако, зависимость это не линейная.

Известны примеры прекрасно работающих процессов, на втором уровне зрелости. Как правило, в устоявшихся, высокопрофессиональных коллективах. Часто также бывает, что процесс на третьем уровне зрелости неэффективен, по причине того, что формализованы были практики, отнюдь не лучшие.

При этом затраты на обеспечение зрелости растут тоже не линейно, а по экспоненте. Т.е. поднять уровень зрелости с уровня 2.5 до 3.5 стоит не очень дорого, а вот поднять зрелость с 4.8 до 4.9 стоит очень дорого. Это видно на рис.7.1.

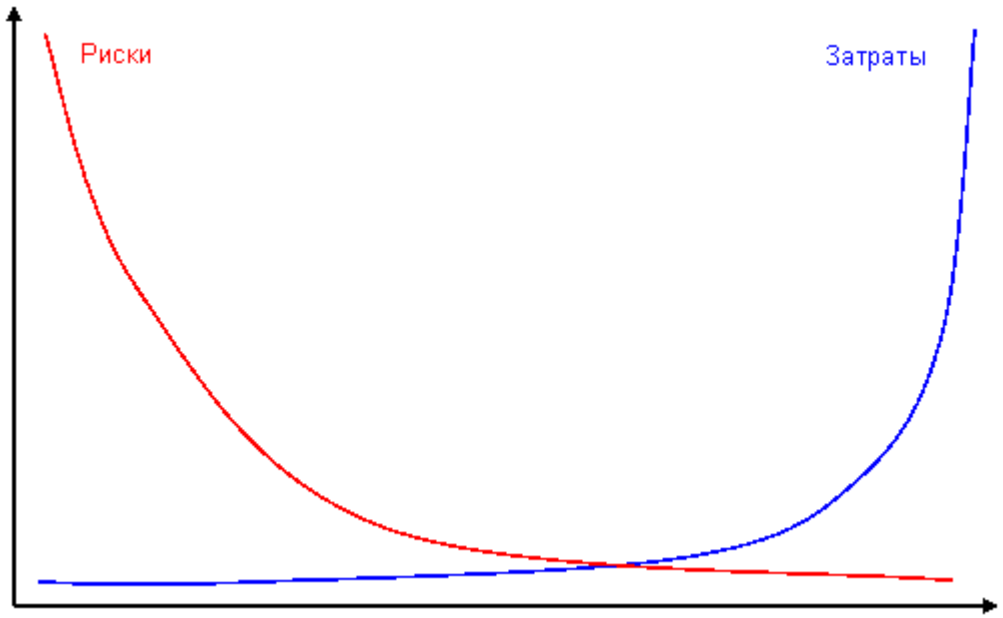


Рисунок 7.1 График зрелости процессов

Также на графике видно, что снижение рисков процесса, также нелинейно и при высоких уровнях зрелости незначительно. Однако, если изначально риски очень высоки (процесс критичен), то даже незначительное их снижение может быть оправдано.

Например, определением целевой архитектуры ИТ, как правило занимаются несколько очень высококвалифицированных сотрудников, прекрасно знающих свою работу. Даже в большом ИТ подразделении может не быть смысла вкладываться в формализацию этого процесса.

Обратный пример – процесс Управления Инцидентами, высоко критичный и затрагивающий почти всех сотрудников ИТ. Таким образом, для каждого процесса необходимо определить точку, где следует остановиться в его развитии, т.е. целевой уровень зрелости.

Универсальных методик для расчета целевого уровня зрелости нет. Однако понятно, что критический для предоставления услуг процесс с низкой зрелостью вызывает серьезные риски, в то время, как излишнее повышение зрелости некритичного процесса приведет к необоснованным затратам.

Кроме того, следует учитывать зрелость других процессов в организации. Например, существенно поднять уровень процесса управления инцидентами, невозможно без зрелого процесса управления конфигурациями.

Также очень важен уровень зрелости заказчика. Наивно было бы пытаться создавать развитый процесс управления уровнем услуг, когда заказчик не знает что такое SLA, и не готов его обсуждать и подписывать.

Таким образом, можно сформулировать несколько критериев, определяющих целевую зрелость процесса.

Критичность процесса для предоставления услуг.

Текущий уровень зрелости.

Уровень зрелости связанных процессов.

Уровень зрелости потребителя услуг, и внешних поставщиков.

Возвращаясь к управлению ИТ можно сказать, что деятельность по управлению – тоже процесс, зрелость которого складывается из зрелости всех поддерживающих процессов.

7.7. Модели зрелости процесса разработки (CMM, CMMI)

Помимо государственных стандартов, существует несколько подходов к сертификации процесса разработки. Наиболее известными из них в России являются, по-видимому, CMM и CMMI.

CMM (Capability Maturity Model) – модель зрелости процессов создания ПО, которая предназначена для оценки уровня зрелости процесса разработки в конкретной компании. В соответствие с этой моделью есть пять уровней зрелости процесса разработки. Первый уровень соответствует разработке «как получится», когда на каждый проект разработчики идут как на подвиг. Второй уровень соответствует более-менее налаженным процессам, когда можно с достаточной уверенностью надеяться на положительный исход проекта. Третий уровень соответствует наличию разработанных и хорошо описанных процессов, используемых при разработке. Четвертый – активному использованию метрик в процессе управления для постановки целей и контроля их достижения. И, наконец, пятый уровень означает способность компании оптимизировать процесс по мере необходимости.

После появления CMM стали разрабатываться специализированные модели зрелости для разработки информационных систем, для процесса выбора поставщиков и некоторые другие. На их основе была разработана интегрированная модель CMMI (Capability Maturity Model Integration). Кроме того, в CMMI была предпринята попытка преодолеть проявившиеся к тому времени недостатки CMM. А именно, преувеличение роли формальных описаний процессов, когда наличие определенной документации оценивалось значительно выше, чем просто хорошо налаженный, но не описанный процесс. Тем не менее, CMMI также ориентирован на использование весьма формализованного процесса.

Таким образом, основой CMM и CMMI является формализация процесса разработки. Они нацеливают разработчиков, желающих сертифицироваться на достаточно высокую степень зрелости процесса разработки, на внедрение жесткого формализованного процесса. На максимальных уровнях зрелости

предполагается, что этот процесс настраивается для каждого конкретного проекта. Однако, в рамках проекта процесс остается очень жестким.

Связь CMM и CMMI с итеративной разработкой более опосредованная. Формально и та, и другая не выдвигают конкретных требований к тому, чтобы придерживаться каскадного или итеративного подхода. Однако, по мнению ряда специалистов, CMM в большей степени совместим с каскадным подходом, в то время как CMMI допускает также и применение итеративного подхода (рис.2).

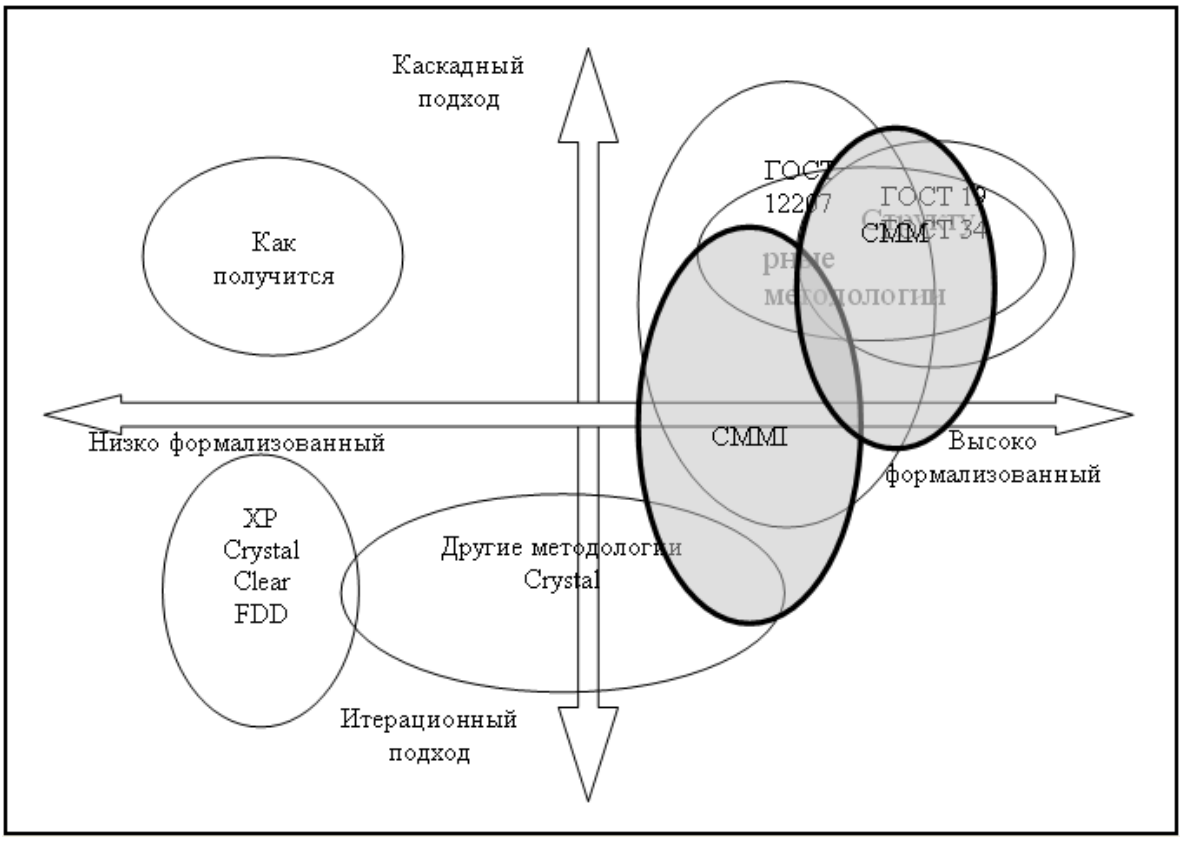


Рис.2 Применение CMMI

7.8. Введение в CMMI: основные термины и понятия

В 2003 году американский Институт программной инженерии (SEI) опубликовал новую комплексную модель CMMI, уточняющую и совершенствующую предшествовавшие модели CMM, а также частично учитывающую основные требования существующих международных стандарто*в* в области менеджмента программных средств. Внедрение этой модели акцентировано на улучшении процессов управления проектами ПС, обеспечении

их высокого качества и конкурентоспособности, с основной целью – сделать процессы проектов *управляемыми*, а результаты – *предсказуемыми*. Значительное внимание в CMMI уделяется процессам разработки и учету итераций при изменении требований заказчиков, их прослеживанию к функциям, компонентам, тестам и документам проекта. Концепцию, определяющую функциональную пригодность и качество продукта, рекомендуется сопровождать проверками возможных сценариев событий при его применении.

В последнее время появилась информация о модернизации американским Институтом программной инженерии SEI версии CMMI-SE/SW/IPPD, V1.1 на основе накопленного опыта и отзывов предприятий. Предполагается выпустить в 2006 году новую, существенно модернизированную, версию модели CMMI-V1.2, после чего постепенно должно прекратиться применение версии 1.1. До конца 2007 года должен проводиться переход пользователей на версию CMMI-V1.2, а в дальнейшем она станет обязательной для формализованной оценки качества (сертификации) технологии предприятий в области программной инженерии. При этом срок действия сертификата будет ограничен тремя годами. К этим изменениям следует готовиться после официальной публикации Институтом SEI версии 1.2.

Два варианта модели CMMI – 1.1 созданы для обеспечения непрерывного оценивания комплекса процессов в определенной области создания программных средств или для поэтапного оценивания и совершенствования зрелости предприятия, а также для организации ЖЦ комплексов программ в целом. Модели CMMI представляют помощь специалистам при организации технологии и совершенствовании их продуктов, а также для упорядочения и обслуживания процессов разработки и сопровождения ПС. Концепция этих моделей покрывает управление и оценивание зрелости сложных систем, инженерии программных средств, а также процессов создания интегрированных программных продуктов и совершенствования их разработки. Компоненты непрерывной и поэтапной моделей в значительной степени подобны, могут выбираться и применяться в разном

составе и последовательности использования в зависимости от свойств и характеристик конкретных проектов.

В стандарте CMMI вводится достаточно много терминов и понятий, составляющих основу этого стандарта. Можно отметить две особенности терминологии стандарта CMMI:

Определенные в CMMI термины и понятия составляют «философию» CMMI, могут рассматриваться как введение в CMMI и являются ключом к пониманию этого стандарта

Большая часть определенных в CMMI терминов и понятий являются «новыми», специфичными для этого стандарта, но в стандарте используется ряд «старых» терминов и понятий, имеющих специфическую для этого стандарта трактовку.

7.8.1. Что такое процесс?

Процесс – базовое понятие управления качеством, идет от TQM и входит во все стандарты, связанные с управлением качеством. Как следствие – определяется по-разному. При этом, в определении процесса была некоторая двойственность. Так, в ISO12207 жизненный цикл ПО определяется как непрерывный *процесс* от момента принятия решения до снятия продукта с эксплуатации. Далее жизненный цикл разбивается на отдельные *процессы* (управления, разработки, тестирования, сопровождения), и понятие *процесс* определяются как набор взаимосвязанных действий (activities), которые преобразуют исходные данные в выходные результаты. Действия процесса разбиваются на отдельные задачи.

В CMMI дается следующее определение процесса: «Процесс – деятельность организации, направленная на достижение ее бизнес целей и имеющая результатом создание продукта или услуги. Процесс – набор действий, покрывающих всю деятельность организации и представленных в виде практик модели CMMI. Результатом процесса может быть не только создание продукта (услуги), но и повышение зрелости процесса».

В CMMI определены 25 областей процесса, которые разбиты на 4 категории:

Process Management (Управление процессом):
- OPF: Organizational Process Focus (Фокус процесса)
- OPD: Organizational Process Definition (Определение процесса)
- OT: Organizational Training (Обучение)
- OPP: Organizational Process Performance (Производительность процесса)
- OID: Organizational Innovation and Deployment (Инновация и развертывание)

Project Management (Управление проектом):
- PP: Project Planning (Планирование проекта)
- PMC: Project Monitoring and Control (Отслеживание и контроль проекта)
- SAM: Supplier Agreement Management (Управление подрядчиками)
- IPM: Integrated Project Management (Управление интегрированным проектом)
- RSKM: Risk Management (Управление рисками)
- IT: Integrated Teaming (Интегрированное управление командами)[1]
- ISM: Integrated Supplier Management (Интегрированное управление подрядчиками)
- QPM: Quantitative Project Management (Количественное управление проектом)

Engineering (Инженерия):
- REQD: Requirements Development (Разработка требований)
- REQM: Requirements Management (Управление требованиями)
- TS: Technical Solution
- PI: Product Integration (Интеграция продукта)
- VER: Verification (Верификация)
- VAL: Validation (Валидация)

Support (Поддержка):
- CM: Configuration Management (Управление конфигурацией)
- PPQA: Process and Product Quality Assurance (Управление качеством процесса и продукта)
- MA: Measurement and Analysis (Измерение и анализ)
- OEI: Organizational Environment for Integration (Инфраструктура для интеграции)
- DAR: Decision Analysis and Resolution ( )
- CAR: Causal Analysis and Resolution (Анализ и разрешение проблем)

Каждая область процесса описывается следующими характеристиками:

Назначение (Purpose)
Вводные замечания (Introductory Notes)
Связанные области процесса (Related Process Areas)
S**pecific goal** (специальные цели)
Specific practice (специальные практики)
Work product (рабочие продукты)
Специальные подпрактики.

7.8.2. Назначения и цели областей процесса

Категория Process Management. Категория Process Management содержит проходящие через весь проект действия, относящиеся к определению, планированию, развертыванию, вводу в действие, отслеживанию, контролю, оцениванию, измерению и улучшению процесса.

Области процесса, входящие в категорию Process Management, имеют следующие назначения:

OPF Organizational Process Focus (Фокус процесса)

Планировать и выполнять совершенствование процесса организации, основываясь на досконального (полном) понимании текущих сильных сторон и недостатков процессов организации и активов процесса (process assets).

Цели области:

Планируй и применяй действия по улучшению процесса.

OPD Organizational Process Definition (Определение процесса)

Установить и поддерживать (establish and maintain) подходящий набор активов процесса (process assets) организации

Цели области:

Установи активы процесса организации.

OT Organizational Training (Обучение)

Совершенствовать умения и знание людей с тем, чтобы они могли выполнять свои роли эффективно и квалифицировано.

Цели области:

Установи возможности обучения.

Проводи необходимое обучение.

OPP Organizational Process Performance (Производительность процесса)

установить и поддерживать (establish and maintain) количественное понимание производительности набора стандартных процессов (standard processes) организации в подтверждение целей качества и производительности процесса (process-performance) и

обеспечить данные, базовые линии (baselines) и модели производительности процесса для количественного управления проектами организации.

Цели области:

Установи базовую линию и модель производительности.

OID Organizational Innovation and Deployment (Инновация и развертывание).

Выбирать и развертывать пошаговые и инновационные усовершенствования, которые измеримо улучшают процессы и технологии организации. Улучшения поддерживают целевые установки качества и производительности процесса в той мере, в какой это соответствует бизнес целям организации

Цели области:

Select Improvements (Выбери улучшения)

Deploy Improvements (Развертывай улучшения)

Категория Project Management. Данная область покрывает деятельность по управлению проектом, связанную с планированием, мониторингом и контролем за процессом разработки.

PP Project Planning

Установить и сопровождать планы, которые определяют действия проекта

PMC Project Monitoring and Control

обеспечить понимание продвижения проекта так, чтобы соответствующие корректирующие действия могли быть предприняты когда выполнение проекта значительно отклоняется от плана.

SAM Supplier Agreement Management

управлять приобретением продуктов у оставщиков, с которыми существует формальное соглашение.

IPM Integrated Project Management

устанавливать и управлять проектом и привлечением уместных заинтересованных лиц согласно интегрированному и определенному процессу, который адаптирован из набора стандартных процессов организации.

Для дисциплины IPPD, Integrated Project Management также покрывает установление согласованного видения проекта и структуры команды для интегрированных команд, которые будут выполнять цели проекта.

RSKM Risk Management

идентифицировать потенциальные проблемы прежде, чем они возникают так, чтобы вызванные риском действия могли быть запланированы и при необходимости выполнены в любой момент создания продукта или выполнения проекта и смягчить неблагоприятные воздействия на достижение целей.

IT Integrated Teaming

формировать и поддерживать интегрированную группу для разработки рабочих продуктов.

ISM Integrated Supplier Management

упреждающе идентифицировать источники продуктов, которые могут использоваться для удовлетворения требований проекта и

управлять выбранными поставщиками при поддержании совместных отношений «проект-поставщик».

QPM Quantitative Project Management

количественно управлять настроенным на проект процессом, чтобы достигнуть установленных целей качества (продукта) и производительности проекта.

Категория Engineering. Инженерные области процесса охватывают действия по разработке и сопровождению, которые разделены среди технических дисциплин. Инженерные области процесса также объединяют процессы программной инженерии и системной инженерии в единый процесс разработки продукта, поддерживая ориентированную на продукт стратегию усовершенствования процесса. Инженерные области процесса применяются к разработке любого продукта или услуги в области технической разработки (например, продуктах программного обеспечения, продуктах аппаратных средств, услугах или процессах).

REQD Requirements Development (Разработка требований)

выявить и проанализировать требования заказчика, требования на продукт и его компоненты

REQM Requirements Management (управление требованиями)

управлять требованиями на продукты проекта и компоненты продукта и

идентифицировать несогласованности между этими требованиями и планами и рабочими продуктами проекта

TS Technical Solution (техническое решение)

проектировать, разрабатывать, и осуществлять решения для удовлетворения требований. Решения, замыслы и реализации охватывают продукты, компоненты продукта и связанный с продуктом процессы жизненного цикла либо по отдельности, либо в соответствующей комбинации.

PI Product Integration (интеграция продукта)

собрать продукт из компонент, гарантировать, что в сборке он работает должным образом и поставить продукт

VER Verification (верификация)

гарантировать, что выбранные рабочие продукты выполняют предписанные им требования.

VAL Validation (валижация)

продемонстрировать, что продукт или компонента продукта выполняют свои предназначенное применение будучи помещенным в предназначенную для этого среду.

Категория Support. Категория Support охватывает действия, которые поддерживают разработку и сопровождение продукта. Области процесса категории включают процессы, которые используются в контексте выполнения других процессов. В общем, области процесса категории Support охватывают процессы, которые направлены на проект и могут включать процессы, которые применяются более широко к организации. Например, Process and Product Quality Assurance может использоваться со всеми областями процесса, чтобы обеспечить объективную оценку процессов и рабочих продуктов, описанных во всех областях процесса.

CM Configuration Management (управление конфигурацией)

устанавливать и поддерживать целостность рабочих продуктов, используя идентификацию конфигурации, контроль конфигурации, учет статуса конфигурации и аудит конфигурации.

PPQA Process and Product Quality Assurance (оценка качества продукта и процесса)

обеспечивать персонал и управление объективным пониманием процессов и ассоциированных рабочих продуктов.

MA Measurement and Analysis (измерение и анализ)

разрабатывать и поддерживать возможность измерений, которые используются для поддержки информационных потребностей управления.

OEI Organizational Environment for Integration (организационное окружение для интеграции)

обеспечивать инфраструктуру Integrated Product and Process Development (IPPD) и управлять людьми для интеграции.

DAR Decision Analysis and Resolution (анализ решений)

анализировать возможные решения, используя формальный процесс оценки, который оценивает идентифицированные альтернативы по установленным критериям.

CAR Causal Analysis and Resolution (анализ причин)

идентифицировать причины дефектов и других проблем и принимать меры, препятствующие их возникновению в будущем.

7.8.3. Maturity levels – уровни зрелости процесса по CMMI

Области процесса распределены по 6 уровням зрелости процесса (maturity levels), которые являются существенным элементом модели зрелости CMMI (capability maturity model):

0 – Incomplete process (неполный процесс)

Процесс не выполняется или выполняется только частично. Одна или более специфических целей процесса не удовлетворены.

1 – Performed process (выполняемый процесс)

Процесс, который выполняет необходимую работу для создания рабочих продуктов. Специальные цели области процесса удовлетворяются.

2 – Managed process (управляемый процесс)

Выполняемый процесс, который планируется и выполняется в соответствии с политикой; использует квалифицированные кадры, имеющие адекватные ресурсы для создания контролируемых результатов (outputs); привлекает релевантных заинтересованных лиц; отслеживаемый; контролируемый и обозреваемый; оцениваемый на соблюдение его описания.

3 – Defined process (определенный процесс)

Управляемый процесс, который:

сформирован из (сшит на заказ, приспособлен) набора стандартных процессов организации в соответствии с принятыми в организации правилами формирования процессов;

имеет поддерживаемое описание процесса;

вносит рабочие продукты, метрики и другую информацию по улучшению процесса в активы процесса организации.

4 – Quantitatively managed process (количественно управляемый процесс)

Определенный процесс, который управляется с использованием статистических или других количественных методов. Качество продукта, качество сервиса и атрибуты производительности процесса измеримы и контролируются на всем протяжении проекта.

5 – Optimizing process (оптимизирующий процесс)

Количественно управляемый процесс, который улучшается на основе понимания общих причин изменений, присущих процессу. Фокус оптимизирующего процесса – на постоянном улучшении степени производительности процесса путем как наращиваемых, так и инновационных усовершенствований.

ВОПРОСЫ

1. Может ли незрелая организация достичь успеха?
2. Да, может. Но он не стабилен, имеет скорее случайный характер. Велики риски внутренние (уход ведущих сотрудников) и внешние (новый тип ПО, новые технологии)
3. Есть ли какие-то преимущества у зрелой организации? В чем, если есть?
4. Стабильность результатов, обеспеченная установленным процессом.
5. В чем состоит идея модели зрелости?
6. Кратко – уровни зрелости, повышающиеся требования к уровням, эволюция с уровня на уровень.
7. Можно ли перескакивать через уровни?
8. Вроде бы и да. Например, инновации надо вводить на всех уровнях. Но вводимые инновации надо оценивать (иначе может быть обратный эффект). Для объективной оценки нужны измерения. Т.е. в полной мере область OID выполнима на 5 уровне, хотя отдельные ее элементы выполнимы и раньше.
9. Какие уровни зрелости вы запомнили и в чем их отличия?

ЛИТЕРАТУРА

1. С.Н. Карпенко. Введение в программную инженерию. Курс лекций. ННГУ, Н.Новгород, 2005. [Электронный ресурс] Режим доступа: www.unn.ru/pages/e-library/aids/2007/16.pdf.
2. Марк Паулк и др. Модель зрелости процессов разработки программного обеспечения – Capability Maturity Model for Software (CMM) Интерфейс-Пресс. 2002 г. · 256с.
3. Оценка и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504 CMM) / Пер.с англ. А.С. Агапов, С.В. Зенин, Н.Э. Михайловский, А.А. Мкртумян А.А. – М.: Книга и бизнес, 2001. – 348с. [Электронный ресурс] Режим доступа: http://www.ntrlab.ru/rus/method/iso15504/15504-intro.html#_Toc495139005
4. Модель зрелости процессов разработки программного обеспечения

Электронная библиотека. [Электронный ресурс] Режим доступа: http://www.RoyalLib.Ru, 2010-2012.

8. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

«Тестирование программ может использоваться для демонстрации наличия ошибок, но оно никогда не покажет их отсутствие»

– Дейкстра, 1970 г.

8.1 Понятие тестирования программного обеспечения

Как бы ни была тщательно отлажена программа, решающим этапом, устанавливающим ее пригодность для работы, является ТЕСТИРОВАНИЕ. Ведь ни один программист не застрахован от ошибок!

Но, как справедливо указывал известный теоретик программирования Э. Дейкстра, тестирование может показать лишь наличие ошибок, но не их отсутствие.

Качество программного продукта характеризуется набором свойств, определяющих, насколько продукт «хорош» с точки зрения заинтересованных сторон, таких как заказчик продукта, спонсор, конечный пользователь, разработчики и тестировщики продукта, инженеры поддержки, сотрудники отделов маркетинга, обучения и продаж. Каждый из участников может иметь различное представление о продукте и о том, насколько он хорош или плох, то есть о том, насколько высоко качество продукта. Таким образом, постановка задачи обеспечения качества продукта выливается в задачу определения заинтересованных лиц, их критериев качества и затем нахождения оптимального решения, удовлетворяющего этим критериям. Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта.

С технической точки зрения тестирование заключается в выполнении приложения на некотором множестве исходных данных и сверке получаемых результатов с заранее известными (эталонными) с целью установить соответствие различных свойств и характеристик приложения заказанным свойствам. Как одна

из основных фаз процесса разработки программного продукта (Дизайн приложения – Разработка кода – Тестирование), тестирование характеризуется достаточно большим вкладом в суммарную трудоемкость разработки продукта

8.1.1. Методы обеспечения качества ПО

Перечислим различные способы контроля качества, используемые на практике при разработке ПО.

Наладка качественного процесса, другими словами совершенствование процесса. Для комплексного улучшения процессов в компании (подход technology push) компаниями-разработчиками ПО используются стандарты CMM/CMMI, а также по стандартам серии ISO 9000 (с последующей официальной сертификацией). Применяются и локальные стратегии, менее дорогостоящие и более направленные на решению отдельных проблем (подход organization pull).

Формальные методы – использование математических формализмов для доказательства корректности, спецификации, проверки формального соответствия, автоматической генерации и т.д.: доказательство правильности работы программ, проверка на моделях определенных свойств (model cheking), статический анализ кода по дереву разбора программы (например, проверка корректности кода по определенным критериям – аккуратная работа с памятью, поиск мертвого кода и пр.), модельно-ориентированное тестирование (model-based testing): автоматическая генерация тестов и тестового окружения по формальным спецификациям требований к системе) и т.д. На практике применяются ограниченно из-за необходимости серьезной математической подготовки пользователей, сложности в освоении, большой работы по развертыванию. Эффективны для систем, имеющих повышенные требования к надежности. Также имеются случаи эффективного использования средств, основанных на этих методах, в руках высококвалифицированных специалистов.

Исследование и анализ динамических свойств ПО. Например, широко используется профилирование – исследование использования системой памяти, ее быстродействие и др. характеристик путем запуска и непосредственных

наблюдений в виде графиков, отчетов и пр. В частности, этот подход используется при распараллеливании программ, при поиске «узких» мест. Еще пример – область, называемая «моделирование и анализ производительности» (performance modeling and analysis). Здесь моделируется нагрузочное окружение системы (число одновременных пользователей системы, сетевой трафик и пр.) и наблюдается поведение системы.

Обеспечение качества кода. Сюда относится целый комплекс различных мероприятий и методов. Вот некоторые, самые известные из них.

- Разработка *стандартов оформления кода* в проекте и контроль за соблюдением этих стандартов. Сюда входят правила на создание идентификаторов переменных, методов и имен классов, на оформление комментариев, правила использования стандартных для проекта библиотек и т.д. Регулярный рефакторинг для предотвращения образования из кода «вермишели». Существует тенденция ухудшения структуры кода при внесении в него новой функциональности, исправления ошибок и пр. Появляется избыточность, образуются неиспользуемые или слабо фрагменты, структура становится запутанной и трудной для понимания.
- *Рефакторинг* – это регулярная деятельность по переписываю кода, но не с целью добавления ново функциональности, а для улучшения его структуры. Рефакторинг появился в контексте «гибких» методов, в данный момент активно поддерживается различными средами разработки ПО.
- *Различные варианты инспекции кода*, например, техника peer code review. Последняя заключается в том, что код каждого участника проекта, выборочно, читается и обсуждается на специальных встречах (code review meetings), и делается это регулярно. Практика показывает, что в целом код улучшается. Есть еще такой подход, как «вычитка» кода, используемый, например, при разработке критических систем

реального времени. Ею занимаются также разработчики, но их роль в данном проекте – вычитка, а не разработка.

Тестирование. Самый распространенный способ контроля качества ПО, представленный, фактически, в каждом программном проекте.

8.1.2. Понятие тестирования

Тестирование (software testing) – деятельность, выполняемая для оценки и улучшения качества программного обеспечения [1]. Эта деятельность, в общем случае, базируется на обнаружении дефектов и проблем в программных системах.

Тестирование программных систем состоит из *динамической* верификации поведения программ на *конечном (ограниченном)* наборе тестов, *выбранных* соответствующим образом из обычно выполняемых действий прикладной области и обеспечивающих проверку соответствия *ожидаемому* поведению системы.

В данном определении тестирования выделены слова, определяющие основные вопросы, которым адресуется данная область знаний:

Динамичность (dynamic): этот термин подразумевает тестирование всегда предполагает выполнение тестируемой программы с заданными входными данными. При этом, величины, задаваемые на вход тестируемому программному обеспечению, не всегда достаточны для определения теста. Сложность и недетерминированность систем приводит к тому, что система может по разному реагировать на одни и те же входные параметры, в зависимости от состояния системы. В данной области знаний термин «вход» (input) будет использоваться в рамках соглашения о том, что вход может также специфицировать состояние системы, в тех случаях, когда это необходимо. Кроме динамических техник проверки качества, то есть тестирования, существуют также и статические техники, рассматриваемые в области знаний «Software Quality».

Конечность (ограниченность, finite): даже для простых программ теоретически возможно столь большое количество тестовых сценариев, что исчерпывающее тестирование может занять многие месяцы и даже годы. Именно поэтому, с практической точки зрения, всестороннее тестирование считается

бесконечным. Тестирование всегда предполагает компромисс между ограниченными ресурсами и заданными сроками, с одной стороны, и практически неограниченными требованиями по тестированию, с другой. То есть мы снова говорим об определении характеристик «приемлемого» качества, на основе которых планируем необходимы объем тестирования.

Выбор (selection): многие предлагаемые техники тестирования отличаются друг от друга в том, как выбираются сценарии тестирования. Инженеры по программному обеспечению должны обладать представлением о том, что различные критерии выбора тестов могут давать разные результаты, с точки зрения эффективности тестирования. Определение подходящего набора тестов для заданных условий является очень сложной проблемой. Обычно, для выбора соответствующих тестов совместно применяют техники анализа рисков, анализ требований и соответствующую экспертизу в области тестирования и заданной прикладной области.

