ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Процесс современной разработки программного обеспечения ориентирован на жизненный цикл программного продукта. Все существующие в настоящее время технологии, методики и стандарты напрямую или косвенно касаются или регламентируют этапы жизненного цикла, как по функциональному наполнению, так и по содержанию.

Процесс разработки программных систем тесно связан с областью управления проектами, потому что любой программный продукт является уникальным результатом. От организации этого процесса напрямую зависят основные характеристики выполнения программного проекта – сроки выполнения, запланированный бюджет, качество выпускаемого продукта.[25]

Но профессиональное управление проектами само по себе не может обеспечить достижение указанных характеристик. Немаловажную роль в этом играет архитектура программной системы, опыт и квалификация участников команды разработки, а также правильное документирование всех процессов разработки программного обеспечения.

Существуют различные определения технологии разработки программного обеспечения. К наиболее распространенным относятся следующие.

Технология разработки программного обеспечения (ТРПО) – это совокупность процессов и методов создания программного продукта.

Технология разработки программного обеспечения – это система инженерных принципов для создания экономичного ПО, которое надежно и эффективно работает в реальных компьютерах. Данное определение имеет частный характер, поскольку учитывает только две из шести характеристик качества ПО – надежность и эффективность. С учетом этого можно сформулировать более общее определение [5].

Технология разработки программного обеспечения – это система инженерных принципов для создания экономичного ПО с заданными характеристиками качества.

Любая технология разработки ПО базируется на некоторой методологии.

Под *методологией* понимается система принципов и способов организации процесса разработки программ. *Цель* методологии разработки ПО – внедрение методов разработки программ, обеспечивающих достижение соответствующих характеристик качества.

Тема 1
КРАТКИЕ ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ КУРСА «ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

Существует множество различных процессов для создания ПО. Тем не менее, технологий, рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного.

За несколько десятилетий эволюции аппаратное обеспечение значительно усовершенствовалось. Вычислительные мощности, которые еще десять-пятнадцать лет назад могли себе позволить лишь немногие научные учреждения и обслуживание которых требовало целого штата специалистов, сегодня доступны практически каждому инженеру. Однако эти мощности требуют соответствующего программного обеспечения. И именно в этой области, несмотря на то что аппаратные ресурсы стали значительно более доступны, наблюдаются значительные проблемы.[25]

Так, по данным американских исследователей, в 80-е только 14% проектов по созданию ПО завершались успешно. Но и сегодня - после нескольких десятилетий эволюции языков программирования, инструментальных средств разработки, при практически неограниченном (по сравнению с 70-ми и 80-ми) машинном времени - процент успешно завершенных проектов составляет всего 26%.

В СССР достижения в области производства ПО были значительно лучшими. Тому способствовали следующие объективные предпосылки:

- плановая организация производства оптимально сочеталась с каскадной моделью разработки ПО;
- контроль успешности проекта был ориентирован не на удовлетворение требований заказчика, а на удовлетворение изначально согласованного ТЗ;
- разработкой ПО занимались, как правило, высококвалифицированные специалисты в специализированных институтах;
- поскольку проекты в основном ориентировались на ВПК, бюджеты были фактически не ограниченными (по сегодняшним меркам).

Но по ряду причин советская школа разработки ПО прекратила свое развитие и многие достижения были утрачены. В рыночных условиях

(быстро меняющиеся требования, ограниченные бюджеты, ориентация на результат, острая конкуренция за высококвалифицированный персонал) использование старых наработок советской школы оказалось ограничено очень узкими областями.

1.1. Технологии

Термин «технология» – он подчеркивает аналогию между созданием программного продукта и промышленным производством. Он отражает современную тенденцию к вводу дисциплины, организации и инструментирования в такой творческий процесс, как программирование. Слово фиксирует ту точку зрения, что программирование, несмотря на интеллектуальность и творческий характер этой деятельности, нуждается в организации и регламентировании, наборе соглашений и правил, не говоря уже об инструментальном обеспечении. Сейчас это кажется тривиальным, но в 60-е годы такую точку приходилось отстаивать. Да и сейчас порой возникают трения на почве регламентирования деятельности разработчиков. Сам русский термин «технология программирования» был введен русским академиком Андреем Петровичем Ершовым. Он трактовал термин «программирование» в обобщенном виде и подразумевал все виды деятельности, выполняемые в ходе создания программных систем. На западе для определения этой деятельности использовался термин «engineering». Сейчас обобщённый термин, применимый к созданию программных средств, обозначают как «разработка» или «конструирование». Справедлива формула:

разработка = анализ + проектирование + программирование (кодирование) + тестирование + отладка

Иногда сюда также включают “сопровождение”. Чтобы подчеркнуть промышленно-производственный аспект, говорят о “технологии разработки” или “технологии конструирования”.

1.2. Этапы развития

Лишь в начале 90-х Британское сообщество вычислительной техники (British Computer Society) начало присваивать разработчикам программ звание инженера. В США только в 1998 году стало возможным хоть где-то зарегистрироваться в качестве профессионального инженера программного обеспечения. Но по-прежнему, даже в начале нынешнего века, общепризнанным остается тот факт, что разработке программного обеспечения не достает достаточно развитой научной базы. По

некоторым оценкам, 75% организаций, занимающихся разработкой программ, делают это на примитивном уровне.

С момента зарождения технология разработки программ испытала несколько подъемов в своем развитии. Один из них связан с публикацией письма Эдстера Дийкстры (Edsger Dijkstra) в Ассоциацию вычислительной техники, озаглавленного так: «О вреде использования операторов GOTO» (GOTO statement considered harmful, 1968). В те времена программы писались с активным использованием операторов безусловного перехода. Обращая внимание на недостатки таких программ, Дийкстра предложил свою концепцию структурного программирования, позволяющую избежать использования таких операторов. Концепция Дийкстры основывалась на том наблюдении Бема и Якоби (Flow Diagrams, Turning Machines and languages with only two formation rules, 1966), что для записи любой программы в принципе достаточно только трех конструкций управления – последовательного выполнения, ветвления и цикла. То есть теоретически необходимость в использовании операторов перехода отсутствует [24, 25].

Следующий шаг в развитии структурного программирования связан с введением аппарата функций, позволяющих разбивать структурную программу на обозримые по своим размерам части. При таком подходе программа пишется в терминах вызова функций верхнего уровня, которые реализуются при помощи функций более низкого уровня. Нисходящее программирование еще так же называли модульным программированием.

Структурным программам недоставало одного важного свойства – в их структуре непосредственно не отображалась сущность предметной области. Из-за этого было трудно их модифицировать в условиях изменяющихся требований. Позднее возникла парадигма объектной ориентированности. Она основана на использовании объектов, объединяющих в себе данные и функциональность. На ОО парадигме основаны многие современные языки и системы программирования.

1.3. Методы проектирования

Говорят, что Генри Форд совершил революцию в производстве автомобилей, когда заметил, что узлы автомобиля можно стандартизировать, так что при сборке автомобилей данной модели можно будет использовать любой экземпляр требуемого узла.

Столь же важным в настоящее время признается возможность при разработке одних приложений заимствовать идеи, архитектуру, проект и исходный код других приложений. Если приложения проектируются

таким образом, что различные их части могут быть использованы многократно, то в конечном итоге это приводит к уменьшению стоимости разработки приложений. Однако, чтобы это было возможным, приложения должны быть модульными. Модульность приложения, собственно, и означает, что оно состоит из легко идентифицируемых и заменяемых частей. По этой причине при правильном проектировании программного продукта особое внимание должно уделяться модульности, особенно на стадии разработки архитектуры.

К формальным методам проектирования относятся те методы, которые основаны на математике. Формальные методы помогают решить задачи обеспечения надежности программ. Они могут быть применены как при анализе требований для обеспечения точности формулировки требований, так и в процессе реализации для обеспечения соответствия кода программы сформулированным требованиям. Как правило формальные методы используют математику в ее логическом аспекте. В вычислительном же аспекте математика задействована в связи с использованием метрик, которые мы будем рассматривать далее.