Ожидаемое поведение *(expected* behaviour*)*: Хотя это не всегда легко, все же необходимо решить, какое наблюдаемое поведение программы будет приемлемо, а какое – нет. В противном случае, усилия по тестированию – бесполезны. Наблюдаемое поведение может рассматриваться в контексте пользовательских ожиданий (подразумевая «тестирования для проверки» – testing for validation), спецификации («тестирование для аттестации» – testing for verification) или, наконец, в контексте предсказанного поведения на основе неявных требований или обоснованных ожиданий.

Общий взгляд на тестирование программного обеспечения последние годы активно эволюционировал, становясь все более конструктивным, прагматичным и приближенным к реалиям современных проектов разработки программных систем. Тестирование более не рассматривается как деятельность, начинающаяся только после завершения фазы конструирования. Сегодня тестирование рассматривается как деятельность, которую необходимо проводить на протяжении всего процесса разработки и сопровождения и является важной частью конструирования

программных продуктов. Действительно, планирование тестирования должно начинаться на ранних стадиях работы с требованиями, необходимо систематически и постоянно развивать и уточнять планы тестов и соответствующие процедуры тестирования. Даже сами по себе сценарии тестирования оказываются очень полезными для тех, кто занимается проектированием, позволяя выделять те аспекты требований, которые могут неоднозначно интерпретироваться или даже быть противоречивыми.

Не секрет, что легче предотвратить проблему, чем бороться с ее последствиями. Тестирование, наравне с управлением рисками, является тем инструментом, который позволяет действовать именно в таком ключе. Причем действовать достаточно эффективно. С другой стороны, необходимо осознавать, что даже если приемочные тесты показали положительные результаты, это совсем не означает, что полученный продукт не содержит ошибок.

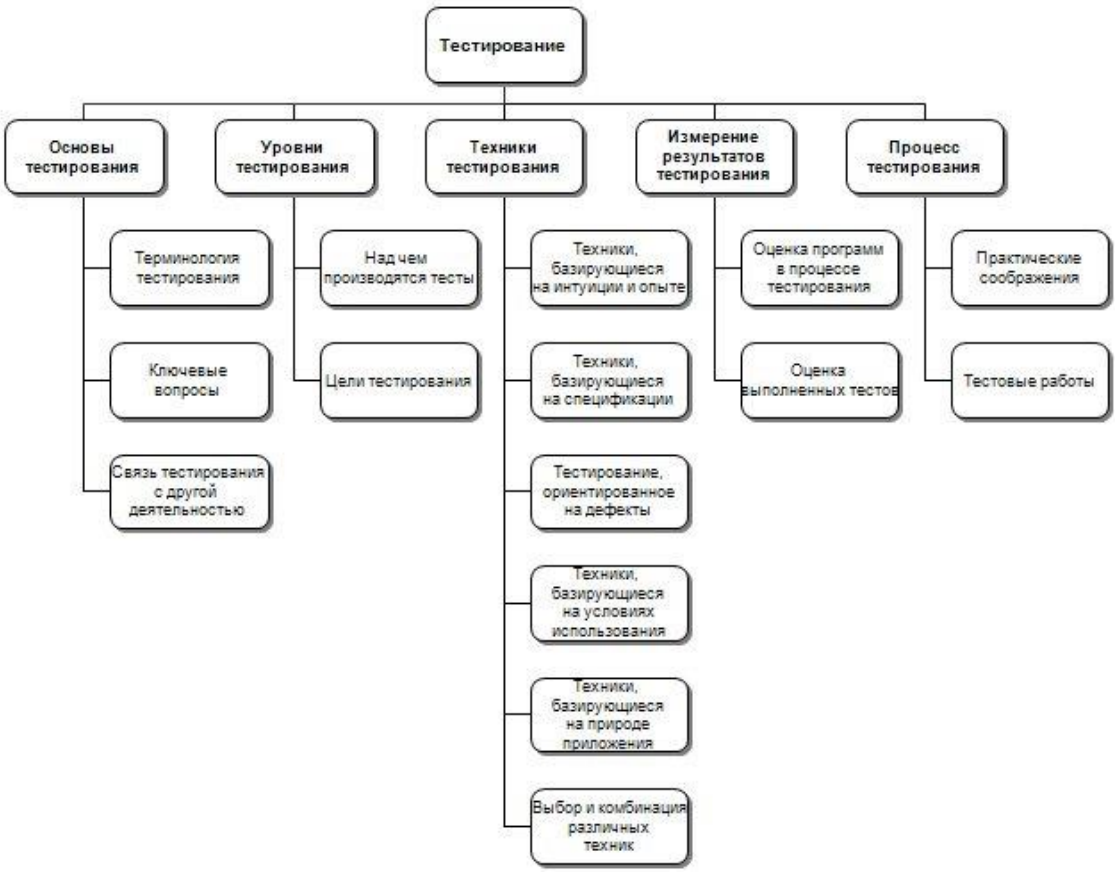


Рис. 1. Область знаний «Тестирование программного обеспечения» SWEBOK

8.2. История развития тестирования программного обеспечения

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

В 1960-х много внимания уделялось «исчерпывающему» тестированию, которое должно проводиться с использованием всех путей в коде или всех возможных входных данных. Было отмечено, что в этих условиях полное тестирование ПО невозможно, потому что, во-первых, количество возможных входных данных очень велико, во-вторых, существует множество путей, в-третьих, сложно найти проблемы в архитектуре и спецификациях. По этим причинам «исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

В начале 1970-х тестирование ПО обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы ПО». В зарождавшейся программной инженерии верификация ПО значилась как «доказательство правильности». Хотя концепция была теоретически перспективной, на практике она требовала много времени и была недостаточно всеобъемлющей. Было решено, что доказательство правильности – неэффективный метод тестирования ПО. Однако, в некоторых случаях демонстрация правильной работы используется и в наши дни, например, приемо-сдаточные испытания. Во второй половине 1970-х тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает. Успешный тест – это тест, который обнаруживает ранее неизвестные проблемы. Данный подход прямо противоположен предыдущему. Указанные два определения представляют собой «парадокс тестирования», в основе которого лежат два противоположных утверждения: с одной стороны,

тестирование позволяет убедиться, что продукт работает хорошо, а с другой – выявляет ошибки в ПО, показывая, что продукт не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки ПО.

В 1980-х тестирование расширилось таким понятием, как предупреждение дефектов. Проектирование тестов – наиболее эффективный из известных методов предупреждения ошибок. В это же время стали высказываться мысли, что необходима методология тестирования, в частности, что тестирование должно включать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс. В ходе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты. «Традиционное» тестирование, существовавшее до начала 1980-х, относилось только к скомпилированной, готовой системе (сейчас это обычно называется системное тестирование), но в дальнейшем тестировщики стали вовлекаться во все аспекты жизненного цикла разработки. Это позволяло раньше находить проблемы в требованиях и архитектуре и тем самым сокращать сроки и бюджет разработки. В середине 1980-х появились первые инструменты для автоматизированного тестирования. Предполагалось, что компьютер сможет выполнить больше тестов, чем человек, и сделает это более надежно. Поначалу эти инструменты были крайне простыми и не имели возможности написания сценариев на скриптовых языках.

В начале 1990-х в понятие «тестирование» стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества, охватывающего весь цикл разработки ПО. В это время начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования. В середине 1990-х с развитием сети Internet и разработкой большого количества Web-приложений, особую популярность стало получать так

называемое «гибкое тестирование» (по аналогии с гибкими методологиями программирования).

В 2000-х появилось еще более широкое определение тестирования, когда в него было добавлено понятие «оптимизация бизнес-технологий» business technology optimization, BTO). BTO направляет развитие информационных технологий в соответствии с целями бизнеса. Основной подход заключается в оценке и максимизации значимости всех этапов жизненного цикла разработки ПО для достижения необходимого уровня качества, производительности, доступности.

8.3. Методики тестирования ПО

Существующие на сегодняшний день методы тестирования ПО не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПО.

Такой процесс формальной проверки или верификации, может доказать, что дефекты отсутствуют с точки зрения используемого метода. (То есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учетом человеческого фактора, присутствующего на всех этапах жизненного цикла ПО).

Существует множество подходов к решению задачи тестирования и верификации ПО, но эффективное тестирование сложных программных продуктов – это процесс в высшей степени творческий, не сводящийся к следованию строгим и четким процедурам или созданию таковых.

С точки зрения ISO 9126, качество программного обеспечения можно определить как совокупную характеристику исследуемого ПО с учетом следующих составляющих:

- Надежность;
- Сопровождаемость;

– Практичность;
– Эффективность;
– Мобильность;
– Функциональность.

Более полный список атрибутов и критериев можно найти в стандарте ISO 9126 Международной организации по стандартизации. Состав и содержание документации, сопутствующей процессу тестирования, определяется стандартом IEEE 829-1998 Standard for Software Test Documentation.

8.4. Классификация видов тестирования

По уровню тестируемых программных объектов обычно выделяют:

– Модульное тестирование:
– Интеграционное тестирование;
– Системное тестирование;
– Альфа-тестирование;
– Бета-тестирование.

Модульное тестирование (юнит-тестирование) – тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Часто модульное тестирование осуществляется разработчиками ПО.

Интеграционное тестирование – тестируются интерфейсы между компонентами, подсистемами или системами. При наличии резерва времени на данной стадии тестирование ведется итерационно, с постепенным подключением последующих подсистем.

Системное тестирование – тестируется интегрированная система на ее соответствие требованиям.

Альфа-тестирование – имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приемочного тестирования. Иногда

альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщикам для дополнительного исследования в окружении, подобном тому, в котором будет использоваться ПО.

Бета-тестирование – в некоторых случаях выполняется распространение предварительной версии для некоторой большей группы лиц, с тем чтобы убедиться, что продукт содержит достаточно мало ошибок. В случае проприетарного ПО, в бета-версию иногда вносятся ограничения по функциональности или времени работы. Бета-тестирование выполняется также для того, чтобы получить обратную связь о продукте от его будущих пользователей.

- Тестирование по объектам тестирования подразделяется на:
- Функциональное тестирование;
- Тестирование производительности;
    - Нагрузочное тестирование;
    - Стресс-тестирование;
    - Тестирование стабильности;
- Юзабилити-тестирование (проверка эргономичности);
- Тестирование интерфейса пользователя;
- Тестирование безопасности;
- Тестирование локализации;
- Тестирование совместимости;

Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

Тестирование производительности – тестирование, которое проводится с целью определения, как быстро работает система или ее часть под определенной нагрузкой. Также может служить для проверки и подтверждения других атрибутов

качества системы, таких как масштабируемость, надежность и потребление ресурсов.

Тестирование стабильности или надежности (Stability / Reliability Testing) – один из видов тестирования ПО, целью которого является проверка работоспособности приложения при длительном тестировании со средним (ожидаемым) уровнем нагрузки.

Стресс-тестирование (англ. *Stress Testing*) – один из видов тестирования программного обеспечения, которое оценивает надежность и устойчивость системы в условиях превышения пределов нормального функционирования. Стресс-тестирование особенно необходимо для «критически важного» ПО, однако также используется и для остального ПО. Обычно стресс-тестирование лучше обнаруживает устойчивость, доступность и обработку исключений системой под большой нагрузкой, чем то, что считается корректным поведением в нормальных условиях.

Юзабилити-тестирование (проверка эргономичности) – исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, пользовательский интерфейс или устройство) для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы.

Юзабилити-тестирование основано на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов. Процесс организуется следующим образом. При испытании продукта (в нашем случае – программного) пользователю предлагают в «лабораторных» условиях решить основные задачи, для выполнения которых этот продукт разрабатывался, и просят высказывать во время выполнения этих тестов свои замечания. Процесс тестирования фиксируется в протоколе (логе) и/или на аудио- и видеоустройства – с целью последующего более детального анализа.

Если проверка эргономичности выявляет какие-либо трудности (например, сложности в понимании инструкций, выполнении действий или интерпретации

ответов системы), то разработчики должны доработать продукт и повторить тестирование.

Наблюдение за тем, *как* люди взаимодействуют с продуктом, нередко позволяет найти для него более оптимальные решения. Если при тестировании используется модератор, то его задача – держать респондента сфокусированным на задачах (но при этом не «помогать» ему решать эти задачи).

Основную трудность после проведения процедуры проверки эргономичности нередко представляют большие объемы и беспорядочность полученных данных. Поэтому для последующего анализа важно зафиксировать:

1. Речь модератора и респондента;
2. Выражение лица респондента (снимается на видеокамеру);
3. Изображение экрана компьютера, с которым работает респондент;
4. Различные события, происходящие на компьютере, связанные с действиями пользователя:
   – Перемещение курсора и нажатия на клавиши мыши;
   – Использование клавиатуры;
   – Переходы между экранами (браузера или другой программы).

Все эти потоки данных должны быть синхронизированы по тайм-кодам, чтобы при анализе их можно было бы соотносить между собой.

Наряду с модератором в тестировании нередко участвуют наблюдатели. По мере обнаружения проблем они делают свои заметки о ходе тестирования так, чтобы после можно было синхронизировать их с основной записью. В итоге каждый значимый фрагмент записи теста оказывается прокомментирован в заметках наблюдателя. В идеале ведущий (т.е. модератор) представляет разработчика, наблюдатели – заказчика (например издателя, дистрибьютора), а испытатели – конечного пользователя (например покупателя).

Кроме вышеизложенного существует еще один подход к проверке эргономичности: для решения задачи предложенной пользователю разрабатывается «идеальный» сценарий решения этой задачи. Как правило, это

сценарий, на который ориентировался разработчик. При выполнении задачи пользователями регистрируются их отклонения от задуманного сценария для последующего анализа. После нескольких итераций доработки программы и последующего тестирования можно получить удовлетворительный с точки зрения пользователя интерфейс.

Тестирование безопасности – оценка уязвимости программного обеспечения к различным атакам и попыткам проникновения. Под проникновением понимается широкий диапазон действий: попытки хакеров проникнуть в систему из спортивного интереса, месть рассерженных служащих, взлом мошенниками для незаконной наживы. Тестирование безопасности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

В ходе тестирования безопасности испытатель играет роль взломщика. Ему разрешено все:

- попытки узнать пароль с помощью внешних средств;
- атака системы с помощью специальных утилит, анализирующих защиты;
- подавление, ошеломление системы (в надежде, что она откажется обслуживать других клиентов);
- целенаправленное введение ошибок в надежде проникнуть в систему в ходе восстановления;
- просмотр несекретных данных в надежде найти ключ для входа в систему.

При неограниченном времени и ресурсах хорошее тестирование безопасности взломает любую систему. Задача проектировщика системы – сделать цену проникновения более высокой, чем цена получаемой в результате информации.

Тестирование совместимости – вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении. Окружение может включать в себя следующие элементы:

- Аппаратная платформа;

– Сетевые устройства;
– Периферия (принтеры, CD/DVD-приводы, веб-камеры и пр.);
– Операционная система (Unix, Windows, MacOS, ...)
– Базы данных (Oracle, MS SQL, MySQL, ...)
– Системное программное обеспечение.

По степени знания системы различают следующие виды тестирования:

– Тестирование черного ящика;
– Тестирование белого ящика.

Тестирование черного ящика – метод тестирования функционального поведения объекта (программы, системы) с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве тестируемого объекта, выбор тестов основан на технических требованиях и их спецификаций.

В этом методе программа рассматривается как черный ящик. Целью тестирования ставится выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Для обнаружения всех ошибок в программе необходимо выполнить исчерпывающее тестирование, то есть тестирование на всевозможных наборах данных.

Для большинства программ такое невозможно, поэтому применяют разумное тестирование, при котором тестирование программы ограничивается небольшим подмножеством всевозможных наборов данных. При этом необходимо выбирать наиболее подходящие подмножества, подмножества с наивысшей вероятностью обнаружения ошибок.

Принципы тестирования по методу «черного ящика»:

– Эквивалентное разбиение;
– Анализ граничных условий;
– Анализ причинно-следственных связей;
– Предположение об ошибке.

Эквивалентное разбиение. Основу метода составляют два положения:

Исходные данные необходимо разбить на конечное число классов эквивалентности. В одном классе эквивалентности содержатся такие тесты, что, если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку.

Каждый тест должен включать, по возможности, максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

Разработка тестов этим методом осуществляется в два этапа: выделение классов эквивалентности и построение теста.

Классы эквивалентности выделяются путем выбора каждого входного условия, которые берутся с помощью технического задания или спецификации и разбиваются на две и более группы.

Выделение классов эквивалентности является эвристическим способом, однако существует ряд правил:

1. Если входное условие описывает область значений, например «Целое число принимает значение от 0 до 999», то существует один правильный класс эквивалентности и два неправильных.
2. Если входное условие описывает число значений, например «Число строк во входном файле лежит в интервале (1..6)», то также существует один правильный класс и два неправильных.
3. Если входное условие описывает множество входных значений, то определяется количество правильных классов, равное количеству элементов в множестве входных значений. Если входное условие описывает ситуацию «должно быть», например «Первый символ должен быть заглавным», тогда один класс правильный и один неправильный.
4. Если есть основание считать, что элементы внутри одного класса эквивалентности могут программой трактоваться по-разному, необходимо разбить данный класс на подклассы. На этом шаге

тестирующий на основе таблицы должен составить тесты, покрывающие собой все правильные и неправильные классы эквивалентности. При этом составитель должен минимизировать общее число тестов.

Анализ граничных значений. Граничные условия – это ситуации, возникающие на высших и нижних границах входных классов эквивалентности.

Анализ граничных значений отличается от эквивалентного разбиения следующим:

Выбор любого элемента в классе эквивалентности в качестве представительного осуществляется таким образом, чтобы проверить тестом каждую границу этого класса.

При разработке тестов рассматриваются не только входные значения (пространство входов), но и выходные (пространство выходов).

Метод требует определенной степени творчества и специализации в рассматриваемой задаче. Существуют несколько правил:

1. Построить тесты с неправильными входными данными для ситуации незначительного выхода за границы области значений. Если входные значения должны быть в интервале [-1.0 .. +1.0], проверяем −1.0, 1.0, −1.000001, 1.000001.
2. Обязательно писать тесты для минимальной и максимальной границы диапазона.
3. Использовать первые два правила для каждого из входных значений (использовать пункт 2 для всех выходных значений).
4. Если вход и выход программы представляет упорядоченное множество, сосредоточить внимание на первом и последнем элементах списка.

Анализ граничных значений, если он применен правильно, позволяет обнаружить большое число ошибок. Однако определение этих границ для каждой

задачи может являться отдельной трудной задачей. Также этот метод не проверяет комбинации входных значений.

Анализ причинно-следственных связей.

Этапы построения теста:

1. Спецификация разбивается на рабочие участки.
2. В спецификации определяются множество причин и следствий. Под причиной понимается отдельное входное условие или класс эквивалентности. Следствие представляет собой выходное условие или преобразование системы. Здесь каждой причине и следствию присваивается номер.
3. На основе анализа семантического (смыслового) содержания спецификации строится таблица истинности, в которой последовательно перебираются всевозможные комбинации причин и определяются следствия для каждой комбинации причин.

Таблица снабжается примечаниями, задающими ограничения и описывающими комбинации, которые невозможны. Недостатком этого подхода является плохое исследование граничных условий.

Предположение об ошибке

Тестировщик с большим опытом выискивает ошибки без всяких методов, но при этом он подсознательно использует метод предположения об ошибке. Данный метод в значительной степени основан на интуиции. Основная идея метода состоит в том, чтобы составить список, который перечисляет возможные ошибки и ситуации, в которых эти ошибки могли проявиться. Потом на основе списка составляются тесты.

Тестирование белого ящика – техника тестирования, также называемая техникой тестирования управляемая логикой программы, позволяет проверить внутреннюю структуру программы. Исходя из этой стратегии тестировщик получает тестовые данные путем анализа логики работы программы.

Метод тестирования белого ящика основан на следующих принципах:

– *покрытие операторов* – каждая ли строка исходного кода была выполнена и протестирована;
– *покрытие условий* – каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
– *покрытие путей* – все ли возможные пути через заданную часть кода были выполнены и протестированы;
– *покрытие функций* – каждая ли функция программы была выполнена;
– *покрытие вход/выход* – все ли вызовы функций и возвраты из них были выполнены.

Статическое и динамическое тестирование

Динамическое тестирование – код исполняется (см. предыдущие виды тестирования)

Статическое тестирование – код не исполняется. анализ программы происходит на основе исходного кода, который вычитывается вручную, либо анализируется специальными инструментами.

Регрессионное тестирование. Регрессионное тестирование программного обеспечения направлено на обнаружение ошибок в ранее протестированных участках исходного кода.

Цель – выявить ошибки, появляющиеся после внесения изменений в программ. Ошибки при которых перестает работать то, что должно было продолжать работать, – называют *регрессионными ошибками*

Регрессионные тесты подтверждают, что сделанные изменения не повлияли на работоспособность остальной функциональности приложения. Регрессионное тестирование может выполняться как вручную, так и средствами автоматизации тестирования.

Из опыта разработки ПО известно, что повторное появление одних и тех же ошибок – случай достаточно частый. Иногда это происходит из-за слабой техники управления версиями или по причине человеческой ошибки при работе с системой

управления версиями. Но настолько же часто решение проблемы бывает «недолго живущим»: после следующего изменения в программе решение перестает работать. И наконец, при переписывании какой-либо части кода часто всплывают те же ошибки, что были в предыдущей реализации.

Модули, методы, функции нижнего уровня, который вызывается из многих точек кода, – распространенное место возникновения регрессионных ошибок.

Хотя регрессионное тестирование может быть выполнено и вручную, но чаще всего это делается с помощью специализированных программ, позволяющих выполнять все регрессионные тесты автоматически. В некоторых проектах даже используются инструменты для автоматического прогона регрессионных тестов через заданный интервал времени. Обычно это выполняется после каждой удачной компиляции (в небольших проектах) либо каждую ночь или каждую неделю.

Также регрессионное тестирование должно выполняться после рефакторинга исходного кода.

Покрытие кода.

Покрытие кода – мера, используемая при тестировании программного обеспечения. Она показывает процент, насколько исходный код программы был протестирован. Техника покрытия кода была одной из первых методик, изобретенных для систематического тестирования ПО. Первое упоминание покрытия кода в публикациях появилось в 1963 году.

Критерии оценки покрытия кода.

Существует несколько различных способов измерения покрытия, основные из них:

- Покрытие операторов – каждая ли строка исходного кода была выполнена и протестирована?
- Покрытие условий – каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована?
- Покрытие путей – все ли возможные пути через заданную часть кода были выполнены и протестированы?

– Покрытие функций – каждая ли функция программы была выполнена
– Покрытие вход/выход – все ли вызовы функций и возвраты из них были выполнены

Для программ с особыми требованиями к безопасности часто требуется продемонстрировать, что тестами достигается 100 % покрытие для одного из критериев. Некоторые из приведенных критериев покрытия связаны между собой; например, покрытие путей включает в себя и покрытие условий и покрытие операторов. Покрытие операторов не включает покрытие условий, как показывает этот код на Си:

```
printf("this is ");
if (bar < 1)
{
  printf("not ");
}
printf("a positive integer");
```

Если здесь bar = −1, то покрытие операторов будет полным, а покрытие условий – нет, так как случай несоблюдения условия в операторе if – не покрыт. Полное покрытие путей обычно невозможно. Фрагмент кода, имеющий n условий содержит *2n* путей; конструкция цикла порождает бесконечное количество путей. Некоторые пути в программе могут быть не достигнуты из-за того, что в тестовых данных отсутствовали такие, которые могли привести к выполнению этих путей. Не существует универсального алгоритма, который решал бы проблему недостижимых путей (этот алгоритм можно было бы использовать для решения проблемы останова). На практике для достижения покрытия путей используется следующий подход: выделяются классы путей (например, к одному классу можно отнести пути отличающиеся только количеством итераций в одном и том же цикле), 100 % покрытие достигнуто, если покрыты все классы путей (класс считается покрытым, если покрыт хотя бы один путь из него).

Покрытие кода, по своей сути, является тестированием методом белого ящика. Тестируемое ПО собирается со специальными настройками или библиотеками и/или запускается в особом окружении, в результате чего для каждой используемой (выполняемой) функции программы определяется местонахождение этой функции в исходном коде. Этот процесс позволяет разработчикам и специалистам по обеспечению качества определить части системы, которые, при нормальной работе, используются очень редко или никогда не используются (такие как код обработки ошибок и т.п.). Это позволяет сориентировать тестировщиков на тестирование наиболее важных режимов.

Практическое применение

Обычно исходный код снабжается тестами, которые регулярно выполняются. Полученный отчет анализируется с целью выявить невыполнявшиеся области кода, набор тестов обновляется, пишутся тесты для непокрытых областей. Цель состоит в том, чтобы получить набор тестов для регрессионного тестирования, тщательно проверяющих весь исходный код.

Степень покрытия кода обычно выражают в виде процента. Например, «мы протестировали 67 % кода». Смысл этой фразы зависит от того какой критерий был использован. Например, 67 % покрытия путей – это лучший результат чем 67 % покрытия операторов. Вопрос о связи значения покрытия кода и качеством тестового набора еще до конца не решен.

Тестировщики могут использовать результаты теста покрытия кода для разработки тестов или тестовых данных, которые расширят покрытие кода на важные функции.

Как правило, инструменты и библиотеки, используемые для получения покрытия кода, требуют значительных затрат производительности и/или памяти, недопустимых при нормальном функционировании ПО. Поэтому они могут использоваться только в лабораторных условиях

8.5. Работа с ошибками

8.5.1. Классификация ошибок

Международный стандарт ANSI/IEEE–729–83 разделяет все ошибки в разработке программ на следующие:

Ошибка (error) – причиной которой являются изъяны (flaw) в операторах программы или в технологическом процессе ее разработки, что приводит к неправильной интерпретации исходной информации, а следовательно и к неверному решению.

Дефект (fault) в программе является следствием ошибок разработчика на любом из этапов разработки и может содержаться в исходных или проектных спецификациях, текстах кодов программ, эксплуатационной документация и т.п. Дефект обнаруживается в процессе выполнения программы.

Отказ (failure) – это отклонение программы от функционирования или невозможность программы выполнять функции, определенные требованиями и ограничениями и рассматривается как событие, способствующее переходу программы в неработоспособное состояние из–за ошибок, скрытых в ней дефектов или сбоев в среде функционирования.

Документирование результатов тестирования в соответствии с действующим стандартом ANSI/IEEE 829, включает описание:

– задач, назначение и содержание программной системы, а также описание функций соответственно требованиям заказчика;

– технологии разработки системы;

– планов тестирования различных объектов, необходимых ресурсов, соответствующих специалистов для проведения тестирования и технологических способов;

– тестов, контрольных примеров, критериев и ограничений оценки результатов программного продукта, а также процесса тестирования;

– учета процесса тестирования, составление отчетов об аномальных событиях, отказах и дефектах в итоговом документе системы.

8.5.2. Средства контроля ошибок

В развитых системах разработки присутствуют специальные средства для работы с ошибками в процессе тестирования программной системы. Основные функции такой системы сводятся к:

- фиксация данных об обнаруженной ошибке;
- прослеживание цикла исправления ошибки;
- контроль состояния ошибок;
- выпуск отчетов о текущем состоянии и динамике работы с ошибками.

Как правило, описание ошибки в системе контроля ошибок имеет следующие основные атрибуты:

- ответственного за ее проверку тестировщика, который ее нашел и который проверяет, что исправления, сделанные разработчиком, действительно устраняют ошибку;
- ответственного за ее исправление – разработчика, которому ошибка отправляется на исправление;
- состояние, например, ошибка найдена, ошибка исправлена, ошибка закрыта,
- ошибка вновь проявилась и т.д.

Использование этих систем давно стало общей практикой в разработке ПО, наравне со средствами контроля версий и многими иными инструментами. Они включают в себя:

- базу данных для хранения ошибок;
- интерфейс к этой базе данных для внесения новых ошибок и задания их;
- многочисленных атрибутов, для просмотра ошибок на основе различных;
- фильтров – например, все найденные ошибки за последний месяц, все ошибки;
- за которые отвечает данный разработчик и т.д.;

– сетевой доступ, так как проекты все чаще оказываются распределенными;
– программный интерфейс для возможностей программной интеграции таких;
– систем с другим ПО, поддерживающим разработку ПО (например, со средствами непрерывной интеграции – они могут автоматически вносить в базу данных найденные при автоматическом прогоне тестов ошибки).

ЛИТЕРАТУРА

1. Руководство к своду знаний по программной инженерии. The Guide to the Software Engineering Body of Knowledge, SWEBOK, IEEE Computer Society Professional Practices Committee, 2004. [Электронный документ] – Режим доступа: http://www.swebok.org

2. Орлик С. Руководство к своду знаний по программной инженерии [Электронный документ] – Режим доступа: http://swebok.sorlik.ru/download.html

3. Соммервилл, Иан. Инженерия программного обеспечения. 6-е издание. Издательский дом “Вильямс”, 2002. – 624 с.

4. Липаев В.В. Программная инженерия. Учебник. – М.: ТЕИС, 2006. – 608 с.

5. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения. – Учебник, М.: 2006 – 320 с.

6. Терехов А.Н. Что такое программная инженерия. Журнал «Программная инженерия», №1, 2010 г. с. 40-45.

7. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

8. Rubinstein D., «Standish Group Report: There's Less Development Chaos Today». 2007. [Электронный ресурс] – режим доступа: http://www. sdtimes.com/content/article.aspx?ArticleID=30247.

9. ГОСТ Р ИСО/МЭК 12207-99. ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ. Информационная технология. ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ.

10. Майерс Г., Баджетт Е., Сандлер К.. Искусство тестирования программ, 3-е издание. – М.: «Диалектика», 2012. – 272 с

11. Портал «Тестирование и качество ПО»/ [Электронный документ] – Режим доступа: http://software-testing.ru/.

12. Портал об автоматизированном тестировании. [Электронный документ] – Режим доступа: http://automated-testing.info.

9. УПРАВЛЕНИЕ ПРОЦЕССАМИ РАЗРАБОТКИ ПО В MS VISUAL STUDIO TEAM SYSTEM

Цель лекции – дать студентами представление концепции Microsoft Solution Framework и средствах разработки ПО в Visual Studio Team System.

В результате изучения материалов лекции студент должен:

- знать методологию Microsoft Solutions Framework;
- знать особенности организации разработки ПО в MSF;
- знать структуру VSTS и Team Foundation Server;
- иметь представление о реализации жизненного цикла разработки в VSTS.

В качестве ознакомления с принципами применения MSF предполагается проведение тренинга по тестированию программы в процессе решения модельной задачи на лабораторных занятиях.

В лекции будут рассматриваться следующие основные темы:

– Методология Microsoft Solutions Framework (MSF);
– Основные дисциплины и модели MSF;
– Среда разработки Visual Studio Team System 2010
– Управление работами в VSTS.

9.1. Методология Microsoft Solutions Framework (MSF)

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

MSF представляет собой согласованный набор концепций, моделей и правил.

В 1994 году, стремясь достичь максимальной отдачи от IT-проектов, Microsoft выпустила в свет пакет руководств по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий. Эти знания базируются на опыте, полученном Microsoft при работе над большими проектами по разработке и сопровождению программного обеспечения, опыте консультантов Microsoft и лучшем из того, что накопила на данный момент

IT-индустрия. Все это представлено в виде двух взаимосвязанных и хорошо дополняющих друг друга областей знаний: *Microsoft Solutions Framework* (MSF) и *Microsoft Operations Framework* (MOF).

Существуют различные прикладные варианты MSF, разработанные Microsoft. Наиболее популярными являются:

– методика внедрения решений в области Управления проектами;
– методика управления IT-проектами на базе методологий MSF и Agile.

Первая версия MSF появилась в 1994 году. Версия MSF 4.0 была представлена в 2005 году. В данной версии произошло разделение методологии на два направления: MSF for Agile Software Development и MSF for CMMI Process Improvement.

При этом если версии до 3.x были именно методологиями (там были изложены принципы, MSF свободно распространялась в виде Word-документов, которые были также переведены на русский язык), то теперь MSF превратилась в шаблоны процесса для VSTS. Эти шаблоны имеют описание в виде html-документов и определяют типы ролей, их ответственности, действия в рамках этих ответственностей, а также все входные и выходные артефакты этих деятельностей и другие формализованные атрибуты процесса разработки. Кроме этого «человеческого» описания MSF for Agile и MSF for CMMI имеют XML-настройки, которые позволяют в точности следовать предложенным выше описаниям, используя VSTS. При этом на процесс накладываются достаточно жесткие ограничения, деятельность разработчиков сопровождается набором автоматических действий – все это задано в шаблонах. Данные шаблоны можно частично использовать (например, без некоторых ролей), а также изменять (VSTS предоставляет обширные средства настойки шаблонов)

В настоящее время действующей версией является MSF 5.0, которая реализована в двух главных продуктах Microsoft, предназначенных для разработки, – Visual Studio Team System 2010 (VSTS) и Team Foundation Server 2010 (TSF). В сентябре 2012 г. выпущены новые релизы этих продуктов – VSTS 2012 и TFS 2012.

MSF предлагает проверенные методики для планирования, проектирования, разработки и внедрения успешных IT-решений. Благодаря своей гибкости, масштабируемости и отсутствию жестких инструкций MSF способен удовлетворить нужды организации или проектной группы любого размера. Методология MSF состоит из принципов, моделей и дисциплин по управлению персоналом, процессами, технологическими элементами и связанными со всеми этими факторами вопросами, характерными для большинства проектов.

MSF состоит из двух моделей и трех дисциплин. Они подробно описаны в 5 основополагающих статьях (whitepapers).

MSF содержит:

– модели:

- модель проектной группы;
- модель процессов;

– дисциплины:

- дисциплина управление проектами;
- дисциплина управление рисками;
- дисциплина управление подготовкой.

9.2. Основные принципы MSF

Единое видение проекта. Успех коллективной работы над проектом немыслим без наличия у членов проектной группы и заказчика единого видения (shared vision), т.е. четкого, и, самое главное, одинакового, понимания целей и задач проекта. Как проектная группа, так и заказчик изначально имеют собственные предположения о том, что должно быть достигнуто в ходе работы над проектом. Лишь наличие единого видения способно внести ясность и обеспечить движение всех заинтересованных в проекте сторон к общей цели. Формирование единого видения и последующее следование ему являются столь важными, что модель процессов MSF выделяет для этой цели специальную фазу – «Выработка концепции», которая заканчивается соответствующей вехой.

Гибкость – готовность к переменам. Традиционная дисциплина управления проектами и каскадная модель исходят из того, что все требования могут быть четко сформулированы в начале работы над проектом, и далее они не будут существенно изменяться. В противоположность этому MSF основывается на принципе непрерывной изменяемости условий проекта при неизменной эффективности управленческой деятельности.

Концентрация на бизнес-приоритетах. Независимо от того, нацелен ли разрабатываемый продукт на организации или индивидуумов, он должен удовлетворить определенные нужды потребителей и принести в некоторой форме выгоду или отдачу. В отношении индивидуумов это может означать, например, эмоциональное удовлетворение – как в случае компьютерных игр. Что же касается организаций, то неизменным целевым фактором продукта является бизнес-отдача (business value). Обычно продукт не может приносить отдачу до того, как он полностью внедрен. Поэтому модель процессов MSF включает в свой жизненный цикл не только разработку продукта, но и его внедрение.

Поощрение свободного общения. Исторически многие организации строили свою деятельность на основе сведения информированности сотрудников к минимуму, необходимому для исполнения работы (need-to-know). Зачастую такой подход приводит к недоразумениям и снижает шансы команды на достижение успеха. Модель процессов MSF предполагает открытый и честный обмен информацией как внутри команды, так и с ключевыми заинтересованными лицами. Свободный обмен информацией не только сокращает риск возникновения недоразумений, недопонимания и неоправданных затрат, но и обеспечивает максимальный вклад всех участников проектной группы в снижение существующей в проекте неопределенности. По этой причине модель процессов MSF предлагает проведение анализа хода работы над проектом в определенных временных точках. Документирование результатов делает ясным прогресс, достигнутый в работе над проектом – как для проектной команды, так и для заказчика и других заинтересованных в проекте сторон.

9.3. Модель проектной группы MSF

Модель проектной группы MSF (MSF Team Model) описывает подход Майкрософт к организации работающего над проектом персонала и его деятельности в целях максимизации успешности проекта. Данная модель определяет ролевые кластеры, их области компетенции и зоны ответственности, а также рекомендации членам проектной группы, позволяющие им успешно осуществить свою миссию по воплощению проекта в жизнь.

Модель проектной группы MSF возникла в результате осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп.

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

Ниже описываются основные принципы, ключевые идеи и испытанные методики MSF в применении к модели проектной группы

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

1. Распределение ответственности при фиксации отчетности
2. Наделяйте членов команды полномочиями
3. Концентрируйтесь на бизнес-приоритетах
4. Единое видение проекта
5. Проявляйте гибкость – будьте готовы к переменам
6. Поощряйте свободное общение

В проектную группу входят такие **ролевые кластеры**:

– управление программой

– управление продуктом

– разработка
– тестирование
– управление релизом
– удовлетворение потребителя.

9.4. Ролевые кластеры

MSF основан на постулате о семи качественных целях, достижение которых определяет успешность проекта. Эти цели обуславливают модель проектной группы и образуют **ролевые кластеры** (или просто **роли**) в проекте. каждом ролевом кластере может присутствовать по одному или несколько специалистов, некоторые роли можно соединять одному участнику проекта. Каждый ролевой кластер представляет уникальную точку зрения на проект, и в то же время никто из членов проектной группы в одиночку не в состоянии успешно представлять все возможные взгляды, отражающие качественно различные цели. Для разрешения этой дилеммы команда соратников (команда равных, team of peers), работающая над проектом, должна иметь четкую форму отчетности перед заинтересованными сторонами (stakeholders) при распределенной ответственности за достижение общего успеха. В MSF следующие ролевые кластеры (часто их называют ролями)

1. Управление продуктом (product management). Основная задача этого ролевого кластера – обеспечить, чтобы заказчик остался довольным в результате выполнения проекта. Этот ролевой кластер действует по отношению к проектной группе как представитель заказчика и зачастую формируется из сотрудников организации-заказчика. Он представляет бизнес-сторону проекта и обеспечивает его согласованность со стратегическими целями заказчика. В него же входит контроль за полным пониманием интересов бизнеса при принятии ключевых проектных решений.

2. Управление программой (program management) обеспечивает управленческие функции – отслеживание планов и их выполнение,

ответственность за бюджет, ресурсы проекта, разрешение проблем и трудностей процесса, создание условий, при которых команда может работать эффективно, испытывая минимум бюрократических преград.

3. Разработка (development). Этот ролевой кластер занимается, собственно, программированием ПО.

4. Тестировани*е* (test) – отвечает за тестирование ПО.

5. Удовлетворение потребителя (user experience). Дизайн удобного пользовательского интерфейса и обеспечение удобства эксплуатации ПО (эргономики), обучение пользователей работе с ПО, создание пользовательской документации.

6. Управление выпуском (release management). Непосредственно ответственен за беспрепятственное внедрение проекта и его функционирование, берет на себя связь между разработкой решения, его внедрением и последующим сопровождением, обеспечивая информированность членов проектной группы о последствиях их решений.

7. Архитектура (Architecture).7 Организация и выполнение высокоуровневого проектирования решения, создание функциональной спецификации ПО и управление этой спецификацией в процессе разработки, определение рамок проекта и ключевых компромиссных решений.

Наличие шести ролевых кластеров не означает, что количество членов команды должно быть кратным шести – один человек может совмещать несколько ролей и наоборот, ролевой кластер может состоять из нескольких лиц в зависимости от размера проекта, его сложности и профессиональных навыков, требуемых для реализации всех областей компетенции кластера. Минимальный коллектив по MSF может состоять всего из трех человек. Модель не требует назначения отдельного сотрудника на каждый ролевой кластер. Смысл состоит в том, что в команде должны быть представлены все шесть качественных целей. Обычно, выделение как минимум одного человека на каждый ролевой кластер

обеспечивает полноценное внимание к интересам каждой из ролей, но это экономически оправданно не для всех проектов. Зачастую члены проектной группы могут объединять роли.

В малых проектных группах объединение ролей является необходимым. При этом должны соблюдаться два принципа:

1. Роль команды разработчиков не может быть объединена ни с какой другой ролью.
2. Избежание сочетания ролей, имеющих предопределенные конфликты интересов.

9.5. Масштабирование команды MSF

Наличие 7 ролевых кластеров не означает, что команда должна состоять строго из 7 человек. Один сотрудник может объединять несколько ролей. При этом некоторые роли нельзя объединять. В таблице ниже представлены рекомендации MSF относительно совмещения ролей в рамках одним членом команды. «+» означает, что совмещение возможно, «+-» - что совмещение возможно, но нежелательно, «-» означает, что совмещение не рекомендуется. MSF не предоставляет конкретных рецептов управления проектами и не содержит объяснений разнообразных методов работы, которые применяют опытные менеджеры.

Принципы MSF формируют такой **подход к управлению проектами**, при котором:

– ответственность за управление проектом распределенная между лидерами ролевых кластеров внутри команды – каждый член проектной группы отвечает за общий успех проекта и качество создаваемого продукта;

– профессиональные менеджеры выступают в качестве консультантов и наставников команды, а не выполняют функции контроля над ней – в эффективно работающей команде каждый ее член имеет необходимые полномочия для выполнения своих обязанностей и уверен, что получит от коллег все необходимое.

Как следует из вышесказанного, одна из характерных особенностей MSF – отсутствие должности менеджера проекта.

Модель проектной группы MSF предлагает разбиение больших команд (более 10 человек) на малые **многопрофильные группы направлений** (feature teams). Эти малые коллективы работают параллельно, регулярно синхронизируя свои усилия. Кроме того, когда ролевому кластеру требуется много ресурсов, формируются т. н. функциональные группы (functional teams), которые затем объединяются в ролевые кластеры.

Использование ролевых кластеров не подразумевает и не навязывает никакой специальной структуры организации или обязательных должностей.

9.6. Модель процессов MSF

Модель процессов MSF (MSF process model) представляет общую методологию разработки и внедрения IT решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов. Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной и спиральной.

Процесс MSF ориентирован на «**вехи**» – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: «Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?», «В достаточной ли степени готов план действий?», «Соответствует ли продукт утвержденной спецификации?», «Удовлетворяет ли решение нужды заказчика?» и т. д.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду.

Модель процессов MSF тесно связана с базовыми принципами MSF, рассмотренными выше. Вообще говоря, тремя особенностями модели процессов MSF являются:

– Подход, основанный на фазах и вехах;
– Итеративный подход;
– Интегрированный подход к созданию и внедрению решений.

Модель процессов включает такие основные фазы процесса разработки:

– Выработка концепции (Envisioning);
– Планирование (Planning);
– Разработка (Developing);
– Стабилизация (Stabilizing);
– Внедрение (Deploying).

Кроме этого существует большое количество *промежуточных вех*, которые показывают достижение в ходе проекта определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки.

Для каждой фазы модели процессов MSF определяет:

– что (какие артефакты) является результатом этой фазы;
– над чем работает каждый из ролевых кластеров на этой фазе.

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. «Живые» документы (living documents) должны

изменяться по мере эволюции проекта вместе с изменениями требований к конечному продукту. В рамках MSF предлагается ряд шаблонов стандартных документов, которые являются артефактами каждой стадии разработки продукта и могут быть использованы для планирования и контроля процесса разработки.

Решение не представляет бизнес-ценности, пока оно не внедрено. Именно по этой причине модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение – вплоть до момента, когда решение начинает давать отдачу.

9.7. Дисциплина управления проектами

Проект – ограниченная временными рамками деятельность, цель которой состоит в создании уникального продукта или услуги. Управление проектами (project management) – это область знаний, навыков, инструментария и приемов, используемых для достижения целей проектов в рамках согласованных параметров качества, бюджета, сроков и прочих ограничений.

Хорошо известна взаимозависимость между ресурсами проекта (людскими и финансовыми), его календарным графиком (временем) и реализуемыми возможностями (рамками). Эти три переменные образуют так называемый «треугольник компромиссов». Нахождение верного баланса между ресурсами, временем разработки и возможностями – ключевой момент в построении решения, должным образом отвечающего нуждам заказчика.

Другое весьма полезное средство для управления проектными компромиссами – матрица компромиссов проекта. Она отражает достигнутое на ранних этапах проекта соглашение между проектной группой и заказчиком о выборе приоритетов в возможных в будущем компромиссных решениях. В определенных случаях из этой приоритезации могут делаться исключения, но в целом следование ей облегчает достижение соглашений по спорным вопросам.

Для иллюстрации использования матрицы компромиссов Майкрософт предлагает использовать следующее предложение (вместо пропущенных слов могут быть вставлены «календарный график», «ресурсы» и «функциональность»):

«Зафиксировав _____, мы согласовываем _____ и принимаем результирующий _____».

Как уже говорилось выше, в MSF нет роли «менеджер проекта». Деятельность по управлению проектом распределяется между лидерами групп и ролевым кластером «Управление программой».

9.8. Дисциплина управления рисками

Управление рисками – это одна из ключевых дисциплин Microsoft Solutions Framework. MSF видит в изменениях и возникающей из-за них неопределенности неотъемлемые части жизненного цикла информационных технологий. Дисциплина управления рисками в MSF (MSF risk management discipline) отстаивает превентивный подход к работе с рисками в условиях такой неопределенности, непрерывное оценивание рисков и использование информации о рисках в рамках процесса принятия решений на протяжении всего жизненного цикла проекта. Данная дисциплина предлагает принципы, идеи и рекомендации, подкрепленные описанием пошагового процесса для успешного активного управления рисками. Этот процесс включает в себя выявление и анализ рисков; планирование и реализацию стратегий по их профилактике и смягчению возможных последствий; отслеживание состояния рисков и извлечение уроков из обретенного опыта. Девиз MSF – мы не боремся с рисками – мы ими управляем.

9.9. Дисциплина управления подготовкой

Управление подготовкой – это также одна из ключевых дисциплин Microsoft Solutions Framework (MSF). Она посвящена управлению знаниями, профессиональными умениями и способностями, необходимыми для планирования, создания и сопровождения успешных решений. Дисциплина управления подготовкой MSF описывает фундаментальные принципы MSF и дает рекомендации по применению превентивного подхода к управлению знаниями на протяжении всего жизненного цикла информационных технологий. Эта дисциплина также рассматривает планирование процесса управления подготовкой. Будучи подкрепленной испытанными практическими методиками, дисциплина

управления подготовкой предоставляет проектным группам и отдельным специалистам базу для осуществления этого процесса.

9.10. Среда разработки Visual Studio Team System 2010

Visual Studio Team System – единая среда для разработки программных приложений, организации совместной работы над проектами, инструментов для тестирования и отладки разрабатываемых программ, а также построения отчетов.

Ядром VSTS является средства обеспечения жизненного цикла *элементов работы*(work items) – некоторых дискретных характеристик проекта, вокруг которых организуется вся работа команды (см. рис. 9.1). Вот примеры элементов работ:

- task – конкретная задача, которую необходимо выполнить в проекте;
- bag – ошибка, которая найдена, ждет своего исправления, исправляется, заново проверяется;
- risk – риск проекта, у которого тоже может быть разное состояние; как правило, за рисками их состояниями следят менеджеры проектов.

Каждый элемент работ имеет набор различных состояний, перечень событий, который могут изменять эти состояния, а также ответственное лицо. Элемент работы используется для оперативного управления проектом следующим образом. Каждый из участников команды видит связанные с ним элементы работ и после выполнения соответствующей работы меняет их состояния, а, возможно и ответственное лицо. Например, программист исправил ошибку и после этого для элемента работ, обозначающего эту ошибку, он меняет состояние (например, Fixed) и ответственного – соответствующего тестеровщика, чтобы последний протестировал изменения кода.

Microsoft ориентирует компании, использующие Team System, использовать Microsoft Solutions Framework (метамодель, описывающую бизнес-процессы и процессы инженерии программного обеспечения) для упрощения реализации эффективного процесса разработки программного обеспечения. Team System поддерживает две основных концепции разработки ПО: гибкую (Agile) и CMMI.

Также предусмотрено добавление других фреймворков для поддержки иных концепций и методик.

9.10.1. Основные компоненты VSTS

Visual Studio Team System состоит из нескольких продуктов, которые можно разделить на серверные и клиентские приложения:

– Среда разработки Visual Studio Ultimate/Premium представляет собой полный набор средств разработки для создания веб-приложений ASP.NET, XML (веб-службы), настольных приложений и мобильных приложений. Visual Basic, Visual C# и Visual C++ используют единую интегрированную среду разработки (IDE), которая позволяет совместно использовать средства и упрощает создание решений на базе нескольких языков.)

– Team Foundation Server – это платформа для совместной работы, лежащая в основе решения Visual Studio для управления жизненным циклом приложений. Сервер Team Foundation Server предоставляет базовые услуги, такие как управление версиями, отслеживание рабочих элементов и ошибок, автоматизация построения, а также хранилище данных.

– Microsoft Test Manager. Менеджер тестирования можно использовать для определения объема работ по тестированию и создания и выполнения тестов вручную. Менеджер тестирования также интегрируется с базой данных рабочих элементов в Team Foundation для создания и отслеживания ошибок, обнаруженных во время тестирования.

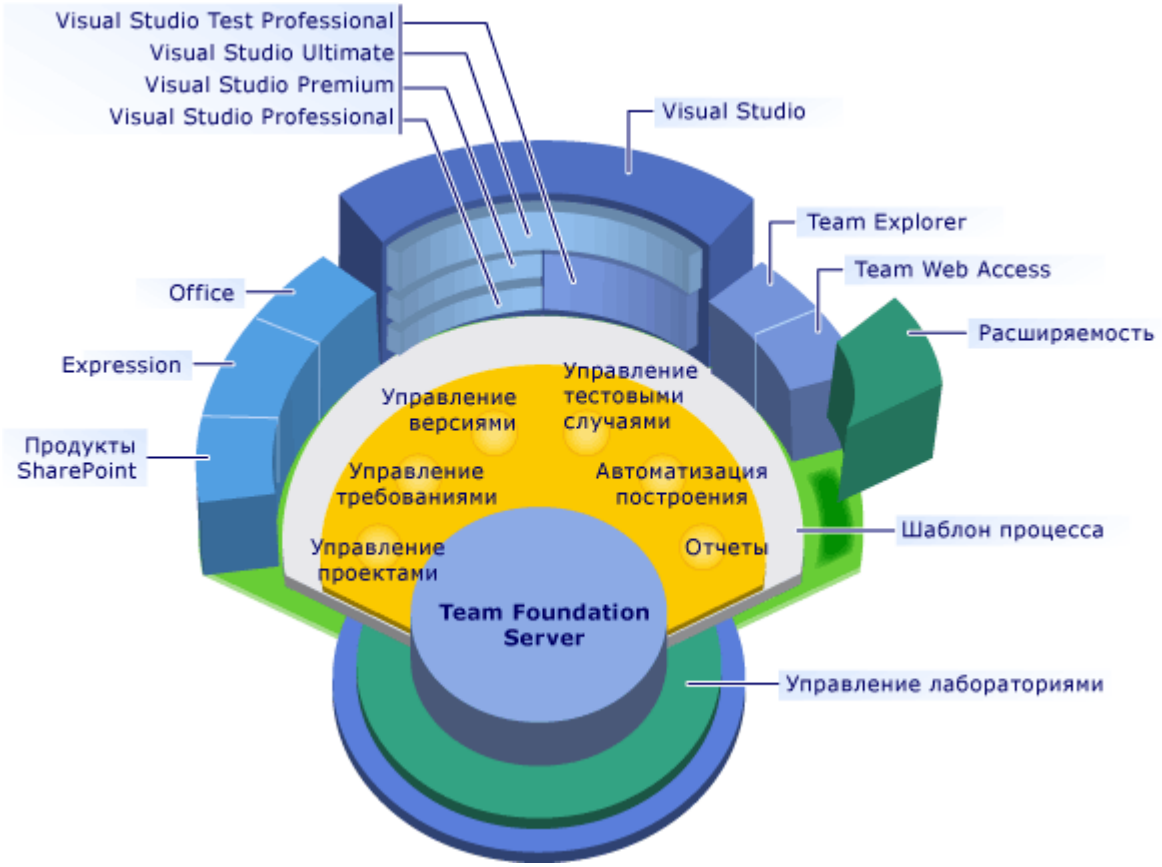


Рис. 9.1. VSTS

Эти средства можно использовать для достижения следующих результатов:

- **Планирование и отслеживание проектов**. Ввод в действие процессов и контроль их качество.
- **Проектирование функциональности** на базе существующих активов или "с нуля", используя архитектурные диаграммы для обмена наиболее важной информацией о программном обеспечении команды.
- **Создание кода**, тестирование модулей, выполнение отладки, анализ и профилирование приложение с помощью средств, интегрированных с жизненным циклом приложения. Использование системы управления версиями для управления исходным кодом и другими файлами.
- **Выполняйте построение приложения** с помощью встроенной системы построения, чтобы команда смогла проверить соответствие критериям качества и определить, какие требования реализованы в каждом построении.

– **Тестирование приложения** с помощью ручных или автоматических тестов, включая тесты производительности и нагрузочные тесты. Управление тестированием, чтобы команда постоянно имела представление о текущем качестве программного обеспечения.

Развертывание в виртуальных средах, чтобы получить возможность осуществлять более сложную разработку и тестирование.

В таблице 9.1 показано использование этих средств на этапах жизненного цикла приложения.

Таблица 9.1

Этап жизненного цикла приложения	Team Foundation Server	Microsoft Test Manager	Visual Studio Premium	Visual Studio Ultimate
Планирование и отслеживание	✓			
Проектирование				✓
Разработка	✓		✓	✓
Тестирование	✓	✓	✓	✓
Построение	✓			

Кроме того существуют дополнительные компоненты:

– Командный обозреватель устанавливается с каждой версией Microsoft Visual Studio 2010. Сред. Командный обозреватель можно использовать для доступа к функциям управления жизненным циклом приложений в составе Visual Studio, включая командные проекты, управление версиями, командные построения и управление проектами.
– Team Web Access – это настраиваемый веб-интерфейс, предоставляющий большинство функций, доступных в Сред. Командный обозреватель (но не все функции). Team Web Access также позволяет с помощью специального синтаксиса поиска быстро находить рабочие элементы.
– Microsoft Excel и Microsoft Project. Team Foundation интегрируется с Excel и Project, чтобы облегчить управление проектом и сохранить синхронизацию с базой данных рабочих элементов. Командный обозреватель приложения Excel и Project могут взаимодействовать с Team Foundation Server. Список рабочих элементов отображается на вкладке Рабочая группа в Excel. В приложении Project появляются

панель инструментов "Рабочая группа" и меню "Рабочая группа", и рабочие элементы отображаются в плане проекта в виде задач.

Для хранения всех данных проекта (репозитарий проекта) TFS использует MS SQL Server 2010/2008 (рис. 9.2). При отсутствии отдельного SQL-сервера, при установке TFS может быть развернута компактная версия SQL Server Express Edition. Кроме того, TFS интегрируется с MS SharePoint.

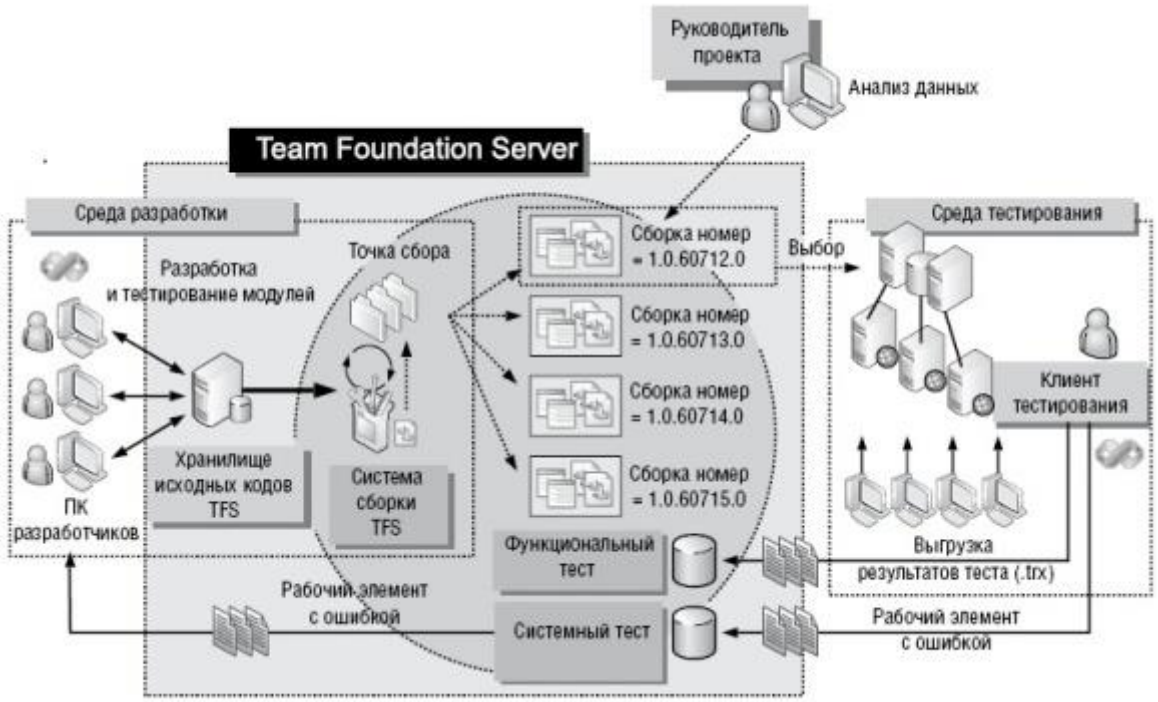


Рис. 9.2 Логический документооборот Team Foundation Server

Тестовая группа запускает результаты сборки в тестовой среде, проводя сочетание ручных и автоматических тестов. Результаты тестов хранятся в TFS и используются для организации обратной связи по вопросам качества сборки. Кроме того, тестовая группа может создавать рабочие элементы и ошибки (особый тип рабочего элемента), на которые группе разработчиков следует обратить внимание. Эти элементы позволяют группе тестирования отслеживать работу группы разработчиков.

9.10.2. Управление элементами работ в VSTS

Вернемся к элементам работ VSTS – ключевым дискретным характеристикам проекта, таким как задача (task), ошибка (bug), риск (risk) и т.д. Эти характеристики выделены в VSTS с целью конкретизировать объекты управления в проекте, сделать это управление сквозным в следующих смыслах:

- обеспечить доступ к одной и той же информации для разных участников (и ролей) в проекте; например, доступ к ошибкам для менеджеров, разработчиков и тестеров;
- прослеживать связи одних элементов с другими, например, изменений исходного кода и теми ошибками, для исправления которых эти изменения были сделаны.

Благодаря единой среде, включающей в себя средства поддержки различных видов элементов работы, в VSTS гораздо проще строить связи между элементами работы различного вида, отслеживать их изменения, чем при использовании отдельных продуктов поддержания процесса. Например, не нужно ждать момента, пока информация об ошибке или задаче будет перенесена из одной системы в другую. Ведь традиционно программные средства планирования (там, где определяются задачи), управления ошибками (там, где происходит учет ошибок), средства версионного контроля – это разные средства. Кроме того, в силу наличия единого информационного репозитория в VSTS возможны строгие ссылки на такие объекты, определенная сборка или тест, и получение, по соответствующему запросу, подробной информации по разным фильтрам, на разную глубину детализации. Возможно, также настроить автоматическую генерацию элементов работы, например, ошибок при неудачной автоматической сборке или при автоматическом прогоне тестов.

Элементы работы можно связывать с другими артефактами проекта - файлами с исходным кодом, сборками (как настройками, так и результатами), документами (по процессу, проектными, пользовательскими и др.), отчетами, которые могут также храниться в VSTS. Каждый элемент работы принадлежит определенному типу. Тип элемента работы определяет набор реквизитов.

Жизненный цикл элемента работы определяется двумя системными реквизитами: состоянием и причиной. Первый описывает текущее состояние элемента работы и определяет его текущую роль в процессе. Каждый тип элементов работы описывает допустимый набор состояний, например, «активный», «завершенный», «проверенный» и т.д. Кроме того, каждый тип элемента работы имеет описание переходов между своими состояниями состояний, причины, вызывающие эти переходы и действия, выполняемые в них. Причинами могут являться, например, «выполнен», «устарел», «отложен» и т.д. Переходы может осуществлять сама система TFS, автоматически, но в большинстве случае разработчик сам инициирует переход, внося в систему информацию о том, что выполнил задачу, исправил ошибку и так далее.

Таким образом, жизненный цикл элемента работы представляется ориентированным графом, нагруженным как по узлам, так и по дугам. Использование такого графа вместо нагруженного только по узлам, как например, часто можно встретить в системах управления ошибками, позволило резко сократить размер описания и повысить информативность.

9.10.3. Пример использования элемента работы task

Как правило, в начале проекта некоторый эксперт (как правило, системный архитектор, ведущий разработчик и т.д.) проводит анализ всей необходимой работы по проекту и разбивает ее на подзадачи, устанавливая ответственных, сроки и т.д. Эти подзадачи с соответствующими атрибутами и являются элементами работы типа task. Затем менеджер проекта, с учетом списка всех задач и их взаимосвязей, строит календарный план. На этом этапе менеджеру могут оказаться полезными средства Project и Microsoft Excel – он пользуется ими на основе соответствующих мостов, имеющихся в TFS.

Далее разработчики начинают реализовывать соответствующие задачи. После того, как было внесено последнее изменение и задача выполнена, разработчик переводит элемент работы в состояние Resolved и информация о нем войдет в следующий отчет по автоматической сборке. При обнаружении ошибок

реализации тестер создаст новый элемент работы типа Bug и проставит ему связь с исходной задачей. Если же тестер обнаружит, что функциональность реализована не в полном объеме, то он может решить перевести задачу обратно в состояние Active. Если же реализованная функциональность достаточно стабильна, а имеющиеся ошибки не являются критичными, тестер переводит задачу в состояние Closed.

За всем этим процессом наблюдает менеджер проекта, используя как запросы на элементы работы, так и средства интеграции с офисными приложениями, а также средства построения отчетов. Таким образом, он получает возможность максимально оперативно реагировать на возникающие нештатные ситуации, отставания от плана и возникающие дополнительные незапланированные работы.

Доступ к элементам работы члены команд получают через Team Explorer, являющийся компонентом Visual Studio Ultimate (Premium).

При редактировании и создании элементов работы учитываются все те правила, заданные в шаблоне процесса, где определен данный тип элементов работы. Это выражается в том, что соответствующие поля формы свойств элемента работы допускают или запрещают редактирование, позволяют выбор значений только из определенного списка и т.д.

Выделенные элементы работы можно экспортировать в пакеты Microsoft Project, Word. Изменения, произведенные с элементами работ, выполненными в этих пакетах, можно затем загрузить обратно в TFS.

Элементы работы при планировании. Не сложно заметить, что такая важная роль как менеджер проекта, не получила собственного издания Visual Studio. Связано это с тем, что основная платформа Visual Studio плохо приспособлена для задач, которые приходится решать этой роли. Гораздо более удачно для этого подходят офисные приложения – Microsoft Excel и Microsoft Project. Поэтому для более полного вовлечения менеджера в информационное пространство проекта Team System предоставляет специальные мосты.

После того, как разработка плана в Project выполнена, запланированные действия вносятся в VSTS. Кроме того, сам файл с планом можно сохранить на диске или портале SharePoint. При этом информация о связи с сервером TFS так же сохранится.

Далее исполнители увидят их в списках своих задач и начнут их исполнять. Изменения попадают в систему контроля версиями и в этот момент они связываются с данными об изменении элементов работы.

Текущие данные об элементах работ используются в TFS для выпуска отчетов о ходе процесса и его характеристиках.

Кроме работ, связанных разработкой кода, VSTS обеспечивает:

- настраиваемое управление исходным кодом в системе контроля версий;
- сборку решения в специальном сервисе MS Bild, который обеспечивает непрерывную сборку по заданному сценарию;
- задание или автоматическое построение тестов, выполнение модульных и регрессионных тестов в Test Manager.

Более подробно знакомиться с основными компонетами VSTS предполагается на лабораторном практикуме.

ЛИТЕРАТУРА

1. Сазерленд Д. Принципы и значение гибкой разработки. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/ru-u/library/vstudio/dd997578(v=vs.100) .aspx.
2. Учебное руководство. Начало работы с Microsoft Visual Studio Team Foundation Server 2010. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/vstudio/dd286491%28v=vs .100%29.aspx.
3. Scrum. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/en-s/library/dd997796%28v =VS.100%29. Aspx.
4. ГОСТ Р ИСО/МЭК 12207-99. ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ. Информационная технология. ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

1. О ПРАКТИКУМЕ

1.1 Общая информация

Практикум предназначен для практического усвоения ряда положений программной инженерии и использует в качестве инструментария продукт Microsoft Visual Studio Team System, поддерживающий командную разработку ПО. В рамках практикума предполагается освоение планирования, конфигурационного управления (средства контроля версий и управление сборками), автоматического тестирования и командной работы в рамках формально определенного процесса разработки. Такой выбор обуславливается с, одной стороны, важностью этих практик в реальном промышленном производстве, с другой стороны, возможностями, предоставляемыми продуктом MS VSTS.

Задачей практикума является не столько обучение продукту MS VSTS, сколько использование его в качестве основы для практического освоения программной инженерии. Достоинство продукта MS VSTS заключается в его доступности для целей обучения.

1.2 Требований к техническому оснащению

Занятия по данному курсу должны проводиться в компьютерных классах, удовлетворяющих следующим условиям.

Все компьютеры должны находиться в домене Active Directory под управлением доменного контроллера версии 2003.

Все учащиеся должны иметь логин и пароль для входа в этот домен.

В рамках этого домена должен быть развернут TeamFoundationServer2010 со всеми необходимыми компонентами (Microsoft SQL 2005, Sharepoint Server 3.0, Internet Information Server 6.1).

В рамках этого домена должен быть развернут Team Foundation Build и настроен для работы с Team Foundation Server.

На всех компьютерах класса должна быть установлена VisualStudio Team

System Ultimate 2010.

На сервере TFS должен быть импортирован шаблон процесса разработки Scrum for TFS.

Каждый участник практикума должен иметь по компьютеру. Также целесообразно, чтобы в классе была доска, проектор.

2.3 Организация процесса

Занятия предполагается проводить в форме ролевой игры. Группа разбивается на команды в 5-6 человек. Каждая команда в процессе выполнения заданий моделирует работу реального коллектива разработчиков.

В качестве методологии процесса разработки ПО используется Scrum. Эта методология очень проста и хорошо проецируется на учебные команды. Кроме того, Scrum получил в последнее время значительное распространение в мире и в России. Наконец, имеется и доступен Scrum-шаблон для MS VSTS.

Команды должны самоорганизоваться в соответствии с методологией SCRUM [1, 2, 3]. В качестве Product Owner на первом занятии выступает вся команда, задачей которой является подготовка projectbacklog по одной из предложенных преподавателем тем. На последующих занятияхкоманда организуется по правилам технологии SCRUM. В роли Scrum-мастера выступать один из ее членов. При этом роль Scrum-мастера может допускать широкие вариации по активностям и ответственностям – от простого слежения за временем и некоторых формальностей (именно Scrum-мастер будет выполнять некоторые общие, единые для всех действия, а остальные будут наблюдать за тем, как он это делает), до некоторых функций project manager. Вся группа учащихся будет являться scrum-командой, работающей над реализацией некоторого проекта в рамках одного sprint. Начальные требования к модельной задаче предоставляются в виде backlog, из которого студенты изготовляют sprint backlog.

2.4 Модельная задача

Данный практикум проводится на основе некоторой практической задачи, которую scrum-команда учащихся реализует в рамках практикума. Требования к

этой задаче следующие.

Она не должна быть очень сложной.

Задача должна быть распределенная, разные ее части, розданные разным участникам, должны быть зависимы друг от друга.

Задача должна хорошо подходить для написания модульных тестов.

Задача НЕ должна содержать сложного пользовательского интерфейса или других технических сложностей.

В качестве своего варианта предлагается реализацияпростого приложения «Калькулятор».

Варианты задач:

1. Арифметический калькулятор

2.Статистический калькулятор (минимум, максимум, среднее значение, дисперсия, среднеквадратичное отклонение для введенной последовательности чисел).

3. Калькулятор формул (арифметические выражения со скобками)

При выполнении задания 3 предлагается использовать учебные материалы [5, 6].

Команде необходимо реализовать не столько приложение в целом, сколько библиотеку классов, позволяющую легко реализовывать различные вариации с разным пользовательским интерфейсом или без оного.

На первом занятии студентам необходимо составить список пользовательских историй, описывающих необходимую функциональность задачи. Описание должно быть достаточно детальным, чтобы максимально точно определить структуру будущей системы.

2.5 Требования к студентам

Для полноценного участия в практикуме студенты должны владеть следующей информацией и практическими навыками.

Они должны быть знакомы с курсом «Методы и средствапрограммнойинженерии». Основной необходимый материал изложен в

лекциях № 4 и № 9. В качестве дополнительного материала для подготовки используется статьи [3, 4].Они должны владеть практическими навыками работы с языком C# и средой разработки MS Visual Studio (хотя бы по 1-2 чел. в каждой команде).

2.6. Обзор тем и задач

На подготовительном занятии группа делится на команды. Каждая команда выбирает одну из предложенных преподавателем задач и разрабатывает подробное ее описание и определяет необходимую функциональность. Результат работы проверяется и корректируется преподавателем.

Тема 1. При изучении первой темы (на первой серии занятий) студенты организуются как Scrum-команда. Они также знакомятся с условиями модельной задачи, разрабатывают пользовательские истории для задачи. Настраивают инфраструктуру TFS для будущей разработки (создают командный проект и распределяют права на работу с ним).

Тема 2. На второй серии занятий студенты практикуются в планировании работ на основе методологии Scrum, а также изучают способы использования системы отслеживания задач TFS. Студентам необходимо импортировать список пользовательских историй их файла Excel в TFS, а затем детально спланировать будущий спринт и распределить задачи.

Тема 3. Третья серия занятий предполагает основную работу по реализации решения модельной задачи. На занятиях этой серии студенты должны также освоитьсистему контроля версий TFS, ее интеграцией с системой отслеживания задач, а также попрактиковаться в создании ветвей и интеграции изменений.

Тема 4. На занятиях четвертой серии студенты практикуются в разработки модульных тестов средствами Visual Studio Team Developer. На этих занятиях студенты должны освоить средства автоматической генерации тестов, наполнить сгенерированные тесты содержимым, а также научится изменять конфигурацию запуска модульных тестов и считать тестовое покрытие.

Тема 5. Пятая серия занятий посвящена системе автоматических сборок TFS.

На этих занятиях студенты должны создать несколько определений для автоматической сборки (build definitions) в разных случаях – с тестами и без, с анализом кода и без и т.д. Кроме того, студенты должны настроить параметры непрерывной интеграции и рассылки уведомлений.

Тема 6. На заключительной, шестой, серии занятий студенты должны повести ретроспективный анализ выполненного Scrum sprint, выявить потенциальные способы оптимизации, а затем и применить их, используя средства настройки процесса разработки TFS. На этих занятиях студенты должны освоить изменение настроек системы отслеживания задач средствами Team Foundation Server Power Tools.

Тема 1. Знакомство и создание проекта

Целями данного занятия является следующее.

- Разделить студентов на scrum-команды, определить и обсудить Scrum-роли.
- Провести начальное знакомство с модельной задачей, которую предстоит реализовать. Разработать пользовательские истории.
- Обсудить открытые вопросы по модельной задаче с Prodcut Owner.
- Создать командный проект на TFS и добавить в него пользователей.

При подготовке к занятию студенты должны разработать:

1. Краткое описание основной концепции разрабатываемого приложения.
2. Список задач в формате product backlog.

После того, как команды получили представление о модельной задаче, им необходимо создать командный проект в MS VSTS и занести всех участников проекта в список пользователей.

Шаг 1. Создание проекта.

Создание командного проекта осуществляется лидером команды по следующему сценарию:

1. Открыть Visual Studio Team Suite и окно Team Explorer в ней (рис.1):

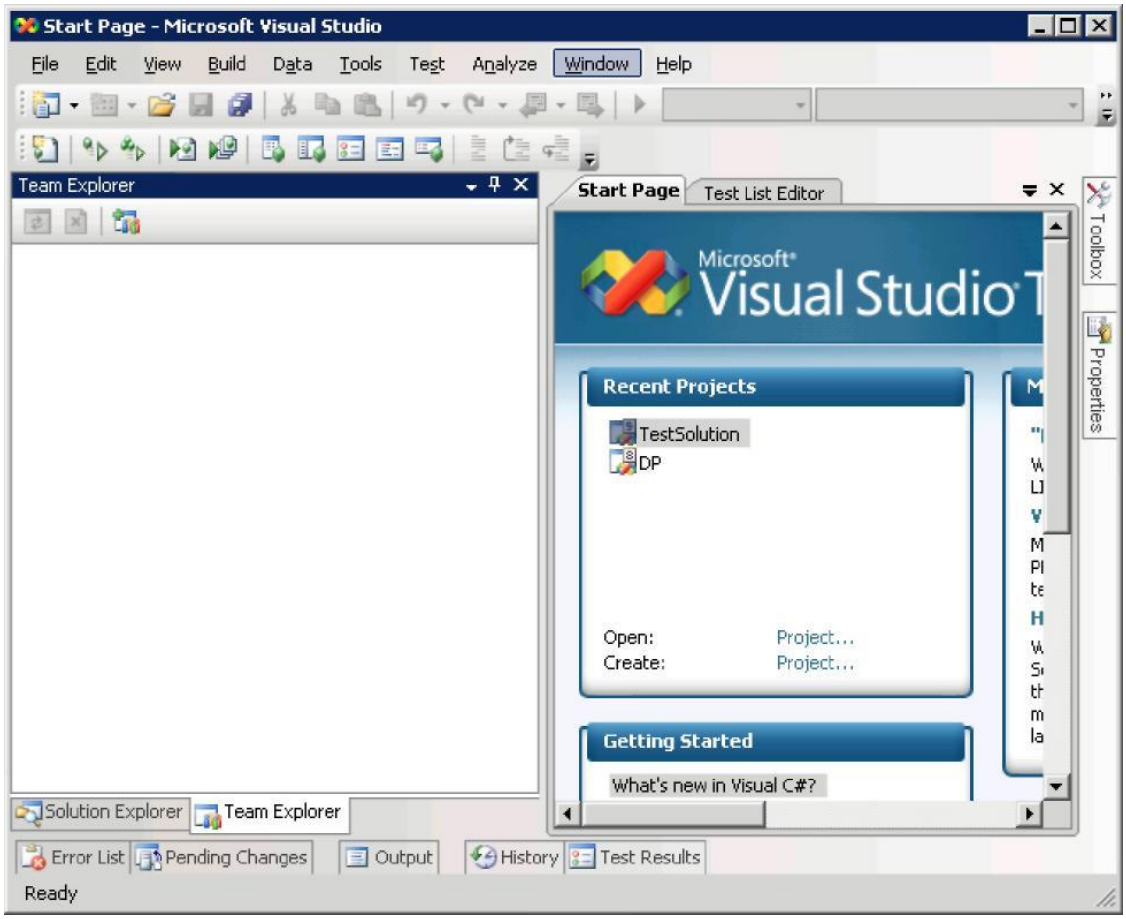


Рис. 1. Окно Team Explorer.

Нажать кнопку соединится с сервером .

Задать имя и порт сервера в открывшееся диалоге (рис. 2):

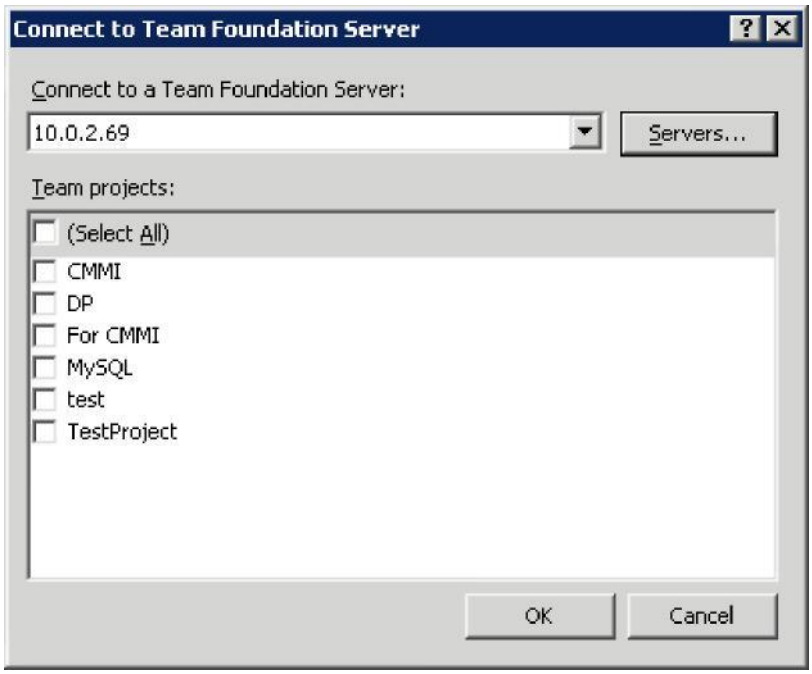


Рис. 2. Диалоговое окно.

4. После того, как соединение с сервером установлено, запустить процедуру создания проекта, используя соответствующую команду контекстного меню New Team Project (рис.3):

Рис. 3. Контекстное меню

5. В качестве имени проекта необходимо указать (рис. 4) TFSCourse<год>-<имя команды>:

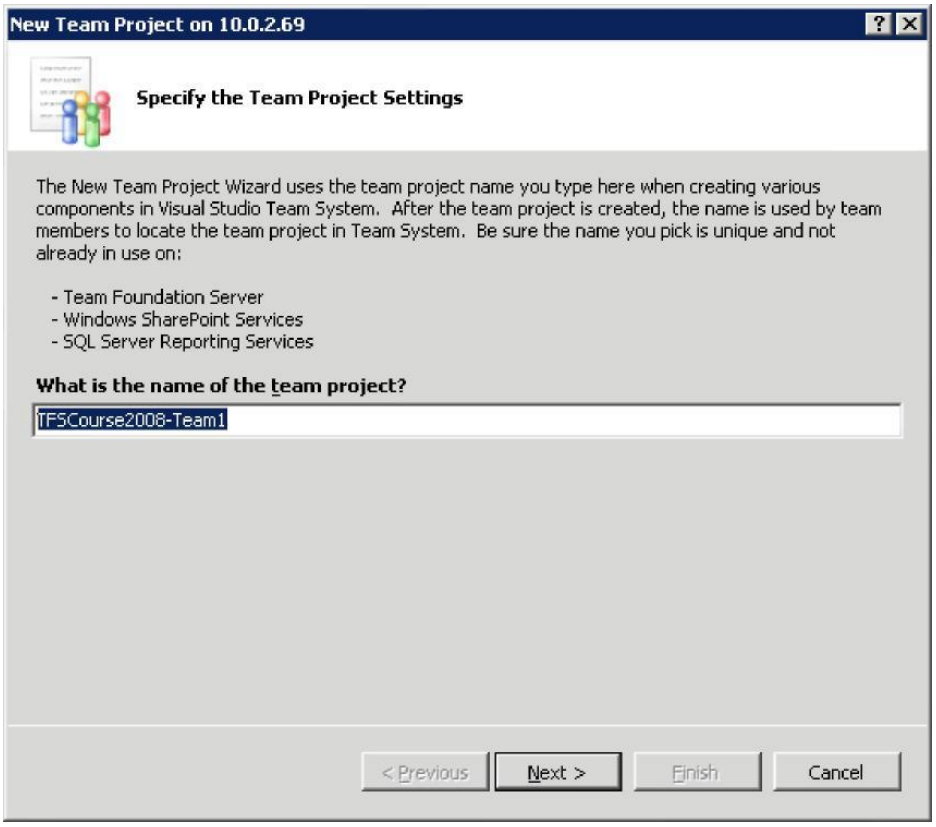


Рис. 4. Задание имени команды.

6. В качестве шаблона процесса разработки выбрать Scrum (рис. 5):

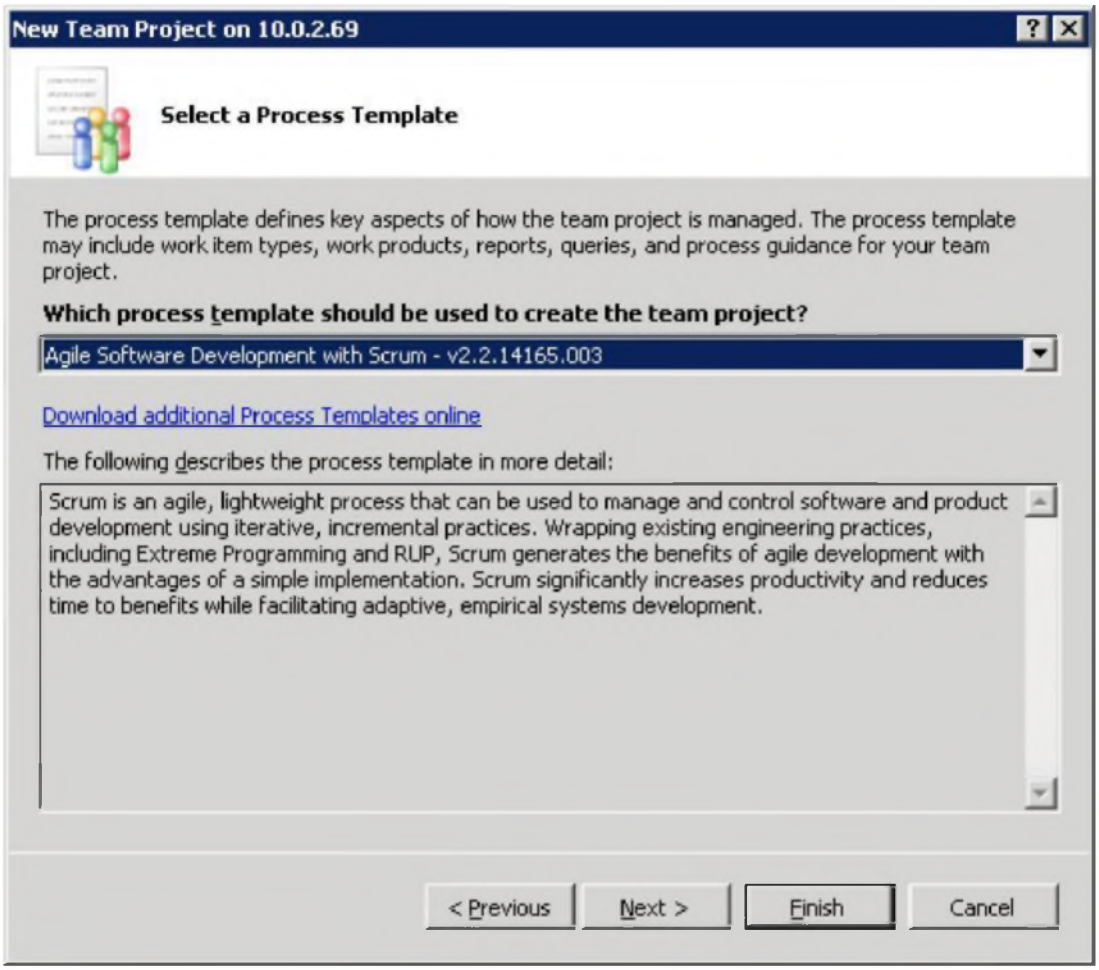


Рис.5 Шаблон процесса разработки.

7. Создать новый пустой раздел в системе контроля версий для данного проекта (рис. 6):

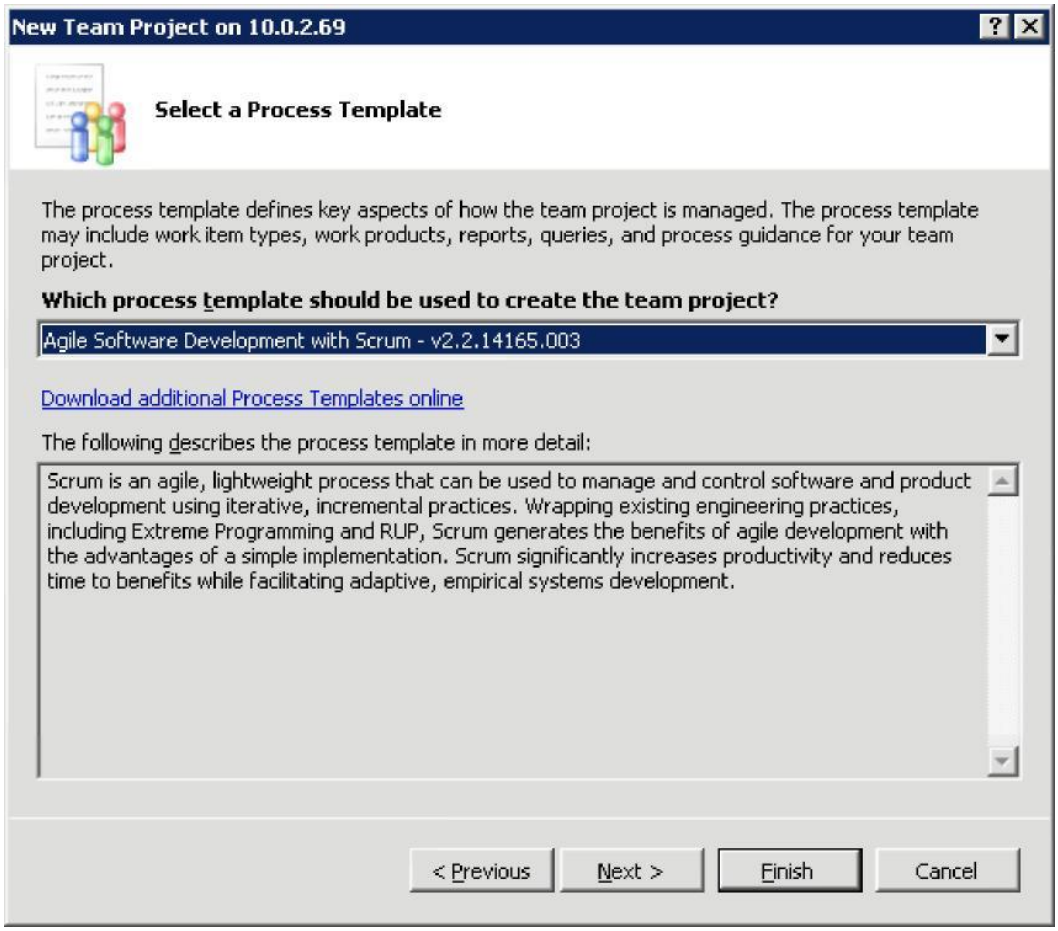


Рис. 6 Создание раздела.

8. После выбора всех настроек, создать проект.

Шаг 2. Настройка прав.

После того, как проект был создан лидеру необходимо выделить права остальным участникам команды для работы с этим проектом. Для этого ему нужно:

1. Выбрать в свойствах проекта раздел Group membership (рис.7):

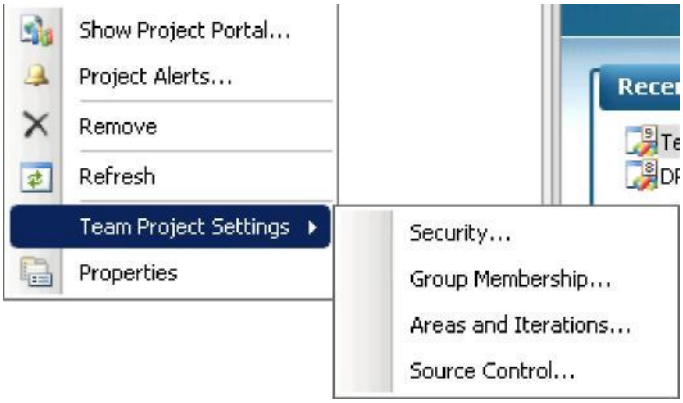


Рис.7. Свойства проекта.

2. В открывшемся диалоге (рис.8) включить всех участников в группы Readers и Contributors:

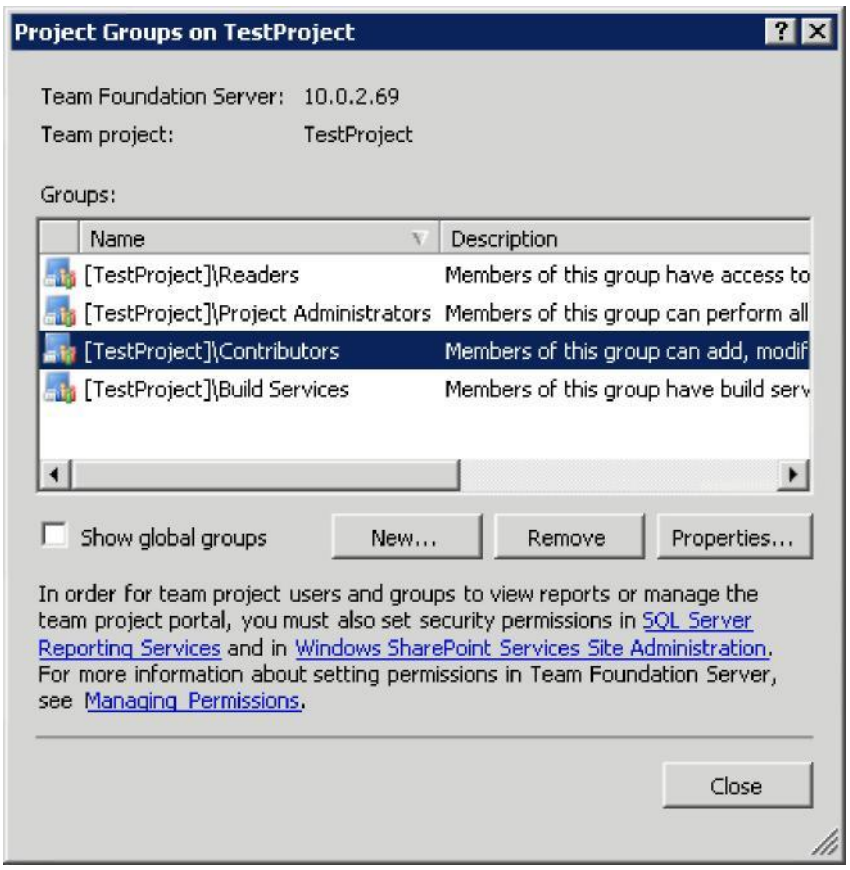


Рис. 8. Ввод участников команды

Шаг 3. Подключение проекта остальными участниками команды.

После того, как все получили права на работу с проектом, каждый участник команды должен на своей машине открыть VisualStudio и добавить соединение с этим проектом. Выполняется это аналогично пунктам 1-4 первого шага занятия.

Тема 2. Работа с системой отслеживания ошибок

Основной целью данного занятия является знакомство участников с системой отслеживания элементов работы.

- Создание элементов работы средствами Visual Studio и Team Explorer;
- Импорт и экспорт элементов работы из/в Mcrosoft Excel;
- Назначение ответственных за элементы работы;
- Отслеживание текущего статуса посредством отчетов.

В рамках данного занятия предполагается провести планирование работы команды по методологии Scrum. Команды уже провели предварительное знакомство с проектов, а на данном этапе от них требуется следующее.

1. Импортировать содержимое списка требований в TFS, используя средства импорта из Excel.
2. Подробно рассмотреть 10 наиболее приоритетных пользовательских историй.
3. Обсудить возникшие вопросы с хозяином продукта.
4. Провести детальное планирование и разбить пользовательские истории на мелкие подзадачи.
5. Распределить подзадачи среди участников проекта.
6. Отчитаться перед хозяином продукта о том, какие пользовательские истории были запланированы. При отчете использовать отчеты TFS.

Шаг 1. Импорт списка пользовательских историй.

Для того, чтобы загрузить пользовательские истории из Excel-файла, полученного командами на прошлом занятии нужно:

1. Выбрать команды Add work items with Microsoft Excel в контекстном меню (рис.9):

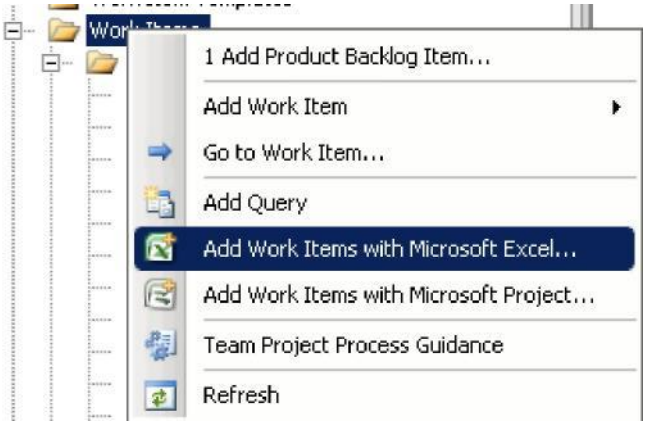


Рис. 9. Контекстное меню.

2. В открывшемся окне Excel (рис.10) настроить колонки таким образом, чтобы они совпадали порядком и смыслом с колонками в исходном Excel документе (для этого можно использовать команды Choose columns):

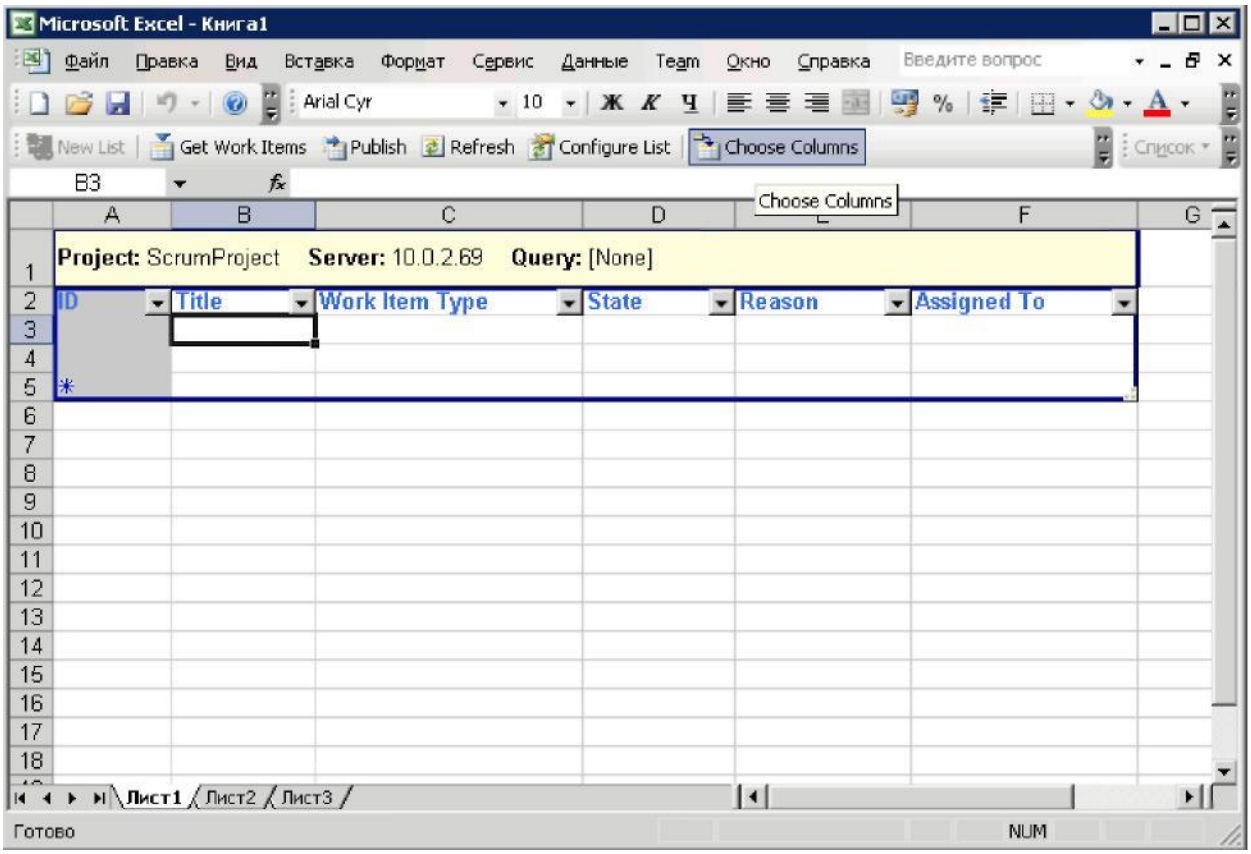


Рис. 10. Окно Excel.

После того, как колонки настроены, скопировать значения из исходного

Excel в редактируемый.

Показать колонку с именем Work Item Type и задать для все строчек значение Product backlog item.

5. Нажать кнопку Publish.

6. Убедится, что при выполнении запроса All product back to items, видны все вновь загруженные пользовательские истории (рис. 11):

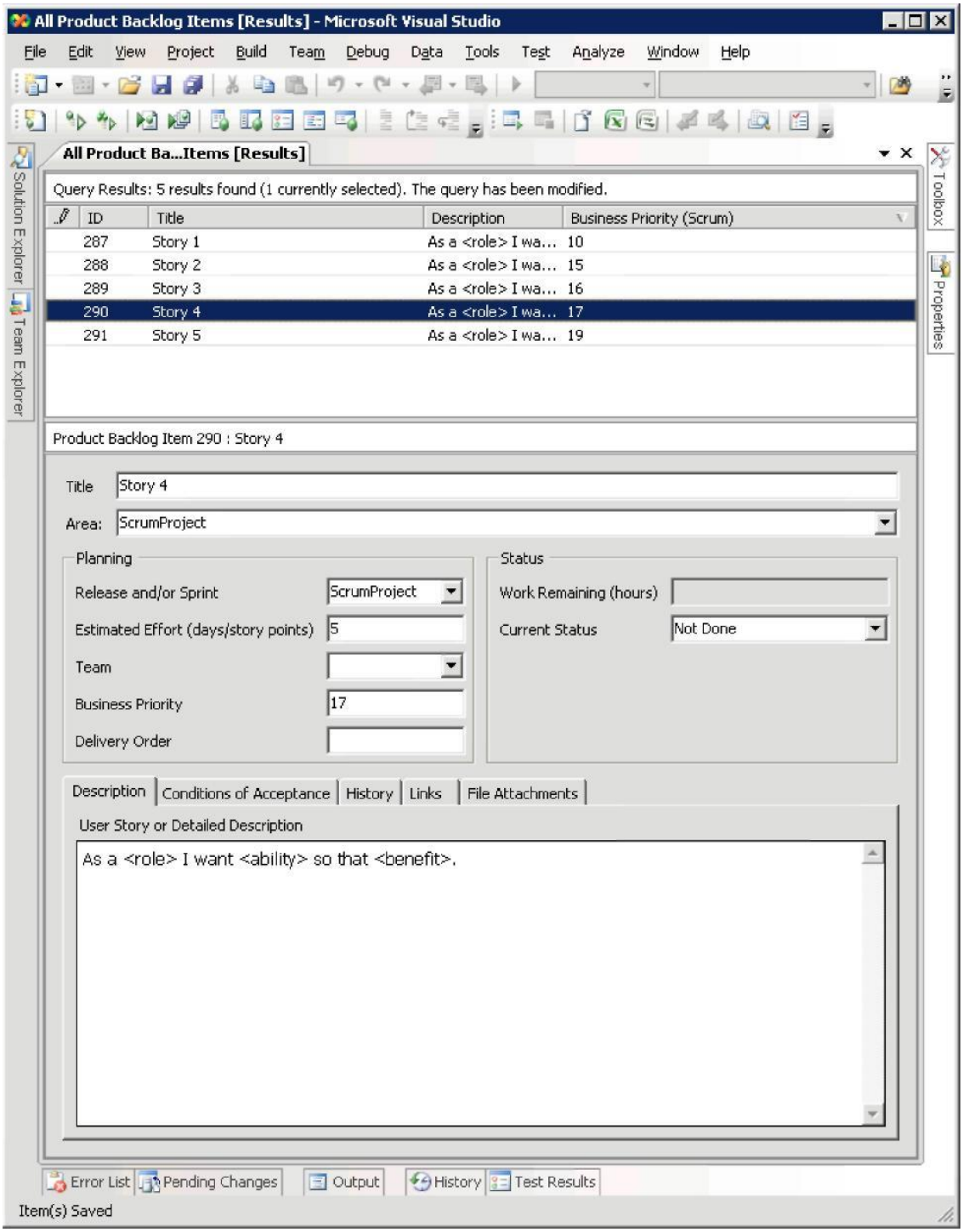


Рис. 11. Пользовательские истории.

Шаг 2. Создание sprint

После импорта списка пользовательских историй команда должна создать элемент работы, соответствующий начинающемуся sprint. Некоторое количество sprints уже создано по умолчанию при создании проекта (рис.12):

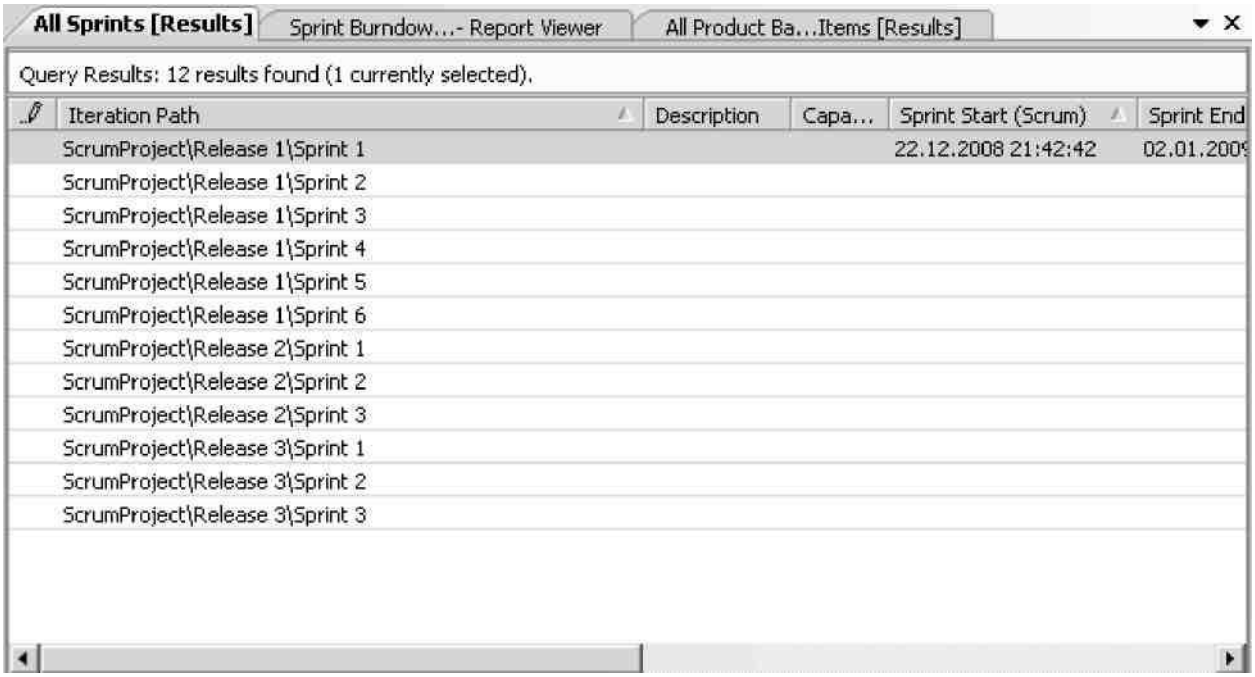


Рис.12. Окно пользовательских историй.

Для активации первого спринта ему необходимо установить дату начала, дату окончания и количество часов, которые команда может потратить в этом спринте.

Шаг 3. Формирование Sprint backlog

После обсуждения открытых вопросов по 10 наиболее приоритетным пользовательским историям команда должна приступить к планированию текущего sprint и формированию sprint backlog. Для этого ей необходимо рассмотреть список всех пользовательских историй и разбить его на список более мелких задач. При этом для каждой задачи необходимо создать элемент работы типа sprint back logitem и проставить следующие атрибуты:

В качестве sprint указать Release1/Sprint1.

Добавить связь с соответствующим элементом product backlog, а также со

всеми связанными элементами работы.

Установить Estimated efforts и Work remaining в соответствии с оценкой команды.

Задать ответственного за задачу (Owned By).

Выполнить все операции нужно средствами Visual Studio и Team Explorer.

После создания и распределения задач каждый член команды должен на своей машине убедиться, что выданные ему задачи отображаются в результатах запроса My Sprint Backlog Items.

Тема 3. Работа с системой контроля версий

Основной целью данного занятия является освоение системы контроля версий Team Foundation Server и ееинтеграции с системой отслеживания задач. Занятие предполагает выполнение следующих действий.

- Разработка кода модельной задачи средствами Visual Studio и внесение его в систему управления версиями;
- Проставление связей между вносимыми изменениями и элементами системы отслеживания задач;
- Создание параллельно поддерживаемых веток кода;
- Интеграция изменений, сделанных параллельно в одном файле или в разных ветках кода.

Шаг 1. Разработка кода.

Перед началом работы команде необходимо создать решение (solution) средствами Visual Studio, включив опцию Add to Source Control (рис.13):

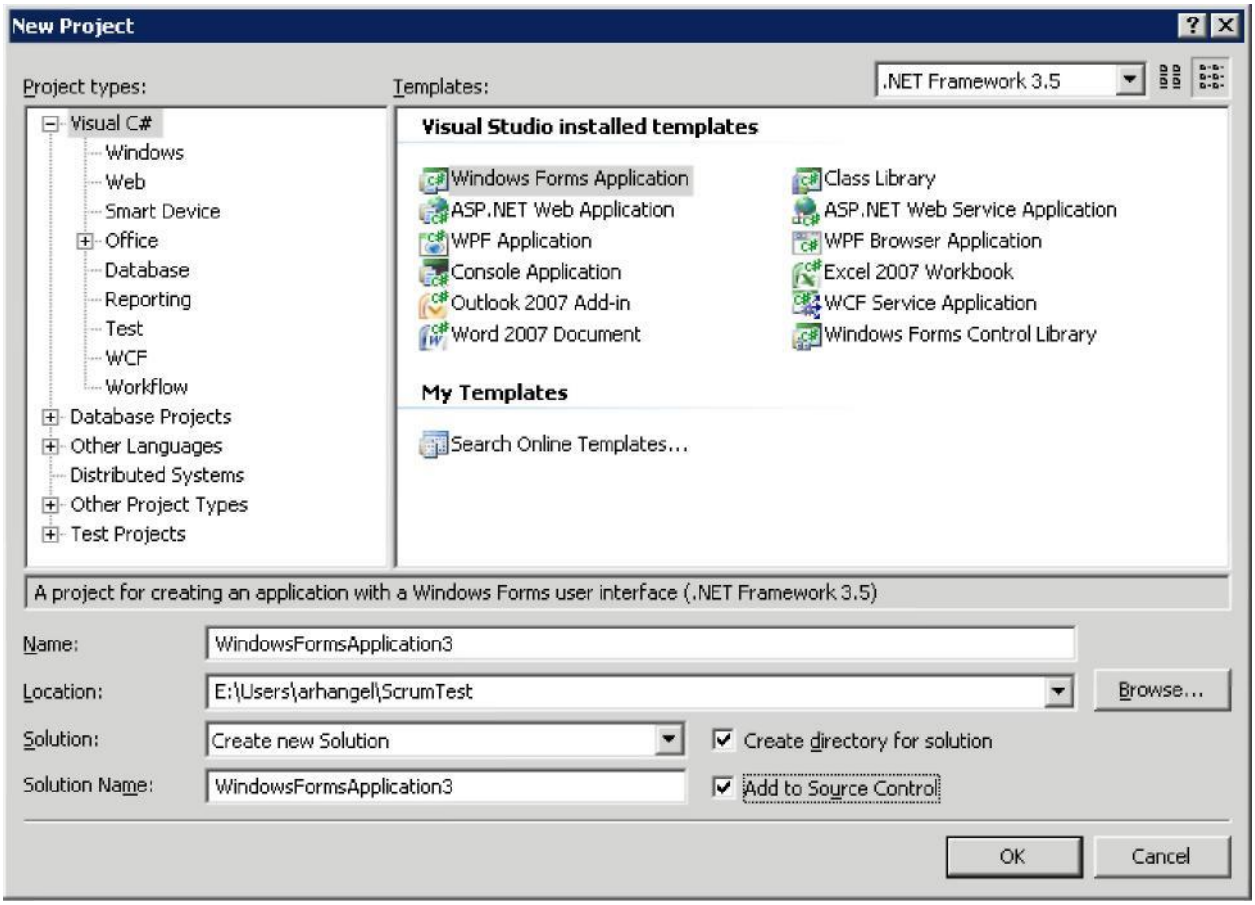


Рис. 13. Создание решения.

В открывшемся после создания проекта окне необходимо выбрать командный проект, в систему контроля версий которого нужно добавить данное решение (рис. 14):

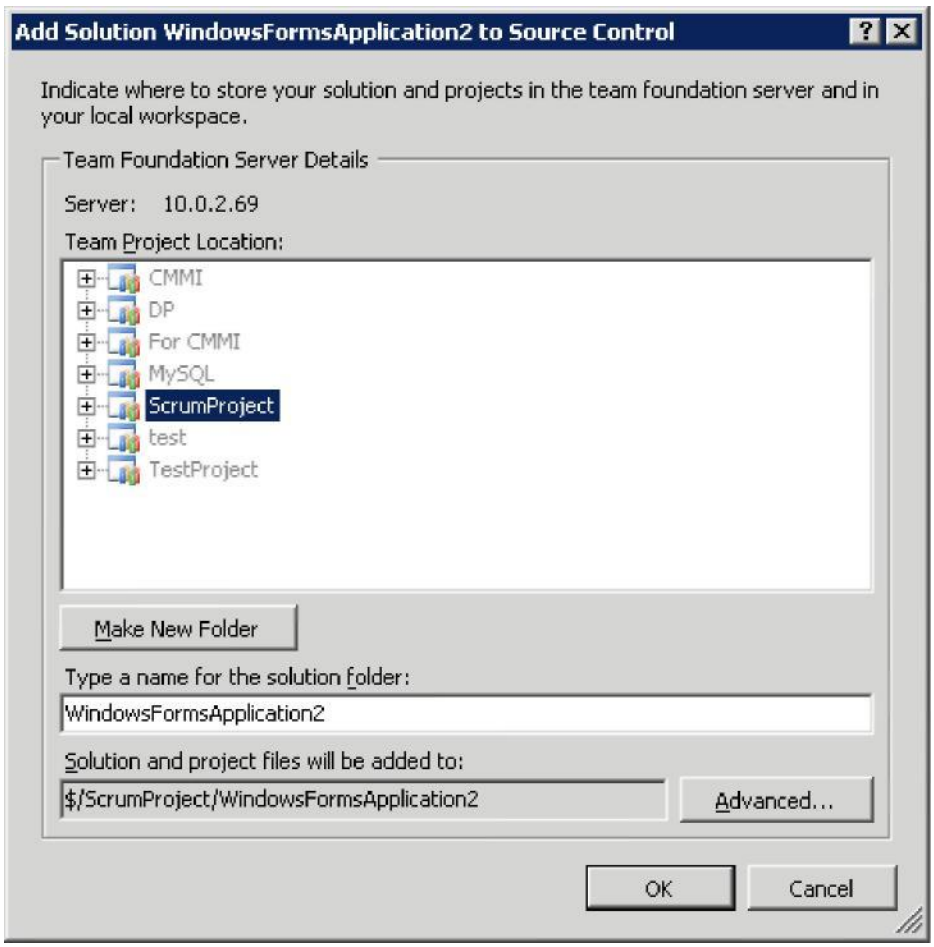


Рис. 14 Окно добавления решения.

Затем необходимо внести все данные в систему контроля версий, используя команду Check-in, открывающую диалог (рис. 15):

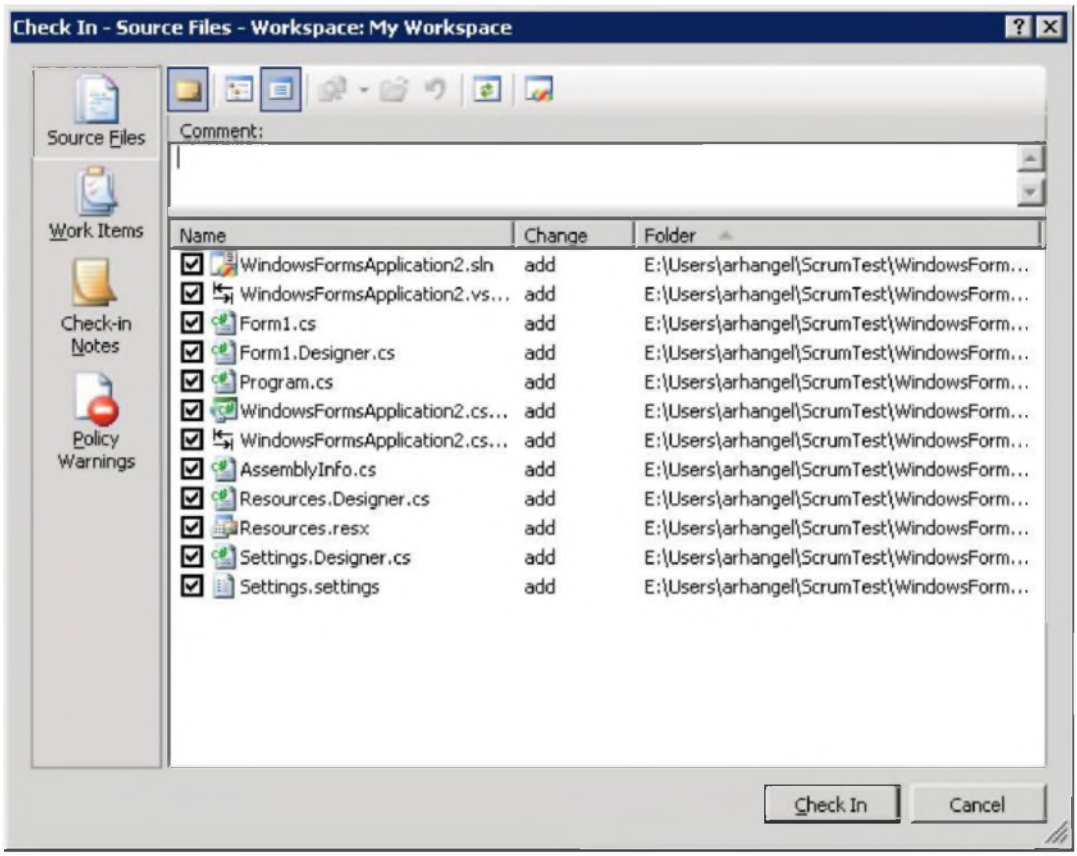
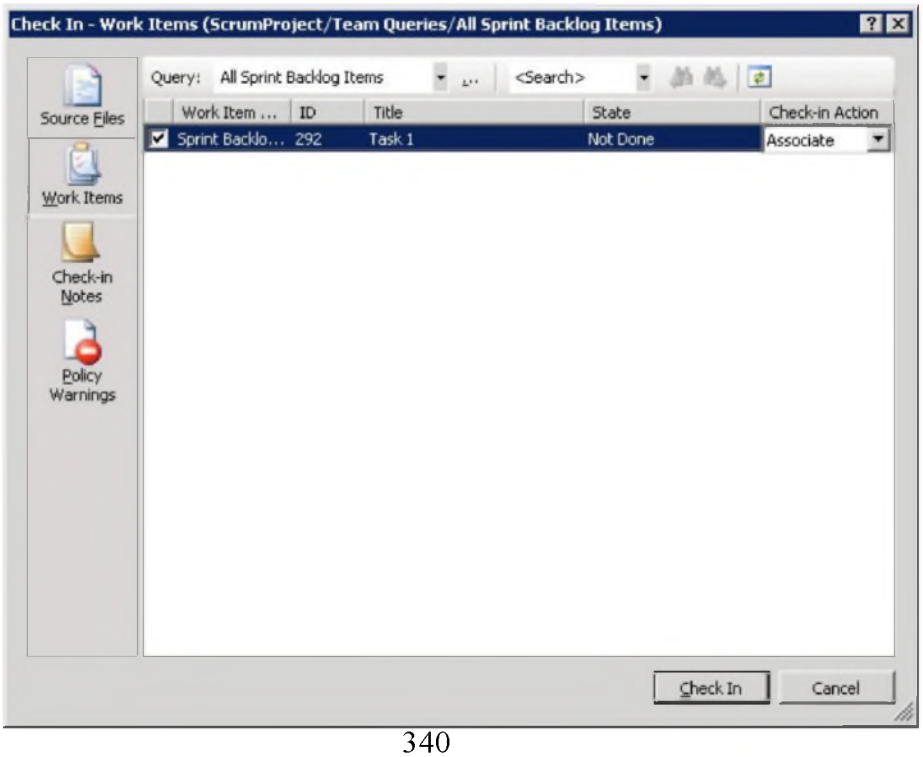


Рис. 15. Диалоговое окно контроля версий.

В этом диалоге необходимо внести комментарии к вносимому коду, а также, на вкладке Work Items, связать вносимое изменение с элементами работы (рис.16):



Шаг 2. Создание ветки кода.

Для того, чтобы освоится с практикой конфигурационного управления, команды должны создать ветвь в системе контроля версий, следуя приведенной ниже инструкции.

Открыть Source control explorer (рис.17):

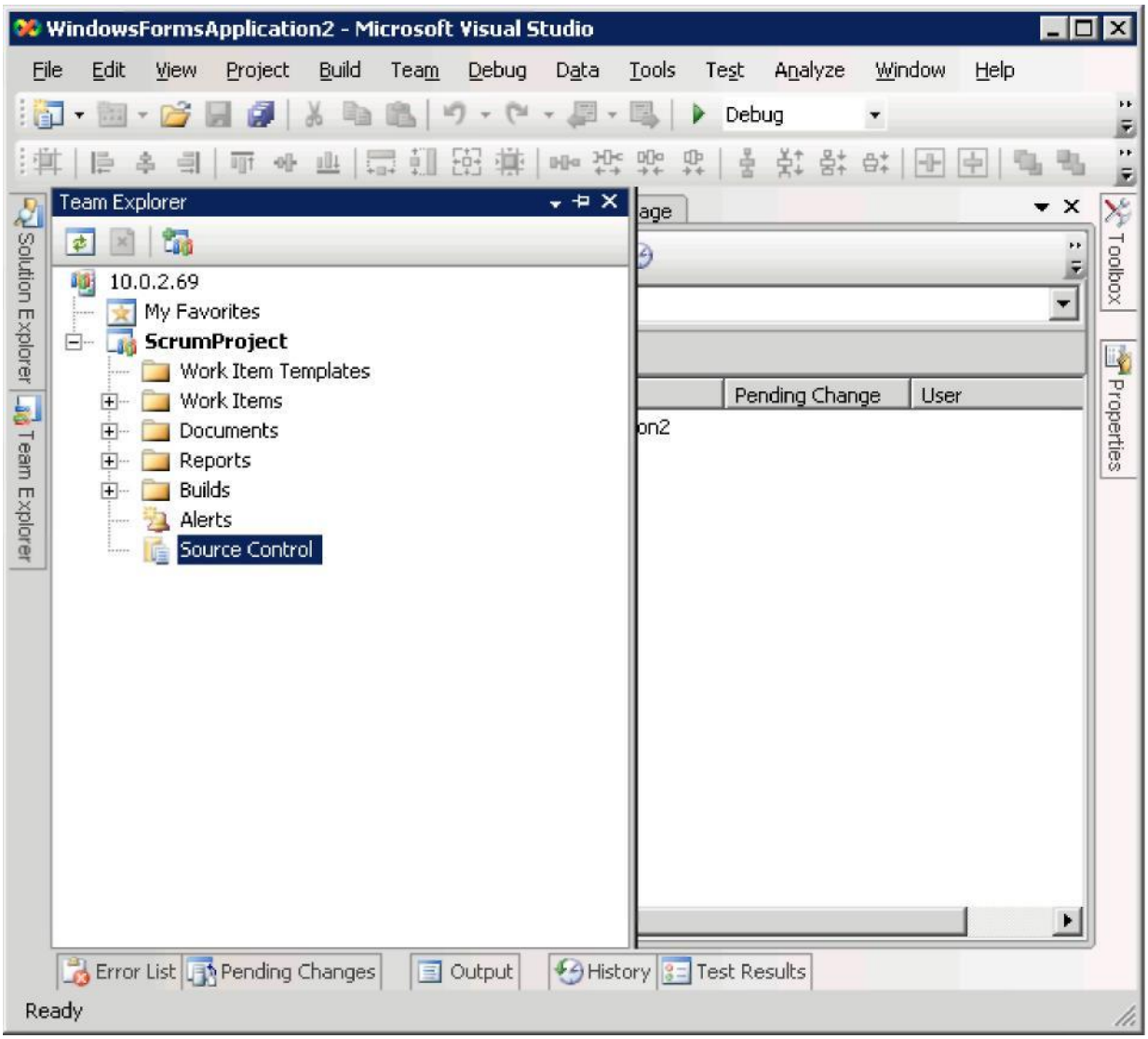


Рис.17. Окно Explorer.

Выбрать нужный проект и в контекстном меню команду Branch: (рис.18)

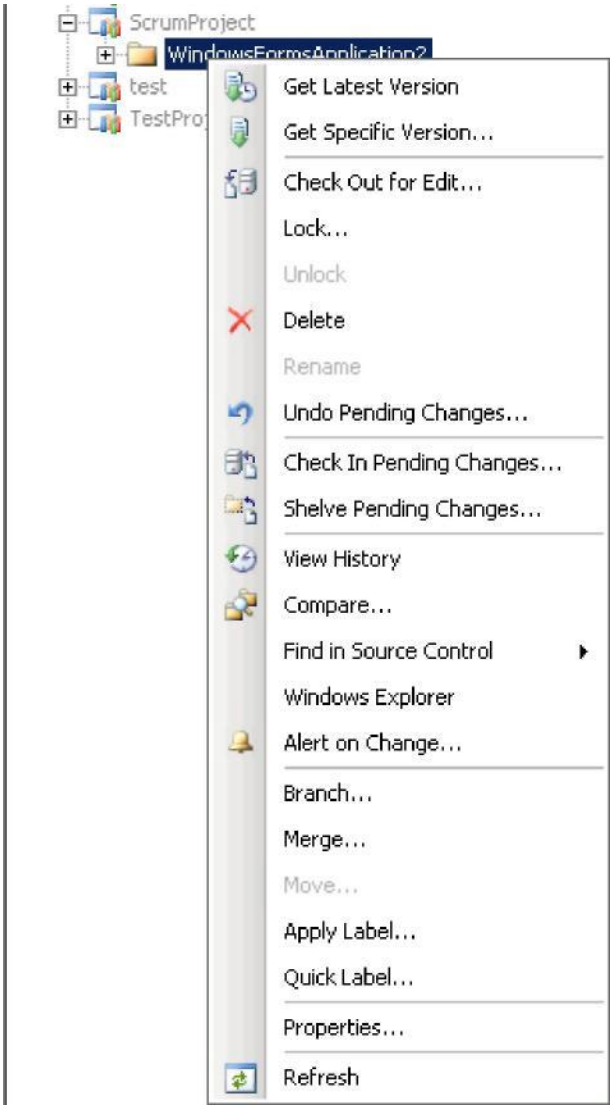


Рис.18.

3. В открывшемся окне задать целевую папку, куда необходимо скопировать данные для новой ветви (рис.19):

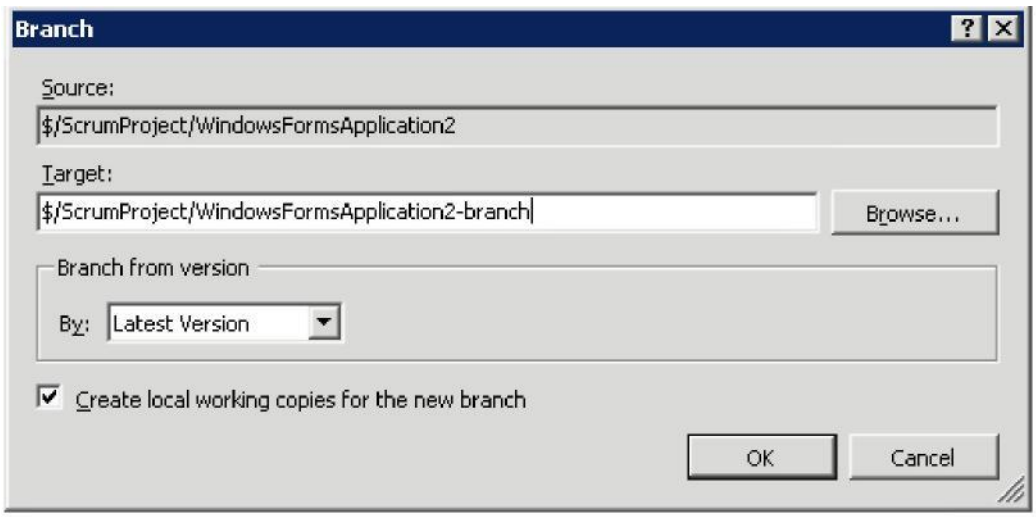


Рис. 19.

После того, как создана ветка, разные участники команды вносят изменения в разные ветки кода, реализую необходимую функциональность приложения.

Шаг 3. Объединение изменений

После того, как в отдельные ветви было внесено некоторое количество изменений, необходимо перенести изменения из отделенной ветви в основную, используя команду Merge (рис. 20):

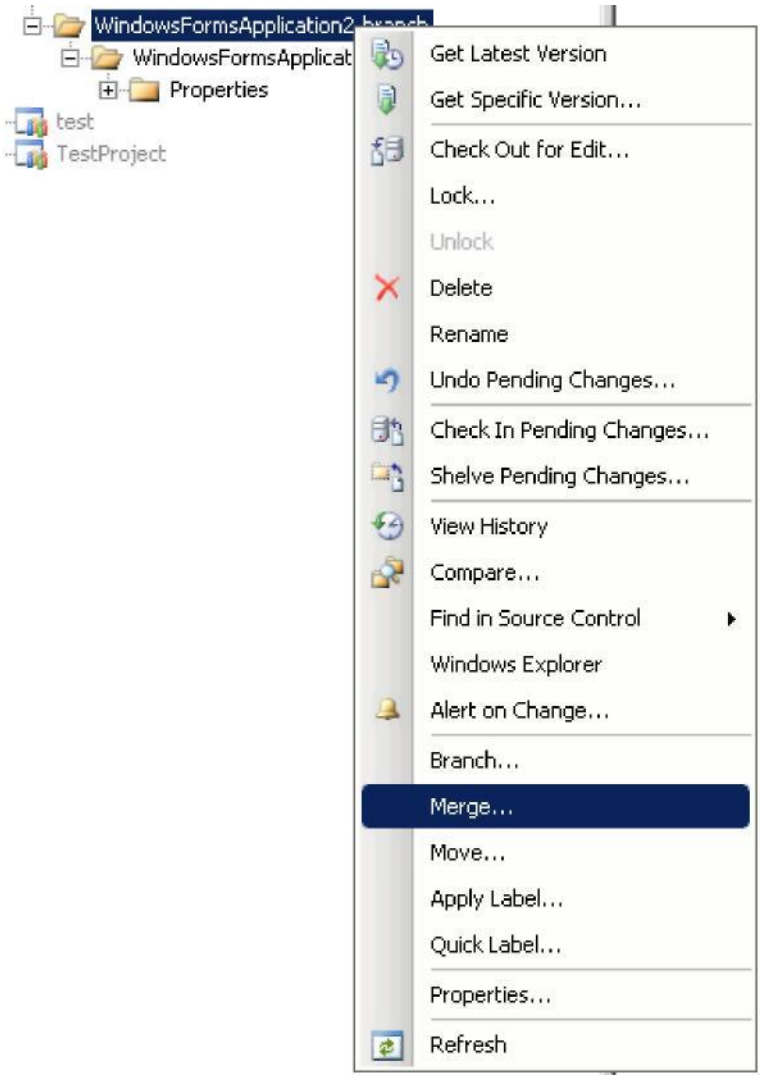


Рис. 20. Перенос изменений.

В процессе объединения изменений могут возникнуть конфликты, информация о которых будет включена в сообщение следующего вида (рис.21):

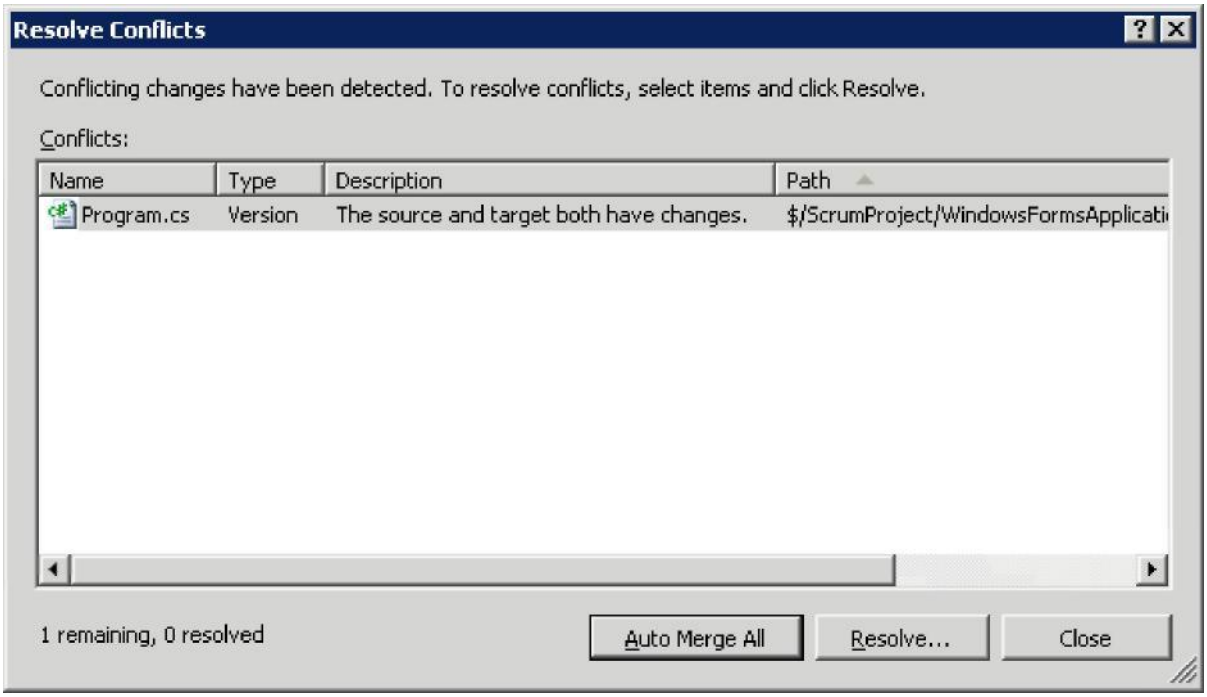


Рис. 21. Окно сообщений

Все конфликты необходимо разрешить, используя команду Resolve и утилиту для объединения результатов.

После разрешения конфликтов все изменения внести в систему контроля версий посредством операции Check-in.

Тема 4. Разработка модульных тестов

На данном занятии команды должны разработать набор модульных тестов, покрывающих функциональность, разработанную на занятии предыдущем. В рамках данного занятие предполагается освоить следующие возможности MS VSTS:

- Автоматическая генерация тестов;
- Наполнение тестов содержимым;
- Запуск тестов и просмотр результатом;
- Изменение конфигурации работы тестов.

Шаг 1. Автоматическая генерация тестов.

Для ускорения разработки тестов команды могут воспользоваться возможность Visual Studio по автоматической генерации тестов. Для этого необходимо воспользоваться командой Test/New Test и выбрать Unit Test wizard в открывшиеся окне (рис.22):

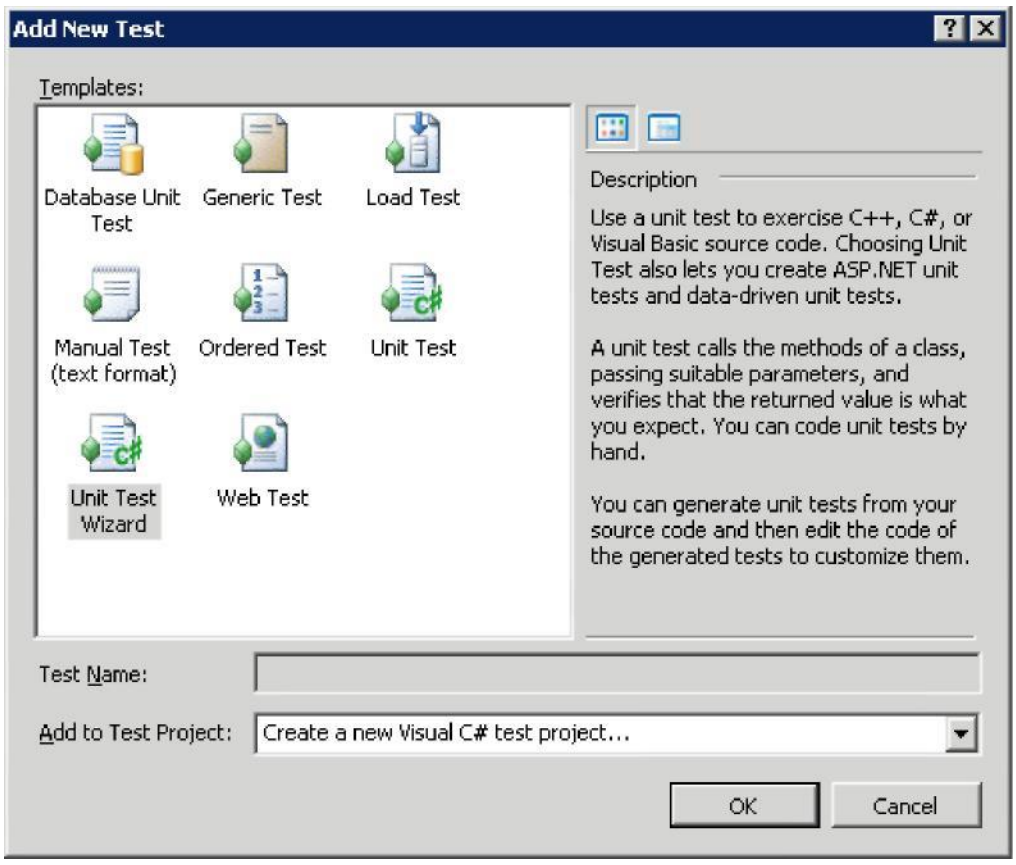


Рис.22. Окно выбора автоматической генерации тестов.

После создания тестового проекта, будет предложен выбор из тех типов и методов, для тестирования которых необходимо создать заглушки (рис.23):

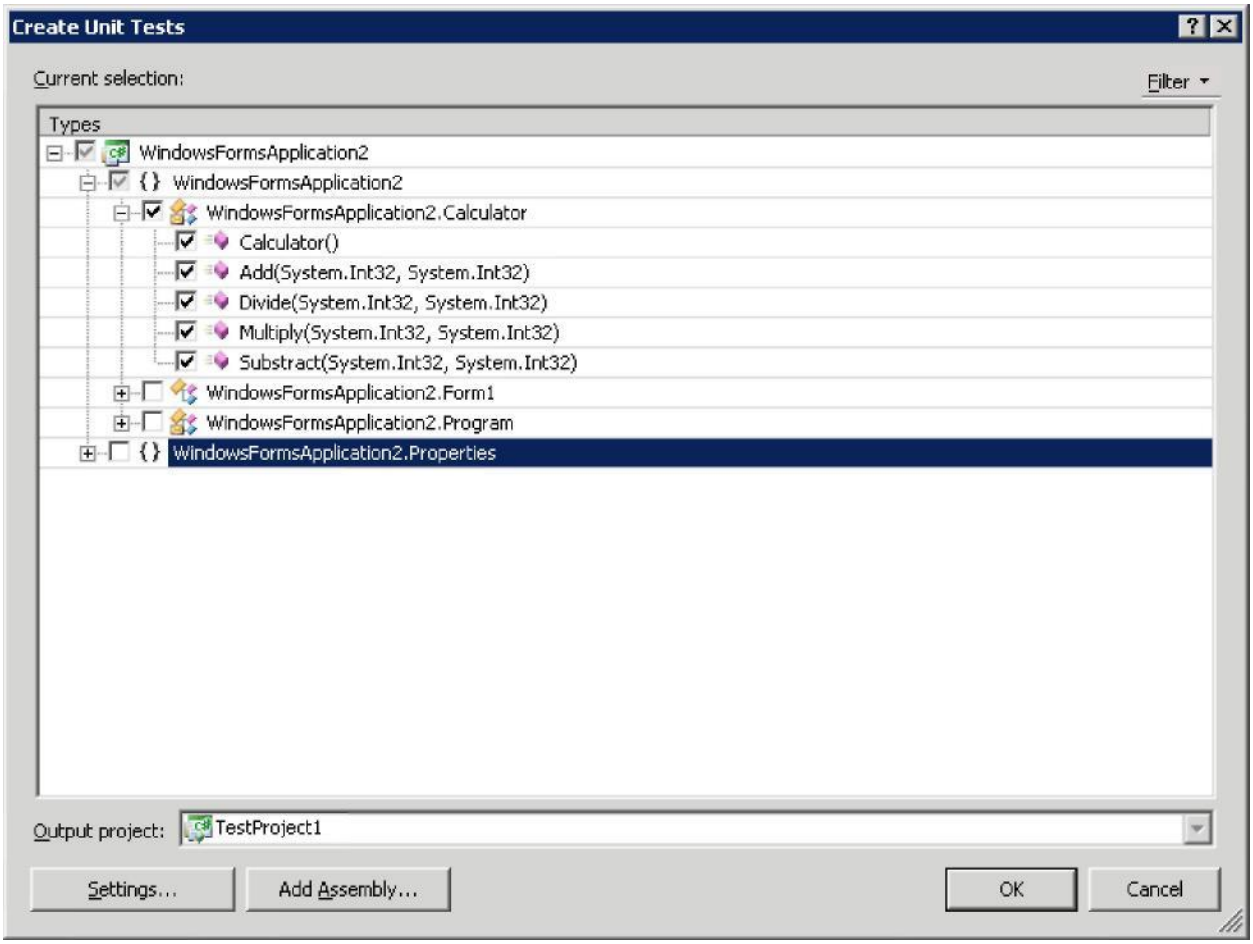


Рис. 23. Создание заглушки.

В этом диалоге команда должна выбрать все основные классы и методы, которые планируется покрыть модульными тестами.

Шаг 2. Наполнение тестов содержимым.

После генерации тестового покрытия команда получить набор скелетов тестов для всех методов, которые были выбраны для тестирования. Однако, эти тесты имеют достаточно простую структуру и пока лишены смысла:

```
/// <summary>
///A test for Multiply
///</summary>
[TestMethod()]
public void MultiplyTest()
```

```
{
// TODO: Initialize to an appropriate value
Calculator target = new Calculator();
int a = 0; // TODO: Initialize to an appropriate value
int b = 0; // TODO: Initialize to an appropriate value
int expected = 0; // TODO: Initialize to an appropriate value
int actual;
actual = target.Multiply(a, b);
Assert.AreEqual(expected, actual);
Assert.Inconclusive("Verify the correctness of this test
method.");
}
```

На следующем шаге команда должна заполнить эти тесты необходимым содержимым, используя функциональность по валидации (Assert), предоставляемую тестовой платформой.

Шаг 3. Запуск тестов

Для того, чтобы исполнить созданные тесты необходимо использовать соответствующую панель инструментов (рис. 24):

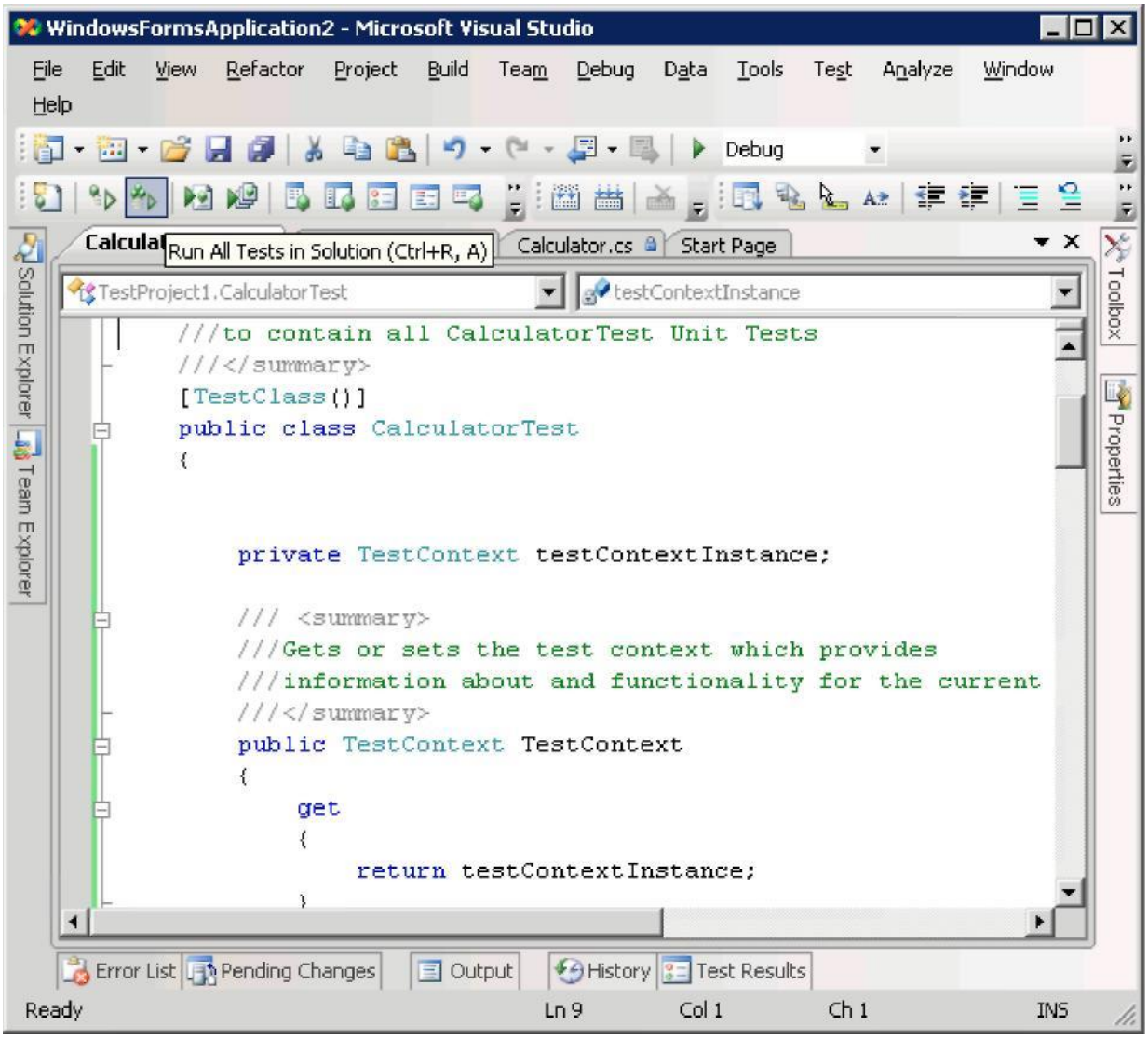


Рис. 24 Панель запуска тестов.

После этого результаты выполнения тестов будут видны в окне результатов (рис.25):

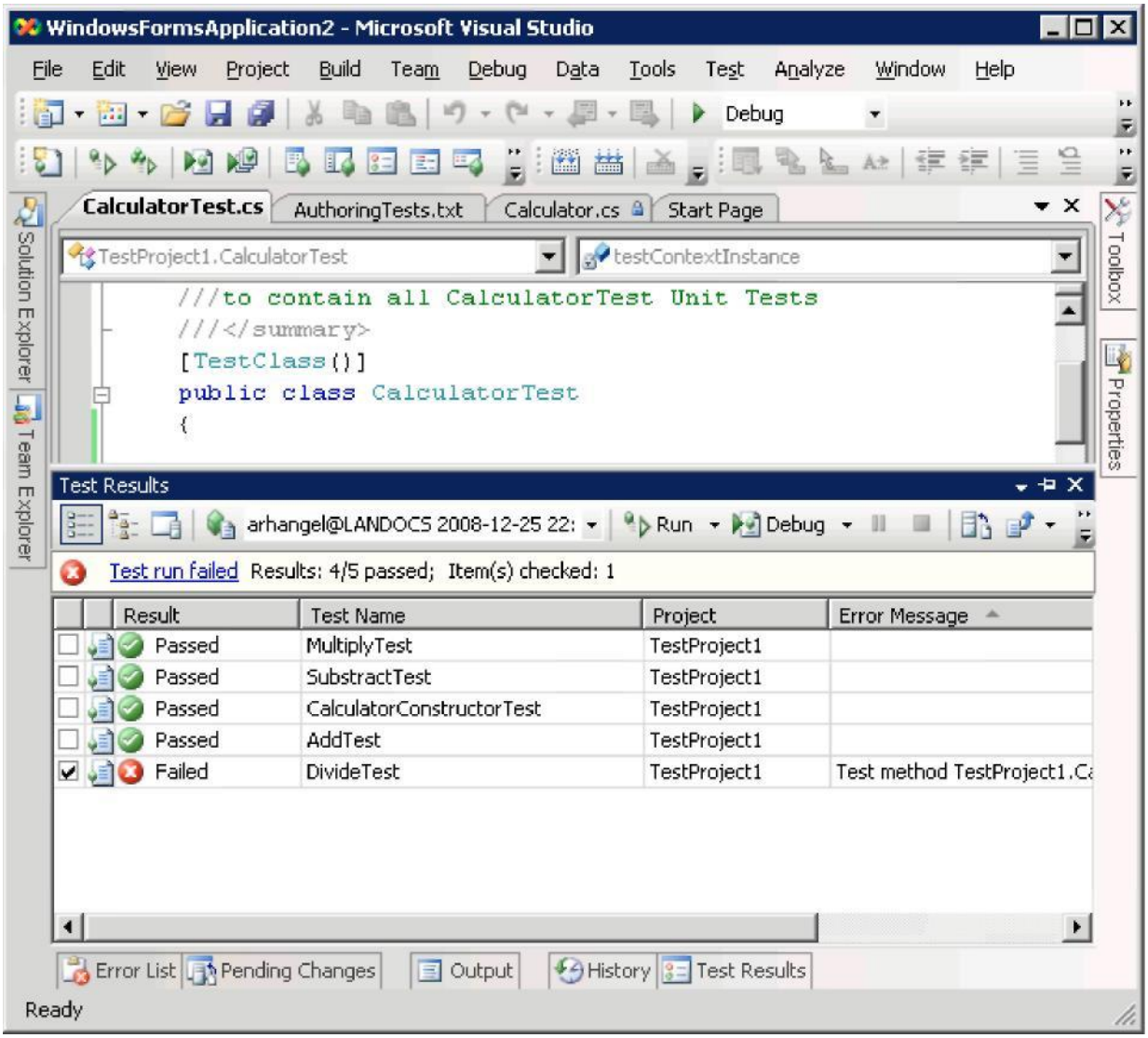


Рис. 25. Окно результатов тестов.

Команды должны добиться того, чтобы все разработанные тесты проходили успешно.

Шаг 4. Изменение конфигурации тестов.

Для того, чтобы проанализировать качество разработанных тестов, команда должна вычислить тестовое покрытие. Для этого е необходимо:

Открыть файл конфигурации запуска тестов, автоматически добавленный к решению при создании тестов (рис. 26):

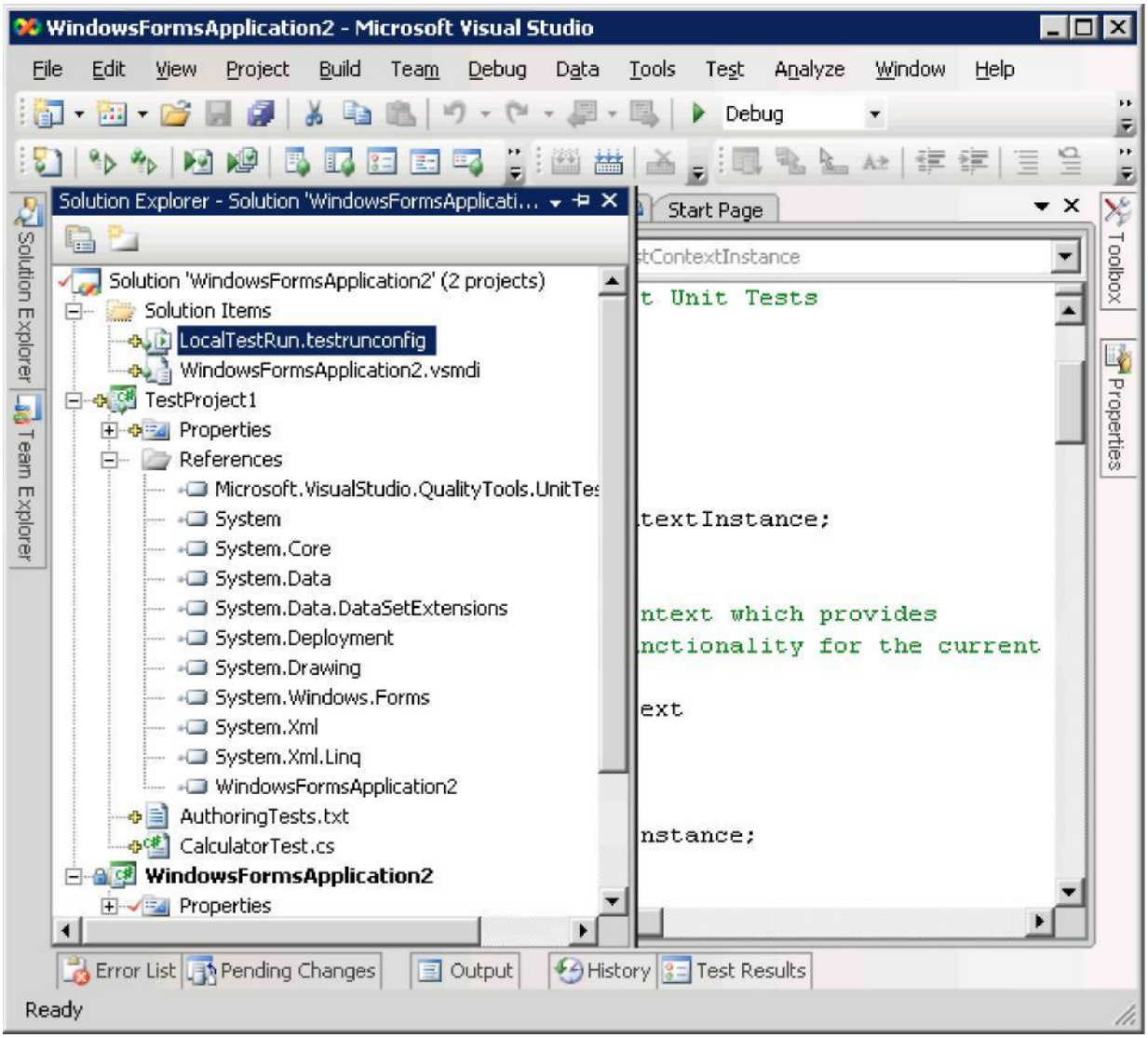


Рис. 26.

2. На открывшемся диалоге выбрать вкладку Code Coverage и установить то, какие именно проекты нужно анализировать (рис.27):

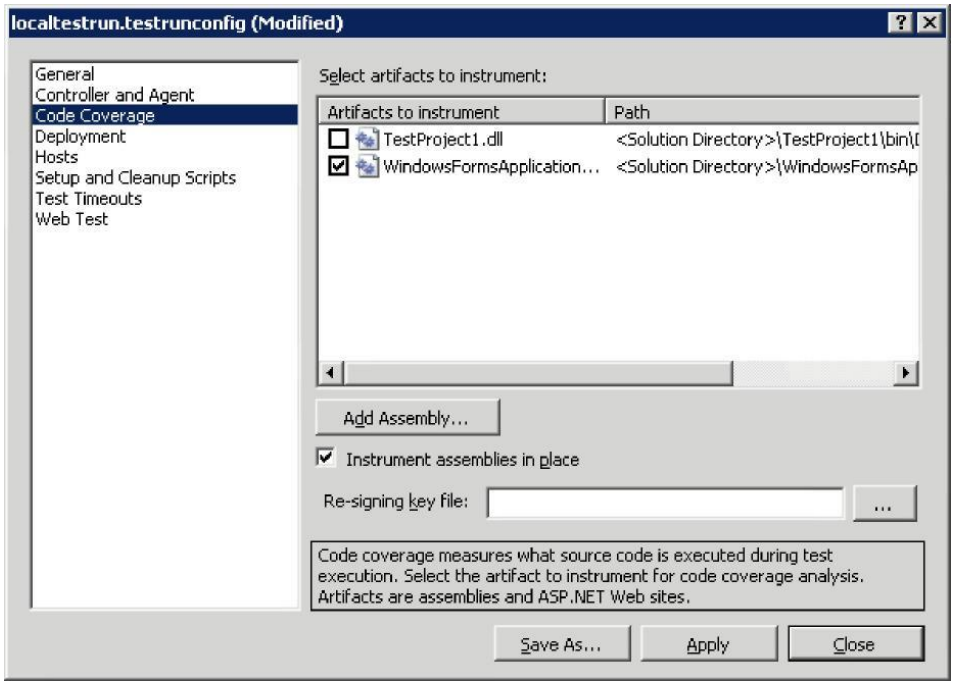


Рис. 27.

3. После сохранения конфигурации запустить тесты и активировать опцию Show Code Coverage Coloring (рис.28):

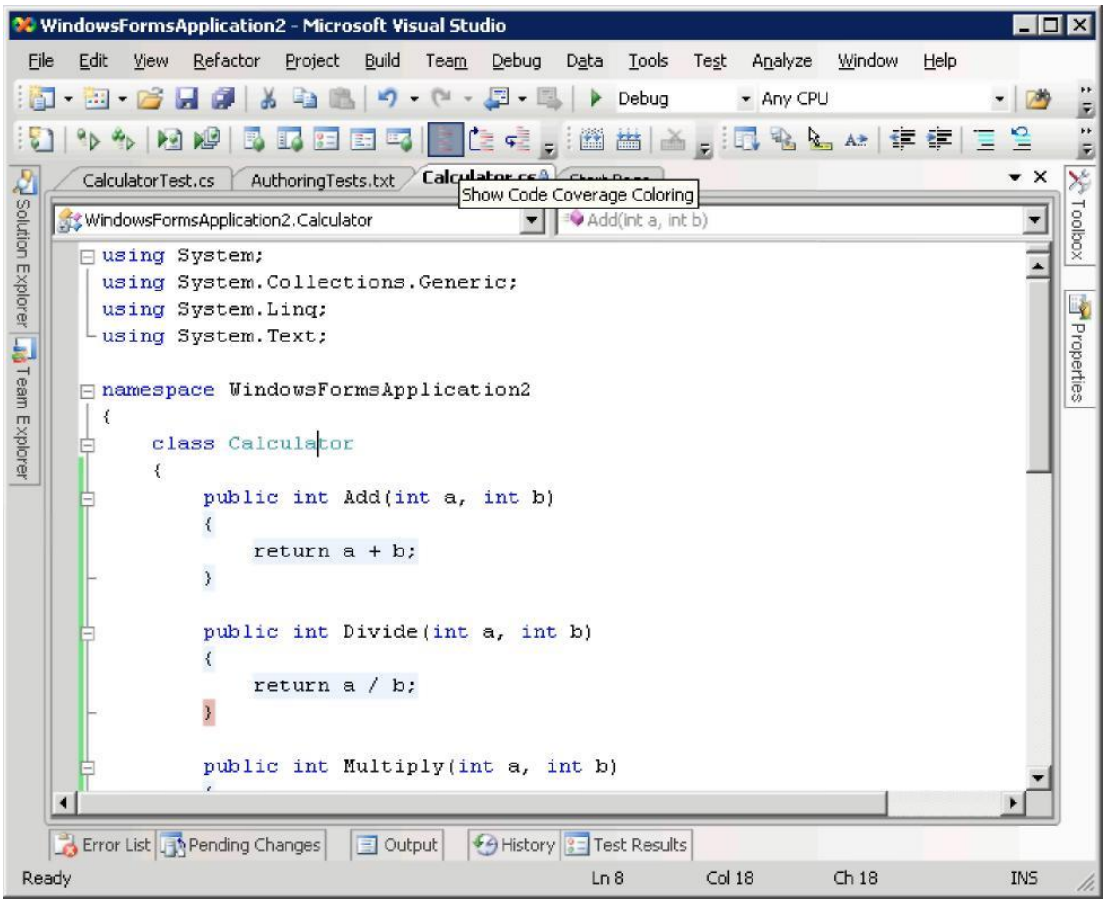


Рис. 28.

Тема 5. Создание и конфигурация автоматической сборки

На данном этапе команда должна создать в своем проекте процедуру автоматической сборки. При этом должно быть создано несколько процедур:

- Простая процедура, включающая только сборку;
- Полная процедура, включающая тесты и анализ кода;
- После создания сборок необходимо настроить параметры непрерывной интеграции;
- Простая сборка должна запускаться после каждого внесенного изменений, но не чаше чем раз в 5 минут;
- Полная сборка должна запускаться каждую ночь.

Шаг 1. Создание простой сборки

Все результаты сборки, проведенной TFS, выкладываются в разделяемую папку, на запись в которую есть права у пользователя, с правами которого работает сервер автоматических сборок (обычно – TFSBuild), а также у пользователя, с правами которого работает сам TFS (обычно – TFSService). Как правило, учащиеся не обладают достаточным количеством прав для создания такого рода папок, поэтому они должны быть заранее подготовлены преподавателем.

Для создания простой сборки необходимо обратится к окну Team Explorer, после чего в разделе Builds выбрать команду Build Definitions (рис. 29):

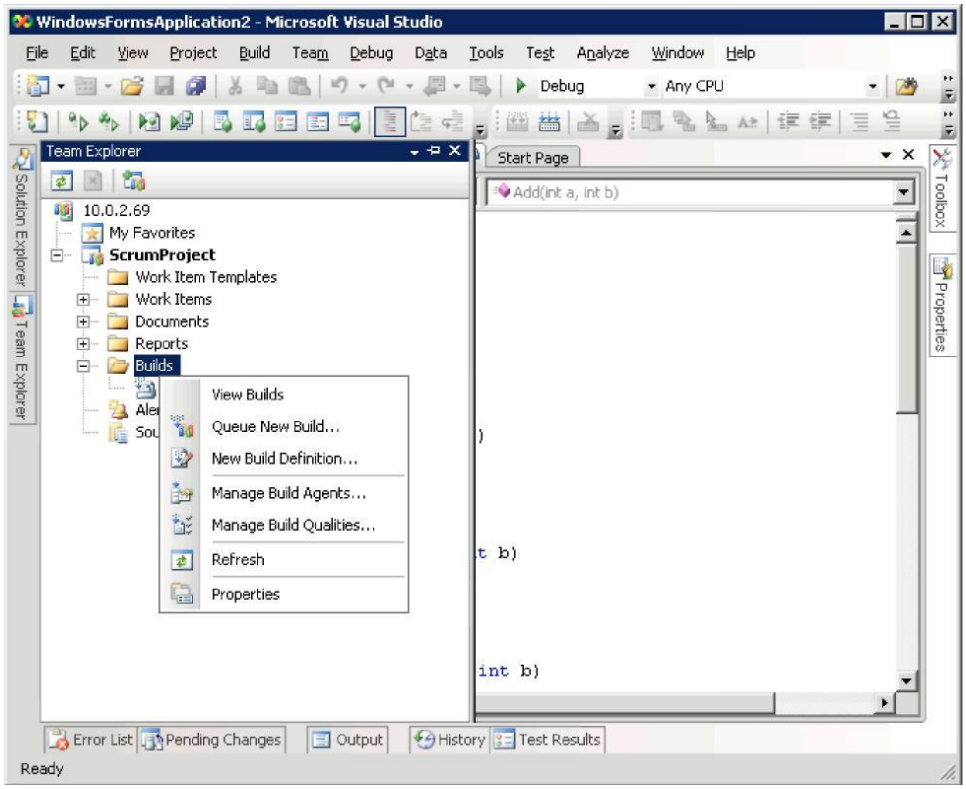


Рис. 29. Создание сборки.

Вызов этой команды приведет к открытию мастера, в котором нужно задать следующие параметры сборки:

1. Задать имя и описание сборки (рис.30):

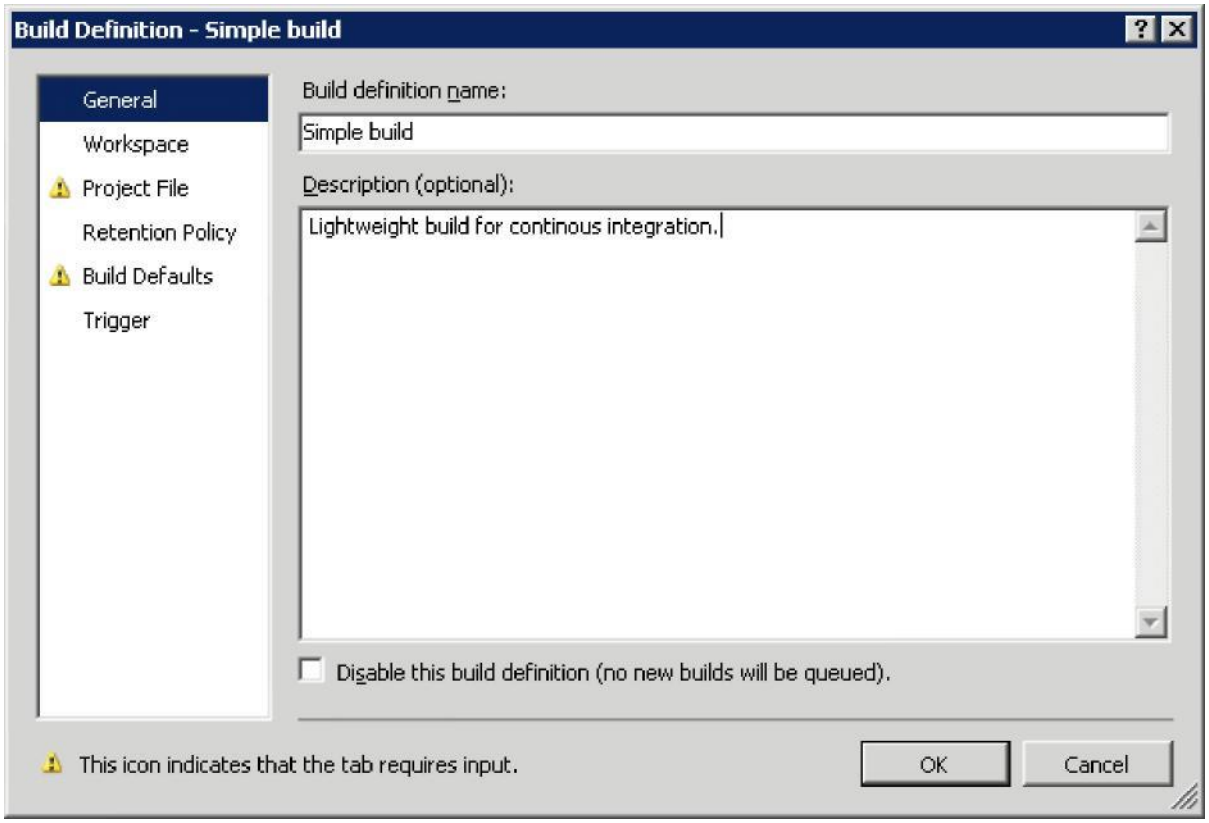


Рис. 30. Задание имени сборки.

2. На закладке Projec tFile, создать новое описание сборки используя кнопку Create, после чего задав проекты и конфигурации для сборки (рис.31-32):

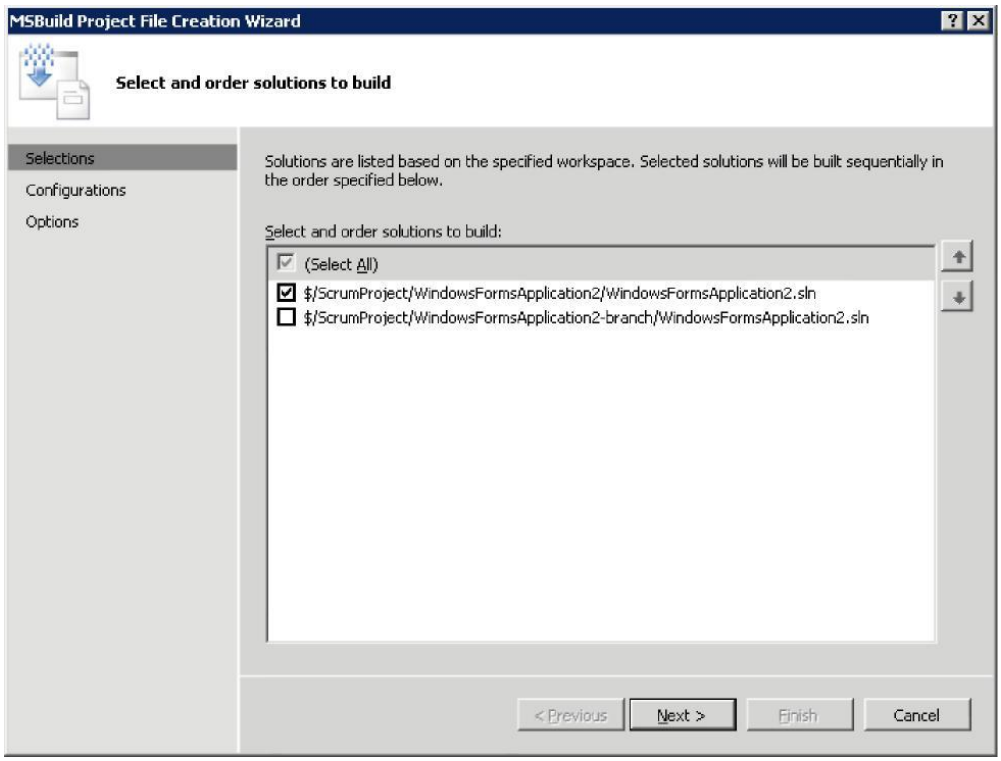


Рис. 31. Задание проектов для сборки.

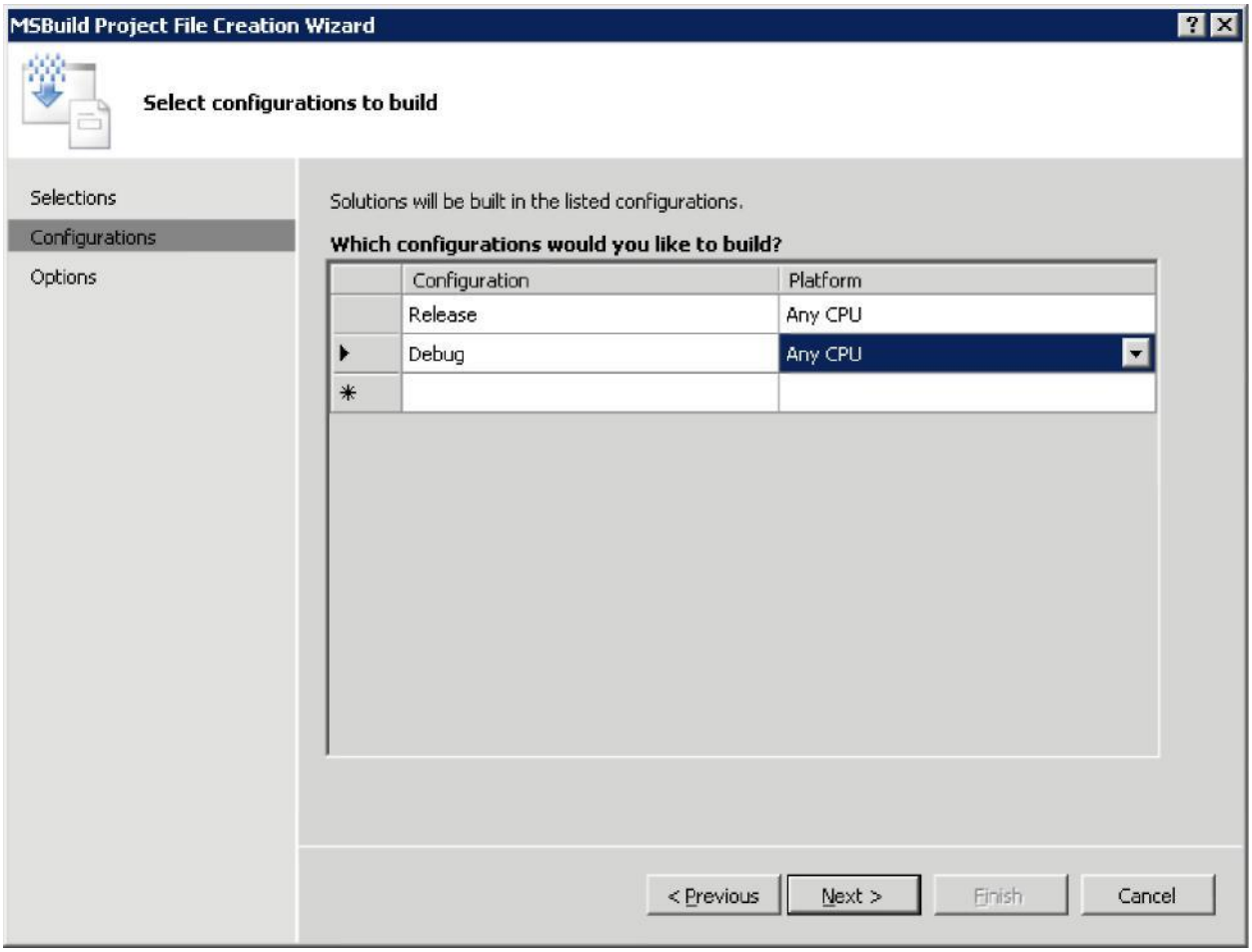


Рис. 32. Задание конфигурации для сборки.

3. На закладке Build Defaults (рис. 33) создать определение агента-сборщика используя кнопку New. Для агента указать имя, описание, и IP адрес сервера сборок (как правило, совпадает с сервером TFS):

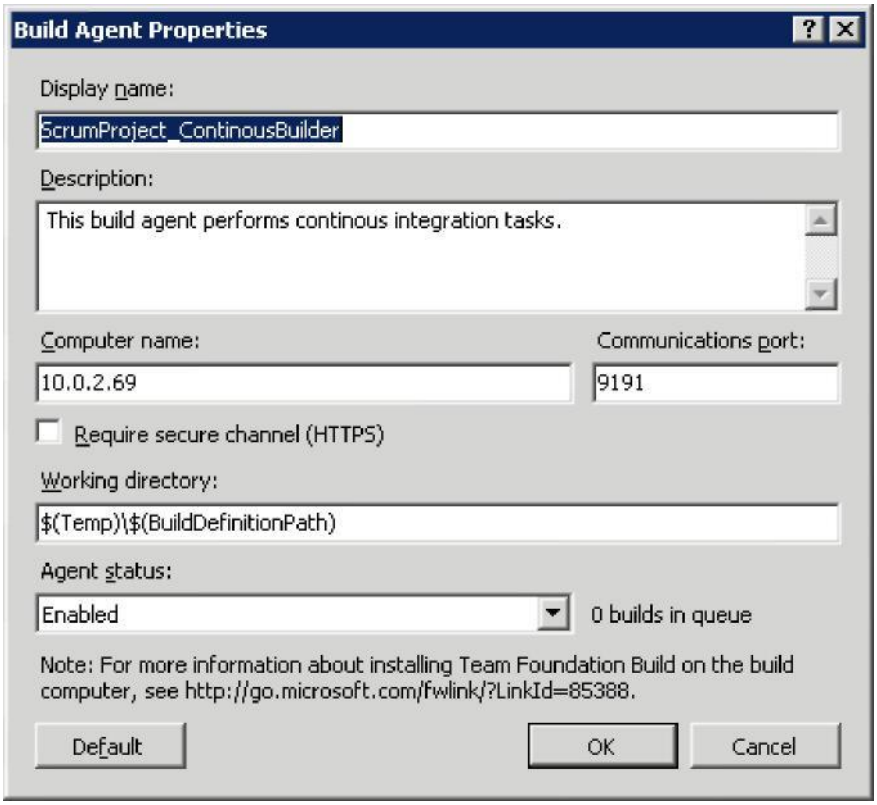


Рис.33. Закладка Build Defaults.

4. На закладке Build Defaults (рис. 34) также необходимо задать имя разделяемой папки, в которую будут сложены результаты:

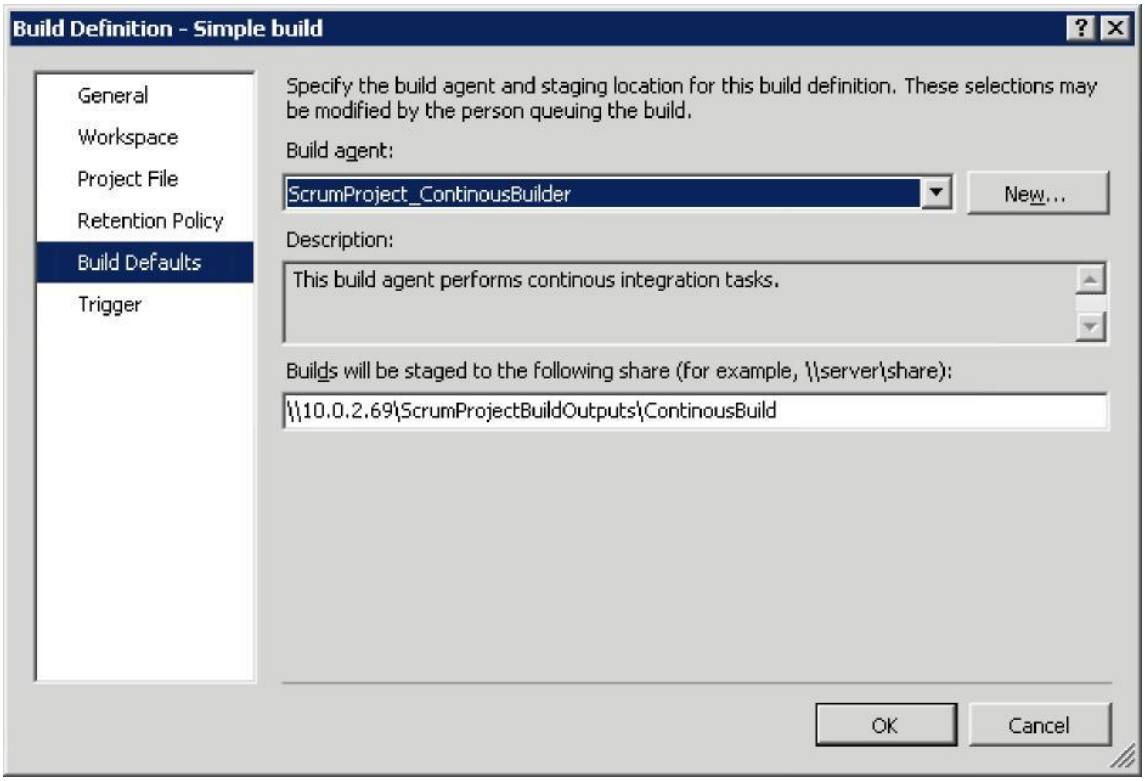
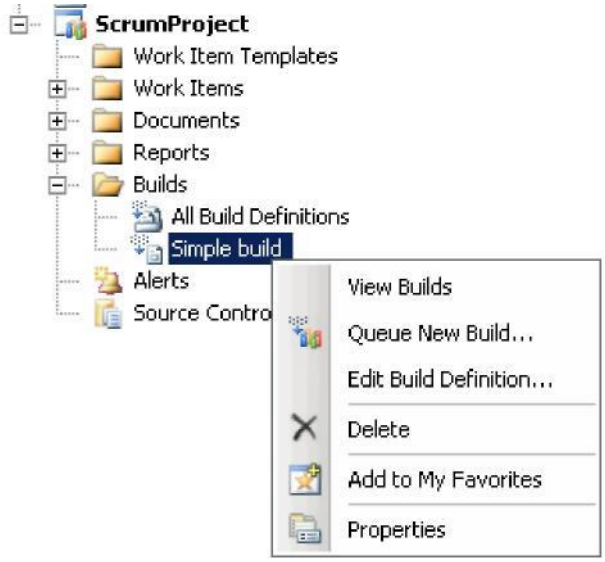


Рис.33. Задание имени разделяемой папки на закладке BuildDefaults.

После того, как определение сборки было создано, ее необходимо запустить, используя команду Queue new build:



После запуска сборки необходимо дождаться ее завершения, и убедится, что на соответствующей сетевой папке появились результаты сборки.

Шаг 2. Создание сложной сборки.

Создание сложной сборки проходит во многом аналогично созданию сборки простой, за исключением нескольких шагов:

1. На закладке опций при создании проекта сборки необходимо включить автоматический запуск модульных тестов и анализ кода (рис. 34):

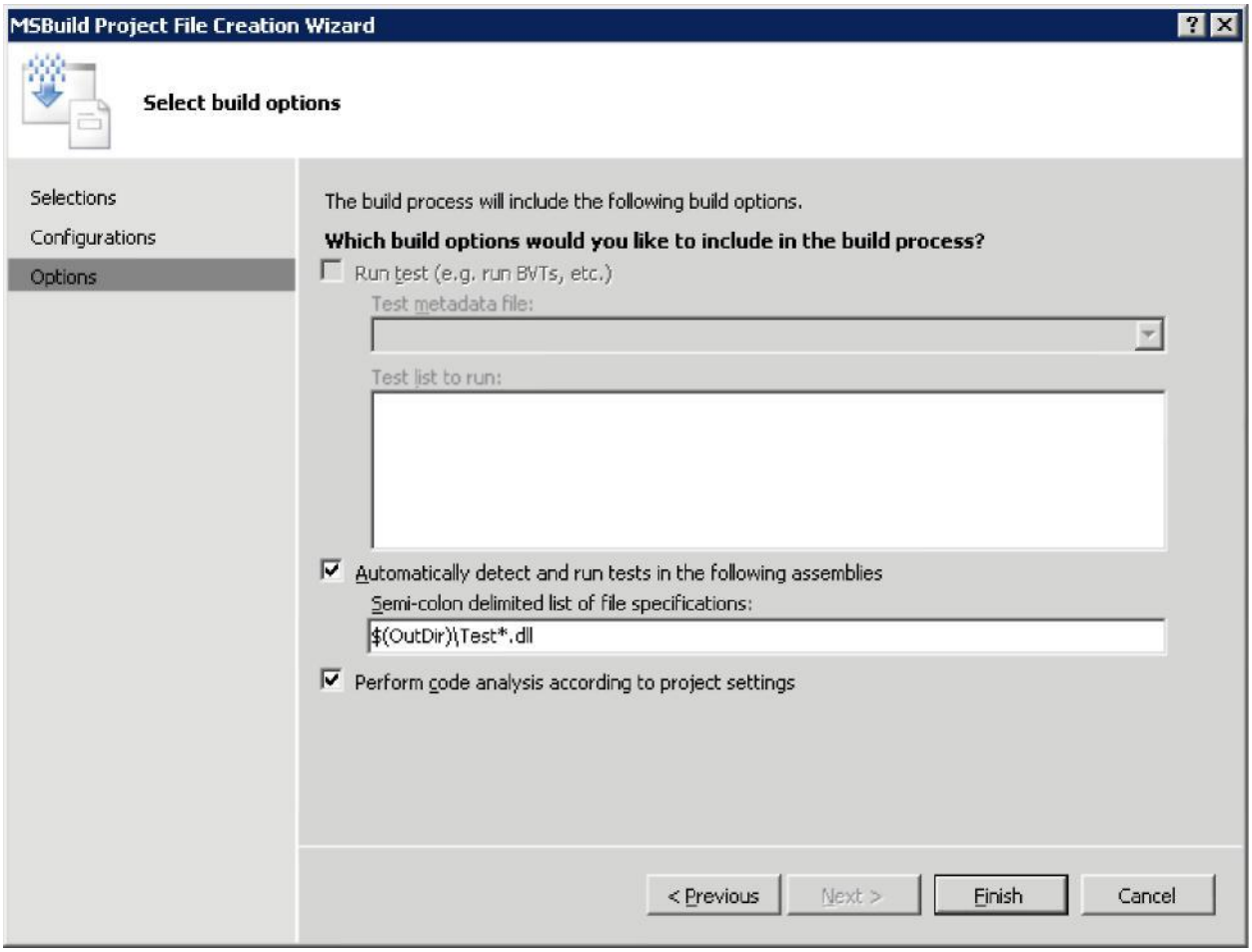


Рис. 34. Закладка опций.

2. На закладке Build Defaults необходимо задать другую папку для сбора результатов (рис. 35):

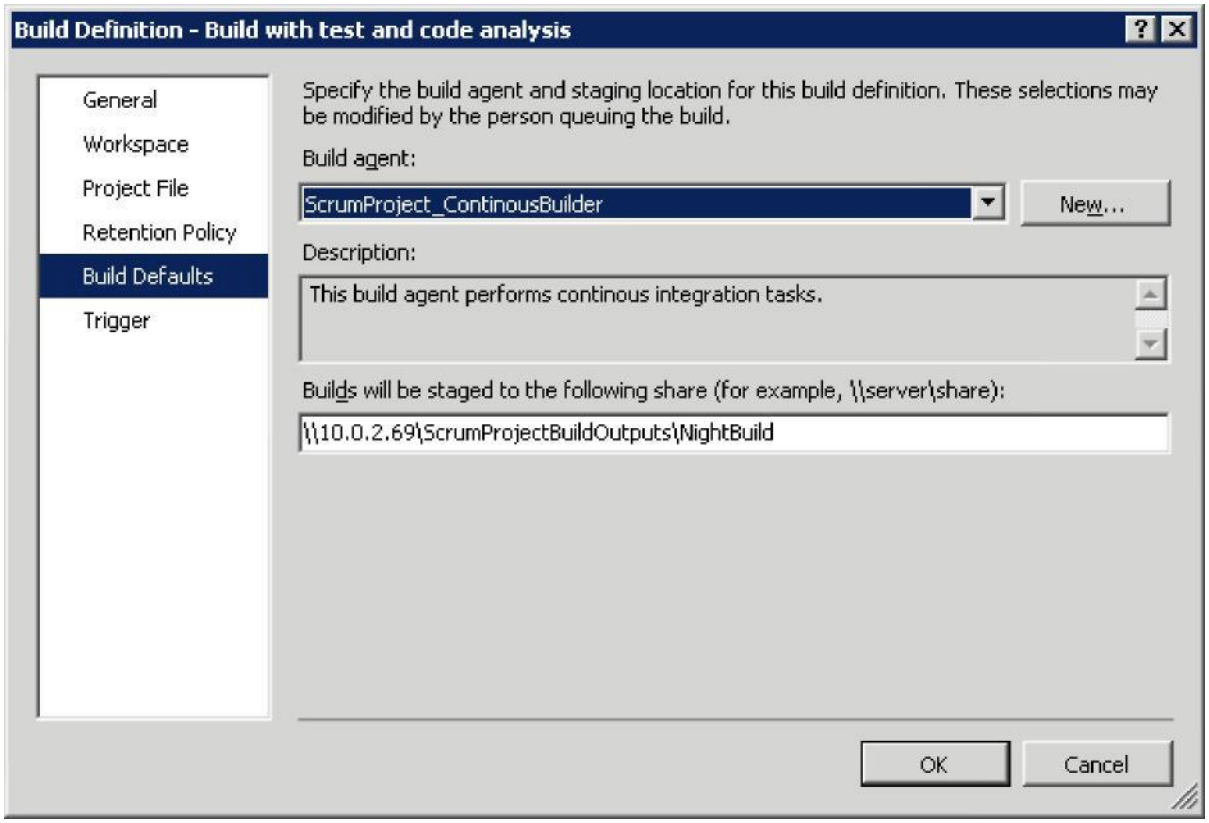


Рис.35. Сбор результатов.

После создания сборки необходимо запустить ее, и обратить внимание, что результаты сборки включают и результаты тестов (рис. 36):

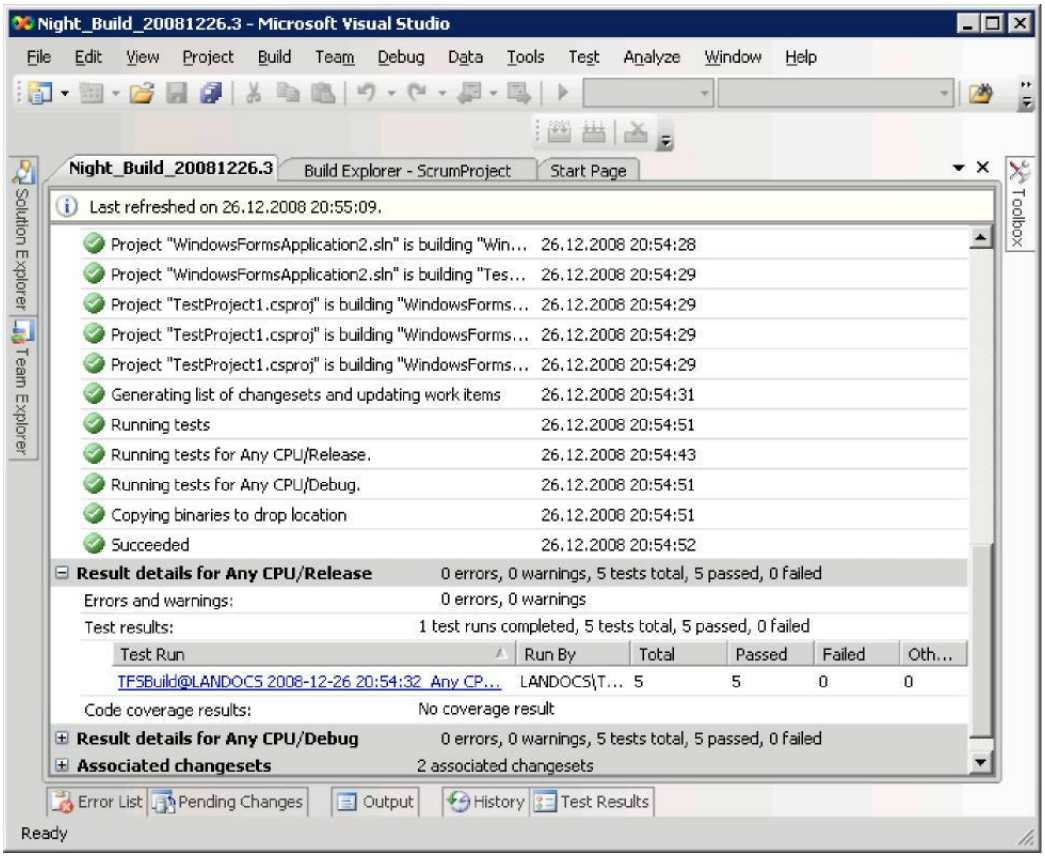


Рис. 36. Включение результатов тестов.

Теперь нужно добиться выполнения статического анализа кода во время ночной сборки. Для этого необходимо активировать анализ кода в настройках соответствующих проектов (рис. 37):

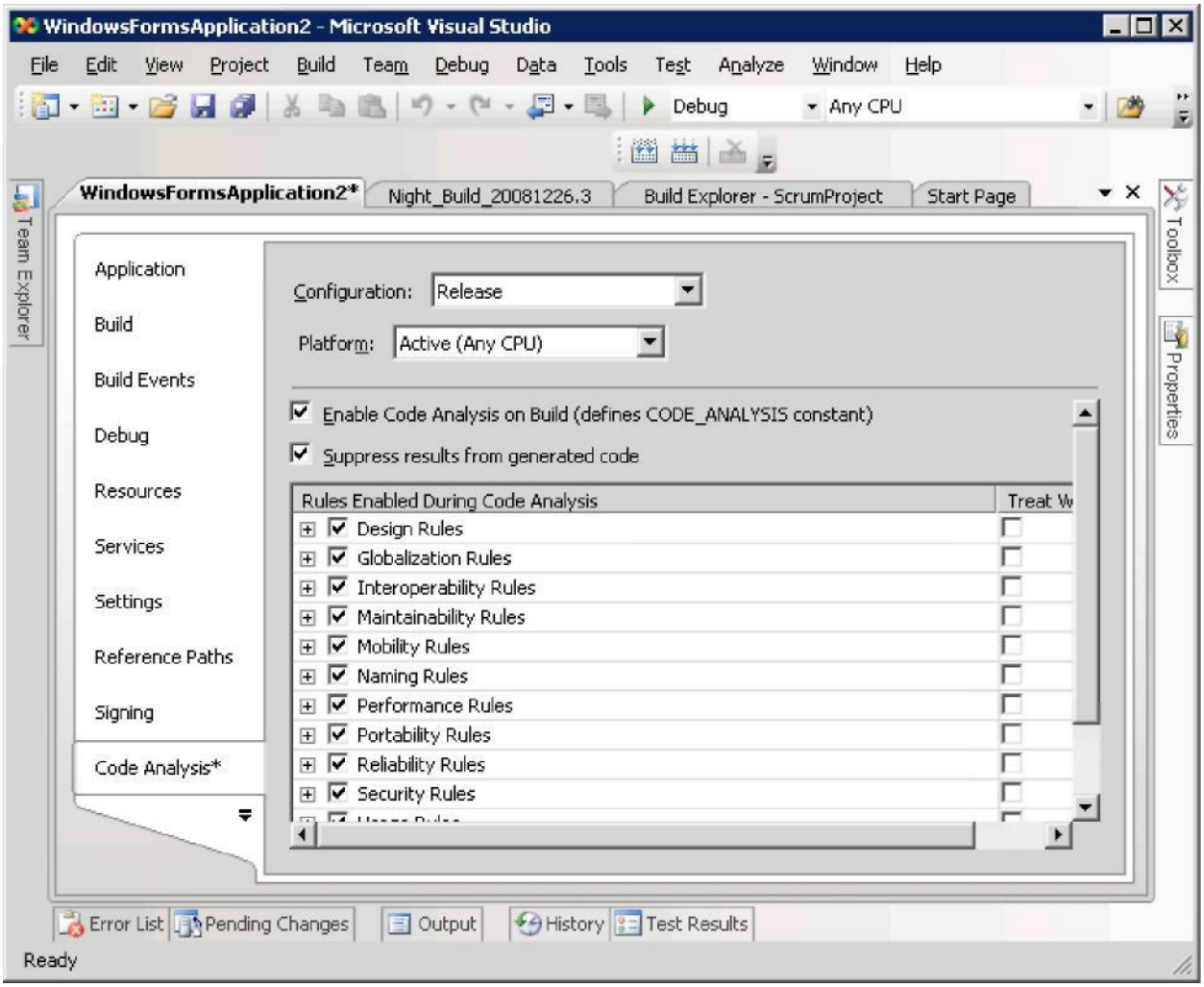


Рис. 37. Настройки проектов.

Следующая же собранная сборка будет содержать большое количество предупреждений от анализатора (рис. 38):

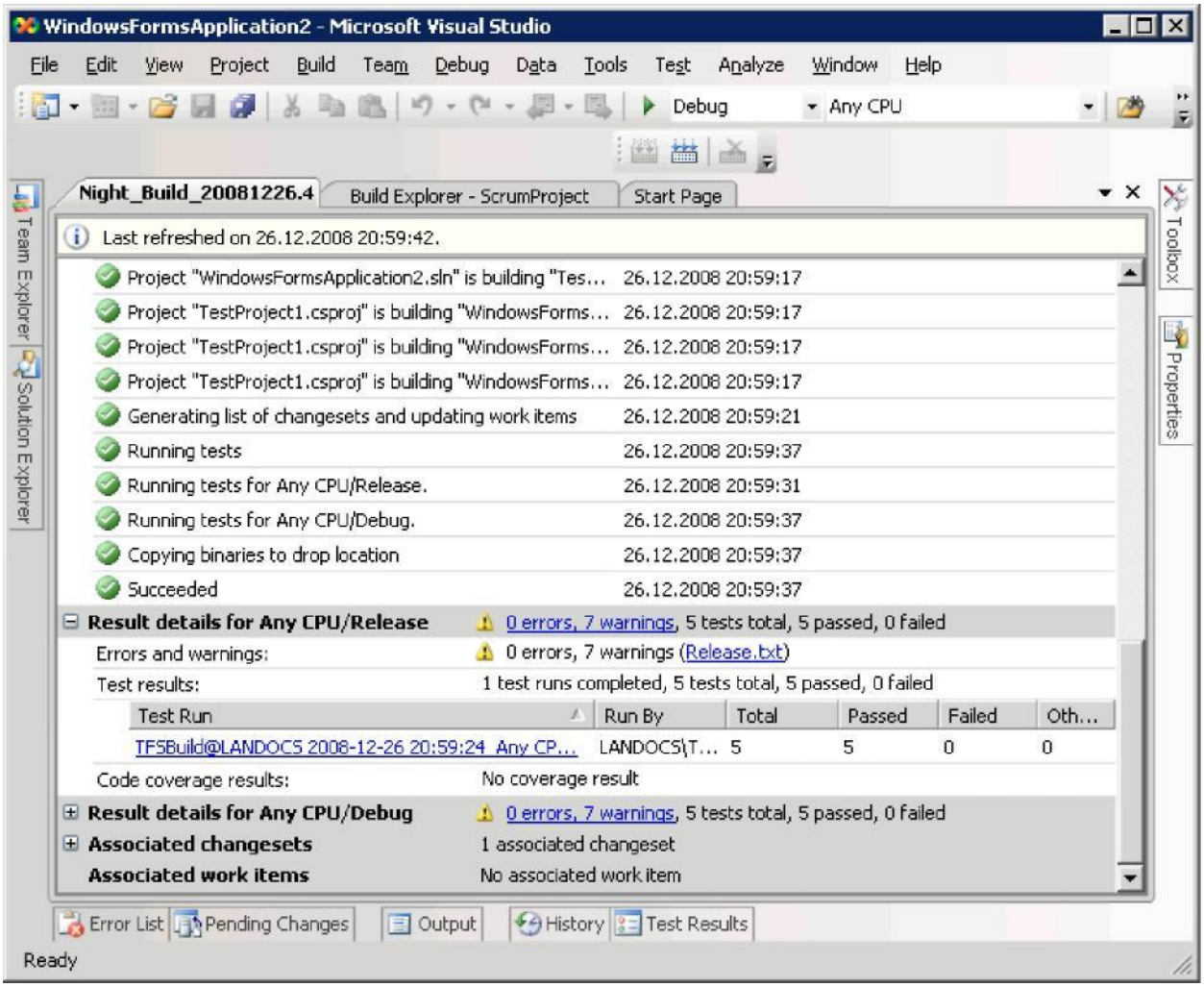
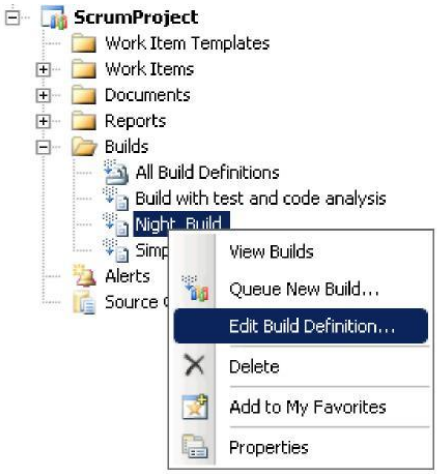


Рис. 38. Окно предупреждений.

Шаг 3. Настройка непрерывной интеграции.

На данном шаге учащимся необходимо исправить описания сборок таким образом, чтобы они выполнялись автоматически при определенных условиях. Простой вариант сборки должен запускаться автоматически после каждого внесения изменений, но не чаще, чем в пять минут. Для того, чтобы добиться этого необходимо:

1. Вызвать команду Edit build definition:



2. Задать настройки автоматического запуска на закладке Trigger (рис. 39):

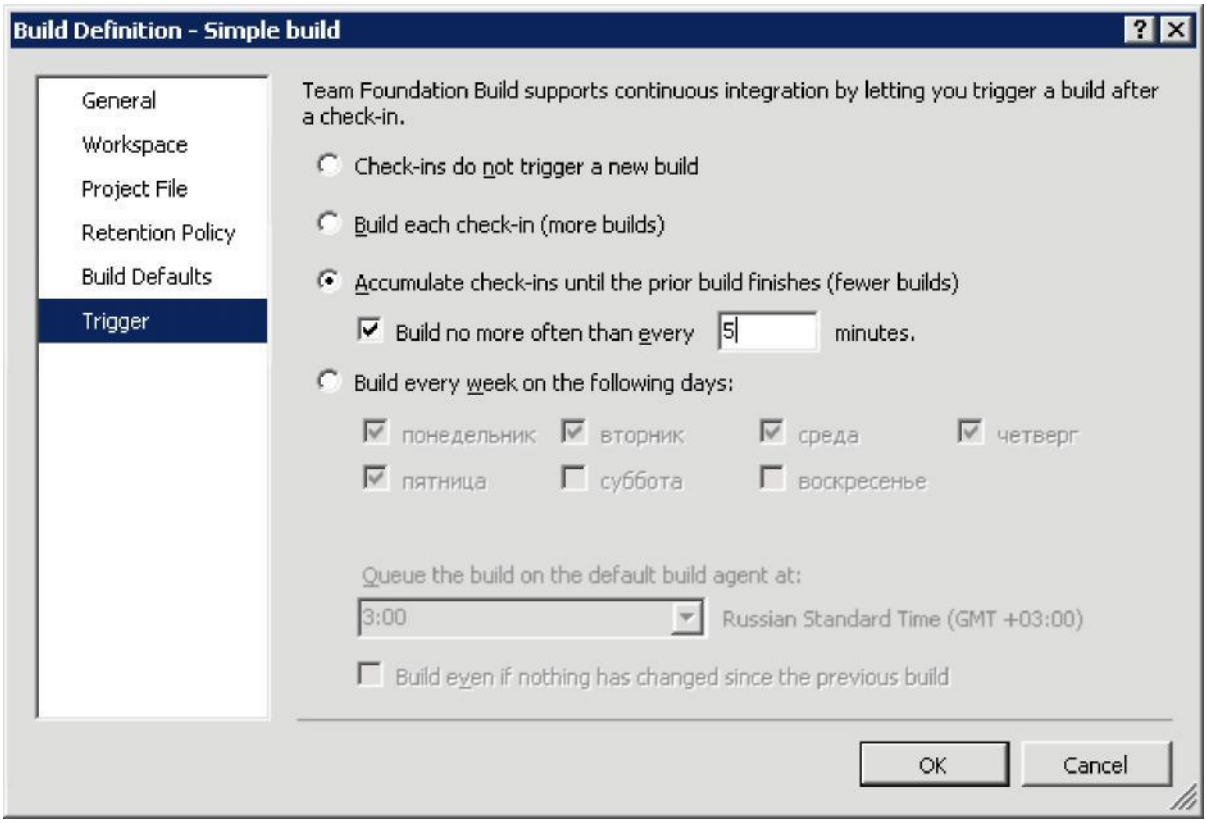


Рис. 39. Настройка автоматического запуска.

3. Настроить политику очистки сборок на закладке Retention Policy (рис. 40). Это необходимо для того, чтобы избежать быстрого исчезновения места на машине-сборщике и для удаления из базы TFS информации о второстепенных сборках:

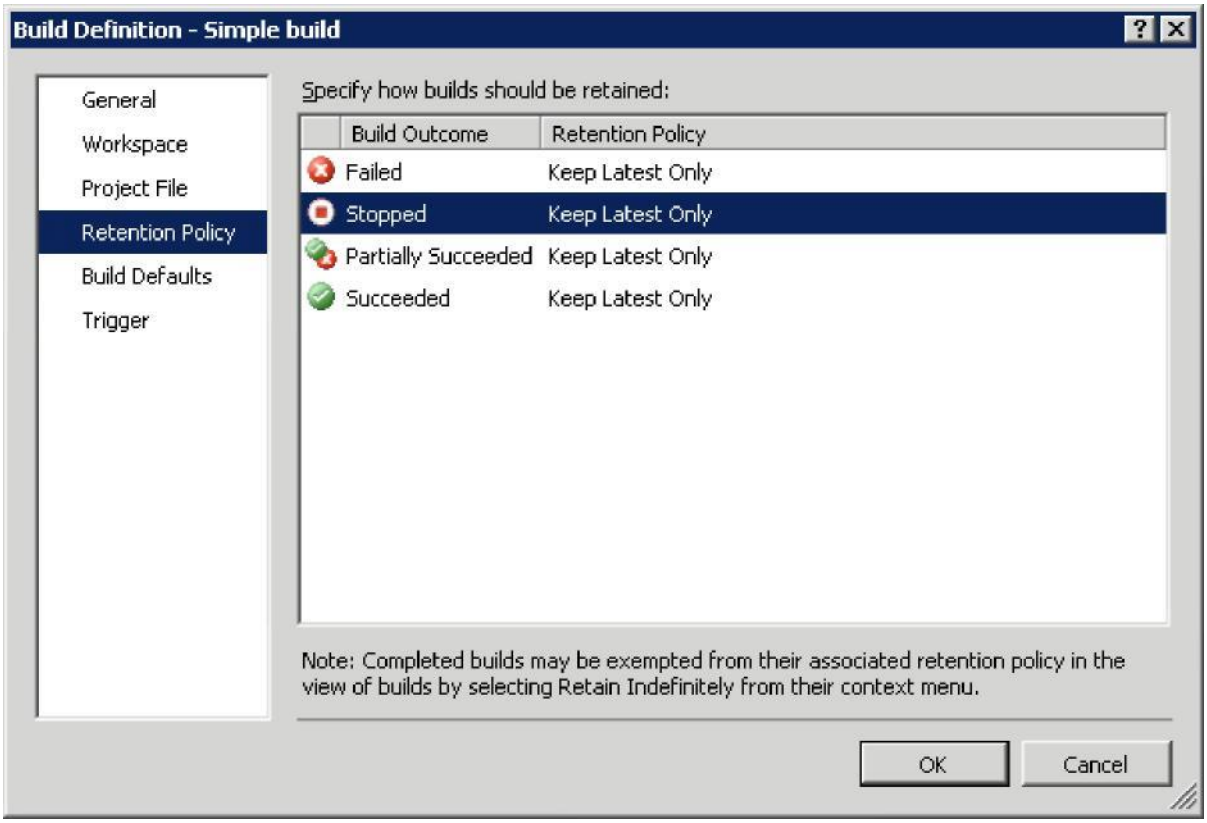


Рис. 40. Закладка Retention Policy.

Проведение сборки после внесения изменений наиболее эффективно в том случае, если участники проекта получают нотификации о том, что сборка была проведена. Для того чтобы этого добиться необходимо:

1. В контекстном меню проекта выбрать команду Project Alerts:

2. В списке событий, о которых нужно слать нотификации, выбрать “a build completes” и задать список адресов электронной почты, на которые нужно отправить сообщение (рис. 41):

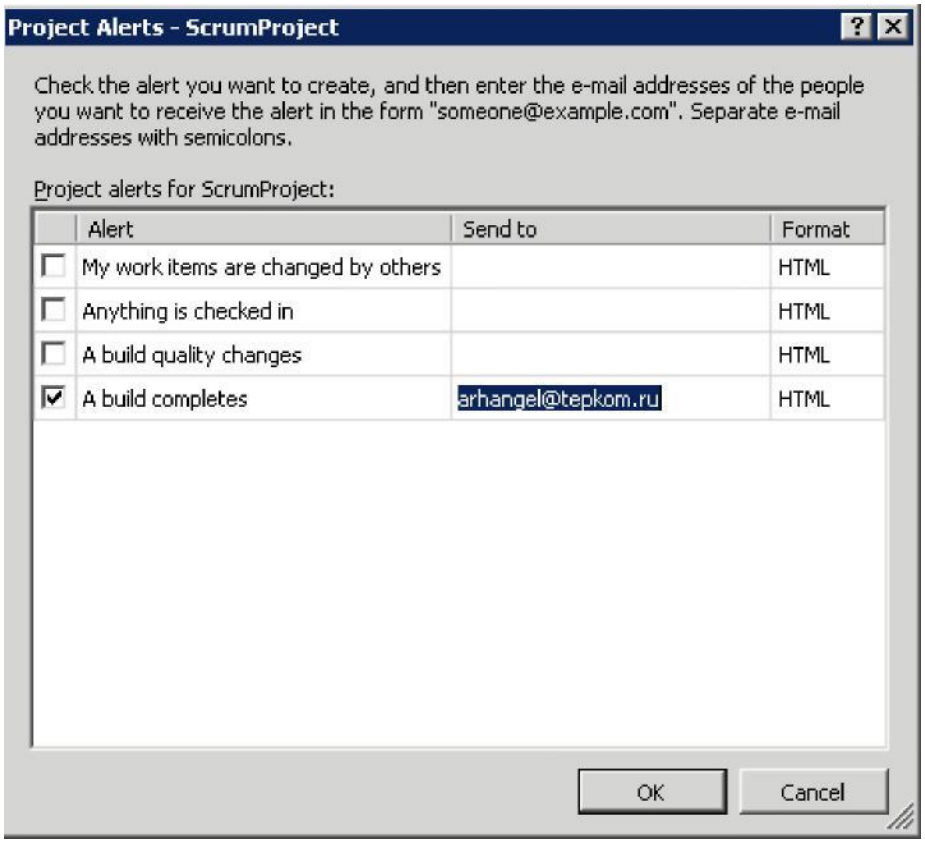


Рис. 41. Задание списка адресов.

После настройки простой сборки для запуска при внесении изменения необходимо проверить работу системы – внести некоторое изменение и дождаться сообщения о сборке.

Аналогичным образом можно настроить и автоматический запуск сложной сборки каждый день в определенное время.

Тема 6. Настройка шаблона процесса

На завершающем этапе, соответствующем окончанию спринта, команды должны провести ретроспективу своей деятельности и сформировать список возможных изменений в процессе и работе с TFS. Все замечания должны быть занесены в TFS как соответствующие элементы работы.

В рамках ретроспективы команда должна предложить некоторые изменения к элементам работы, вовлеченным в процесс, а затем и реализовать эти изменения.

Шаг 1. Ретроспектива

На ретроспективе команда в течение 20-30 минут обсуждает то, как прошел данный спринт, выделяя позитивные и негативные моменты, а также предложения по изменениям.

Все комментарии должны быть внесены в соответствующий элемент работы типа Sprint retrospective, а для каждого предложения по улучшению заведены элементы работы типа Sprint backlog item, а для каждого идентифицированного негативного момента, требующего устранения – элемент работы типа Impediment.

Результаты ретроспективы необходимо обсудить с хозяином продукта.

Шаг 2. Изменение элемента работы

На этапе ретроспективы команда должна выявить некоторые изменения в формате и жизненном цикле элементов работы, которые помогут повысить эффективность команды. На следующем шаге им нужно воплотить эти изменения в жизнь. Для этого необходимо:

1. Открыть тип элемента работы на редактирование с помощью команды меню Tools (рис. 42):

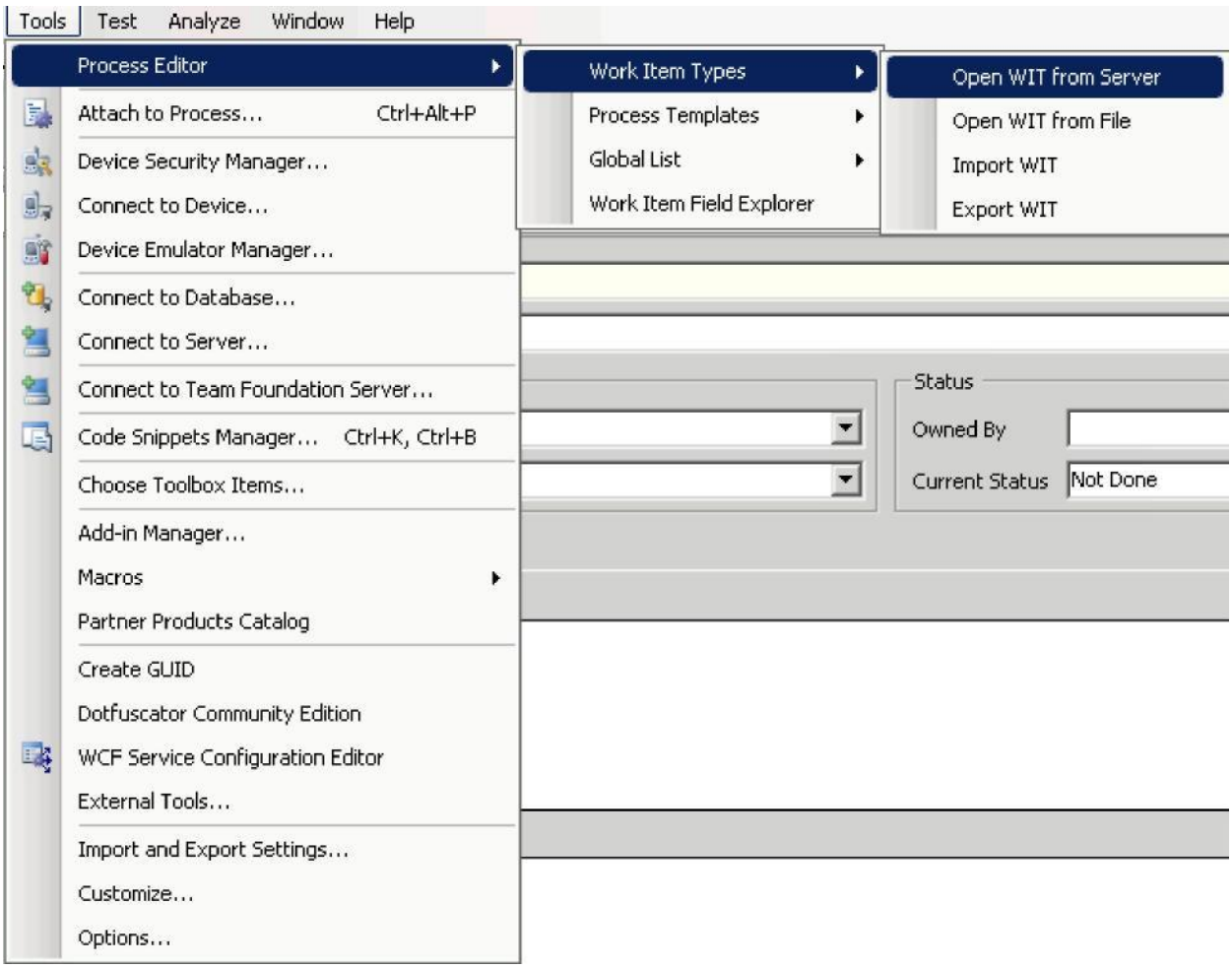


Рис. 42. Меню Tools.

2. Выбрать нужный элемент работы в открывшемся диалоге (рис. 43):

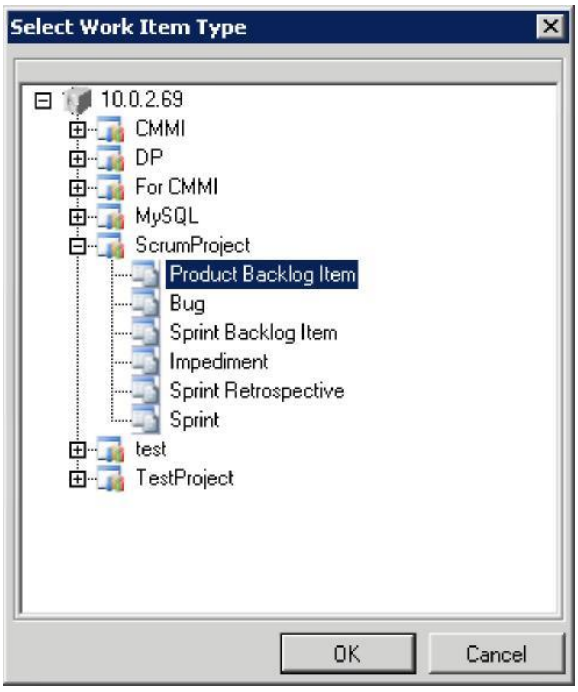


Рис. 43.

3. На закладке Fields добавить, удалить или изменить необходимые поля (рис.44):

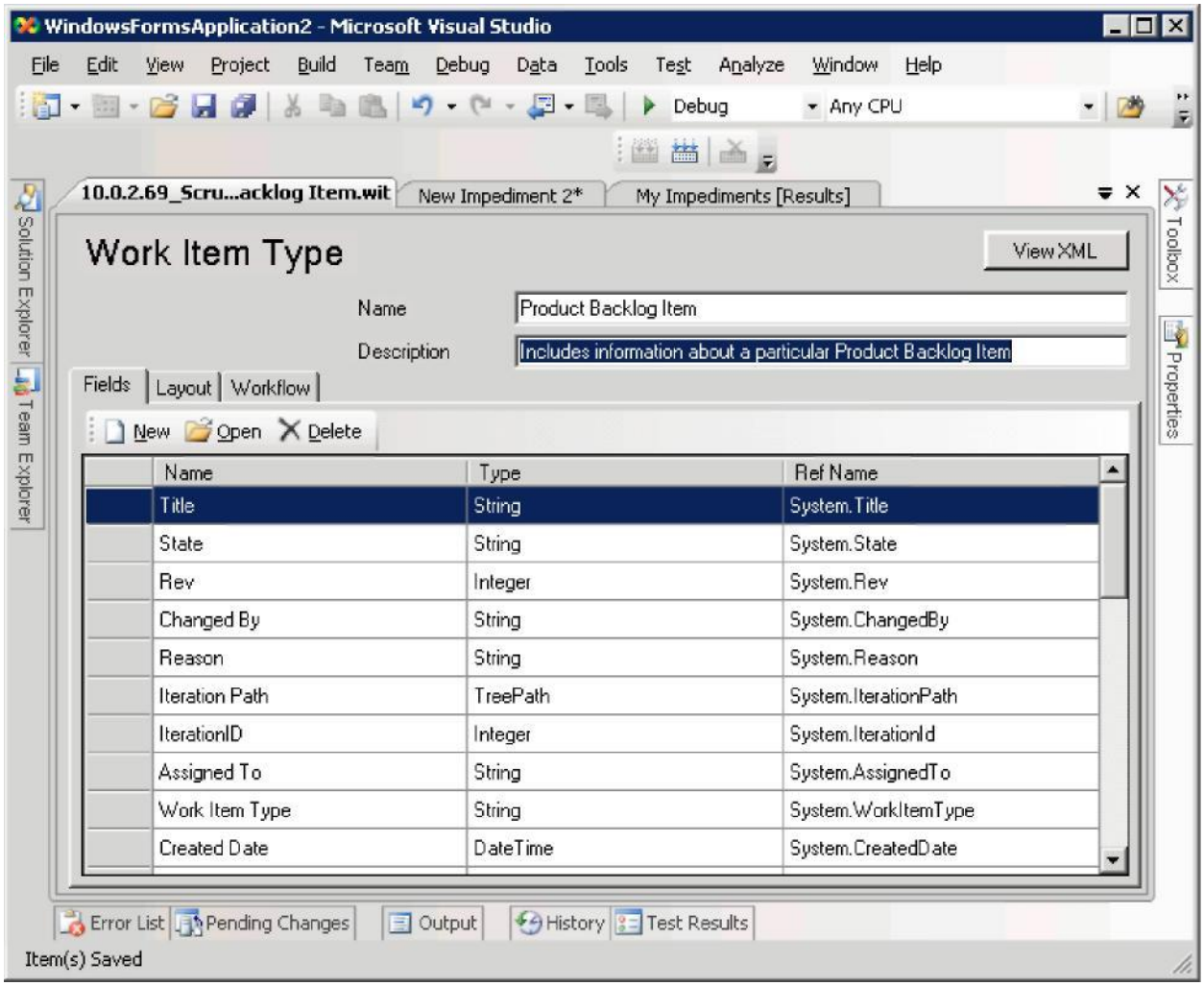


Рис. 44. Закладка Fields.

4. На закладке Layout изменить соответствующим образом визуальное представление элемента работы (рис.45):

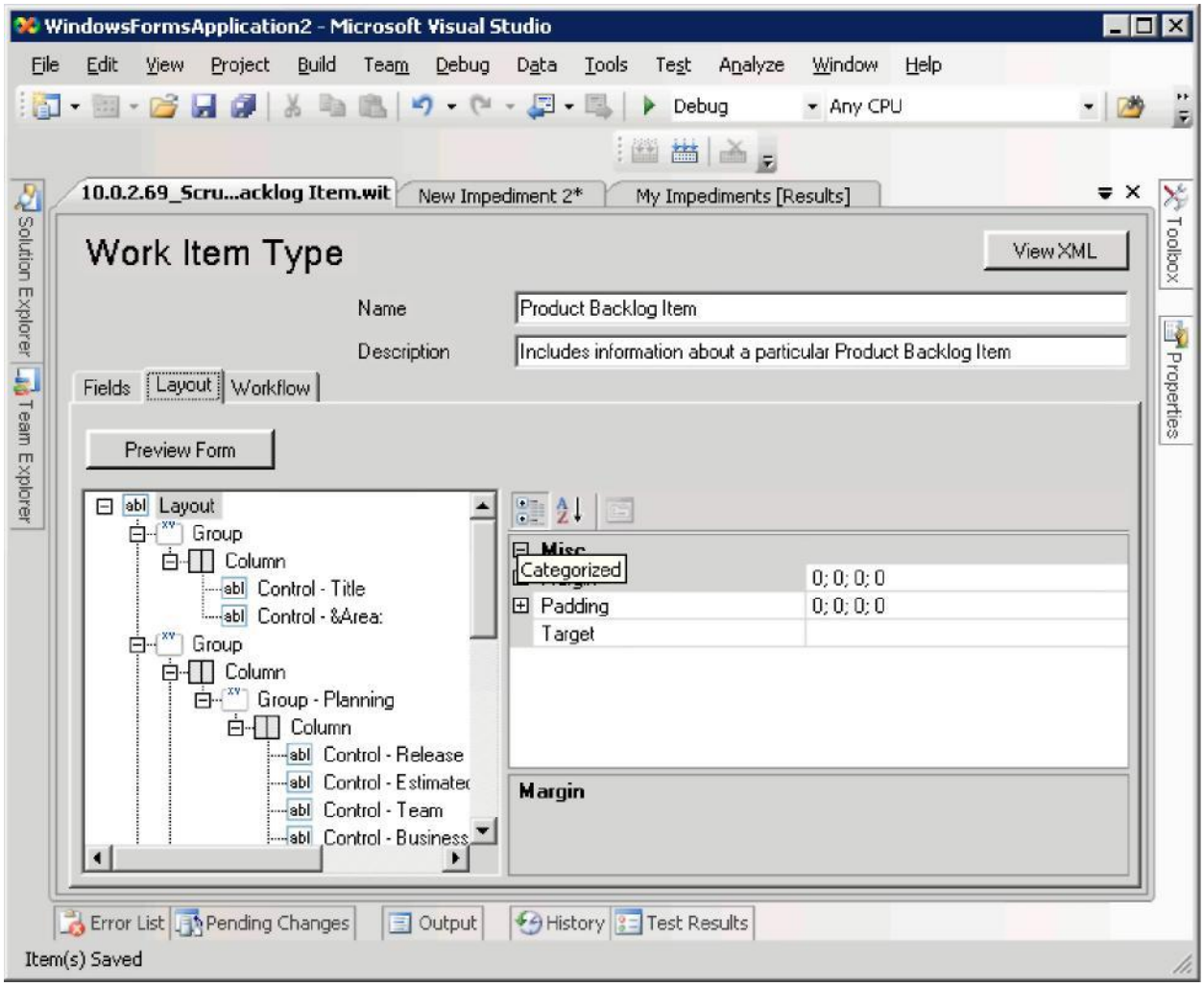


Рис. 45. Закладка Layout.

5. На закладке Workflow внести необходимые изменения в жизненный цикл элемента работы (рис. 46):

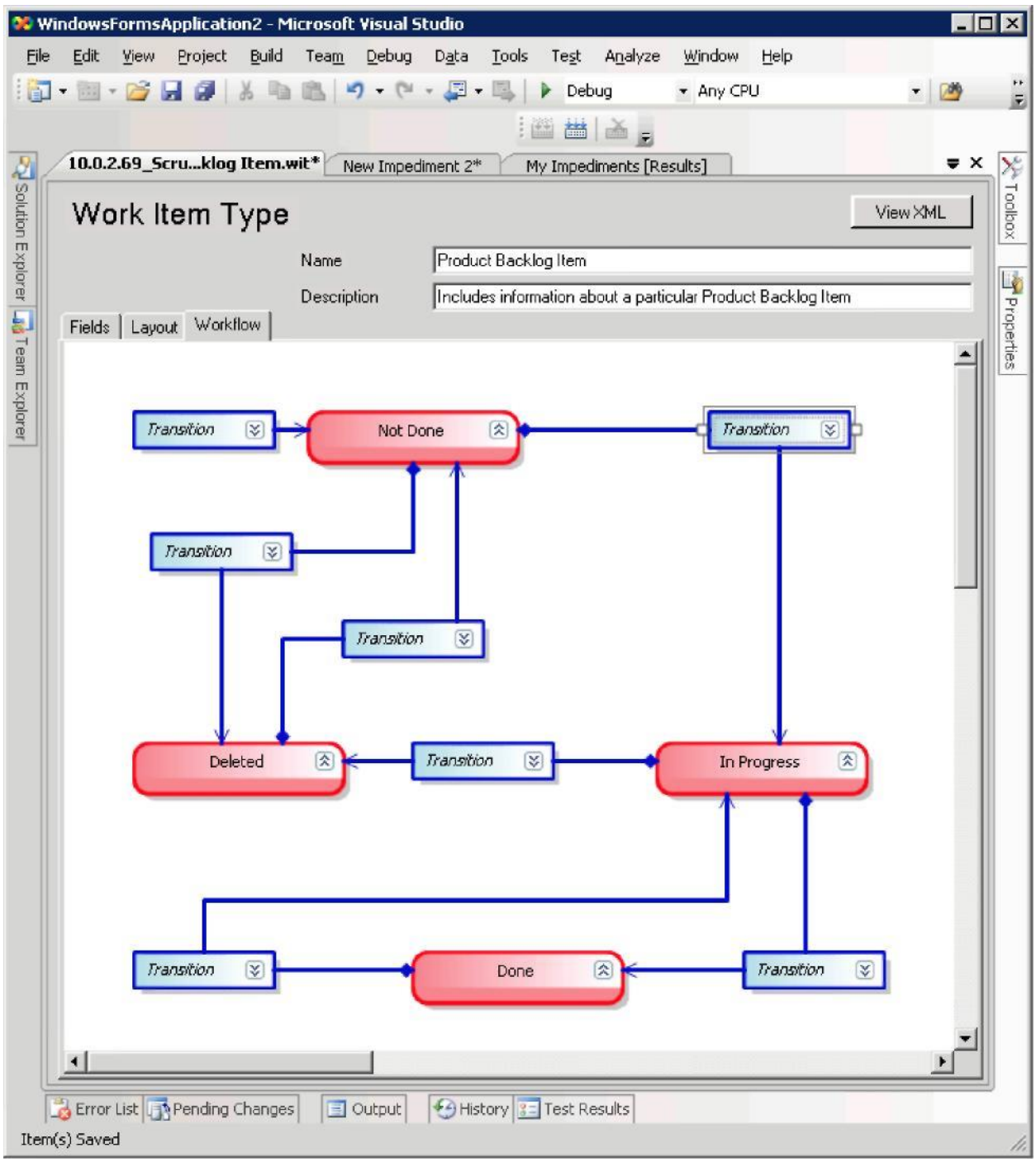


Рис. 46. Закладка Workflow.

После внесения всех изменений их нужно сохранить, а затем убедиться, что они применились к существующим и к вновь создаваемым элементам работы.

ЛИТЕРАТУРА

1. Джеф Сазерленд. Принципы и значение гибкой разработки. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/vstudio/dd997578(v=vs.100).aspx

2. Учебное руководство. Начало работы с Microsoft Visual Studio Team Foundation Server 2010. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/ru-ru/library/vstudio/dd286491%28v=vs.100%29.aspx

3. Scrum. [Электронный документ] – Режим доступа: http://msdn.microsoft.com/en-us/library/dd997796%28v=VS.100%29.aspx

4. Планирование и отслеживание проектов. – дополнительный материал по курсу «Методы и средства программной инженерии».

5. Алгоритмы и методы: Обратная польская запись (исходники). [Электронный документ] – Режим доступа: http://www.interface.ru/home.asp?artId=1492

6. Язык Си в примерах/Калькулятор выражений в обратной польской нотации. [Электронный документ] – Режим доступа: http://ru.wikibooks.org/wiki/'Язык Си в примерах/Калькулятор выражений в обратной польской нотации'/.

7. Кознов Д.В. Введение в программную инженерию. ИНТУИТ, [Электронный документ] – Режим доступа: http://www.intuit.ru/department/se/inprogeng/.