Сегодня существует огромное количество различных процессов для создания ПО. Тем не менее, именно технологий, рассматривающих полный жизненный цикл проекта разработки ПО, сочетающих в себе научный подход, серьезную базу исследований и имеющих историю реального использования и адаптации, относительно немного. Из методологий и технологий, получивших определенное признание на данный момент, можно назвать следующие: Datarun, CMM, Microsoft Solution Framework (MSF), Oracle Method, Rational Unified Process (RUP), SADT (IDEFx).

Особое место в этом списке занимает технология компании Rational Software. В ее методологии применен наиболее современный процессно-ориентированный подход: так как разработка ПО является производством, то, как и на всяком производстве, при выявлении проблем в продукции (симптомов) необходимо корректировать процесс (устранять причины). Особенностью этой технологии является то, что в ее создании участвуют ведущие методисты в области разработки ПО, такие как Г. Буч (ООАП), Дж. Рамбо (OMT), А. Джекобсон (Objectory), внесшие весомый вклад в теорию и практику разработки современного ПО. Кроме того, следует заметить, что эта технология развивалась и проходила проверку с участием военного ведомства США [2, 24].

1.4. Этапы и элементы процесса разработки

В 80-е и 90-е в области разработки ПО преобладали две тенденции. Одна – это быстрый рост приложений, в том числе создаваемых для Web. Другая – расцвет инструментальных средств и парадигм (подходов к проектированию).

Несмотря на появление новых тенденций, основные этапы разработки ПО остались неизменными:

- Определение процесса разработки ПО;
- Управление проектом разработки;
- Описание целевого программного продукта;
- Проектирование продукта;
- Разработка продукта;
- Тестирование частей;
- Интеграция частей и тестирование продукта в целом;
- Сопровождение продукта.

Разработчики меняют последовательность проработки этих направлений. В реальности разработка ПО обычно определяется требуемым набором функций или сроком сдачи проекта. В результате, только хорошо организованные группы инженеров, владеющих методами разработки ПО, способны правильно построить работу. В противном случае разработчиков обычно ожидает хаос.

Система разработки ПО включает в себя 4 “П” (Персонал, процесс, проект, продукт)[15, 24].

Персонал – те, кем это делается. Команда разработчиков наилучшим образом работает, если каждый участник знает, что он должен делать, и имеет определенные обязанности. Другая сторона аспекта персонала – это заинтересованные в проекте лица: заказчиками, пользователи и инвесторы. В любом производстве результаты определяются используемой технологией. В силу специфичности производства ПО (практически нулевая стоимость тиражирования, очень быстрое моральное старение и т.д.) технология его создания сильно зависит от качества команды разработчиков, поэтому должна включать в себя организационный и управленческий аспекты.

Процесс – способ, которым это делается. Выделяют: водопадный процесс, итеративный процесс, XP. Индивидуальный процесс разработки (Personal Software Process), командный процесс разработки (Team Software Process). Модель зрелости возможностей (Capability Maturity Model) для оценки возможностей команды разработчиков.

Проект – совокупность действий, необходимая для создания артефакта. Проект включает контакт с заказчиком, написание

документации, проектирование, написание кода и тестирование продукта.

Продукт – это не только программное обеспечение, но и все составляющие его артефакты. Под артефактами понимается объектные модули, исходный код, документация, результаты тестов и измерений продуктивности.

Качество – приложения должны удовлетворять заранее определенному уровню качества. Для достижения требуемого уровня качества применяются следующие методы:

- инспектирование (процесс проверки качества, ориентированный на команды разработчиков. Он применяется на всех этапах разработки);
- формальные методы (доказательство правильности – математическое или логическое);
- тестирование;
- методы управления проектом [3, 24].

1.5. Инструментарий технологии программирования

Инструментарий технологии программирования – совокупность программ и программных комплексов, обеспечивающих технологию разработки, отладки и внедрения создаваемых программных продуктов.

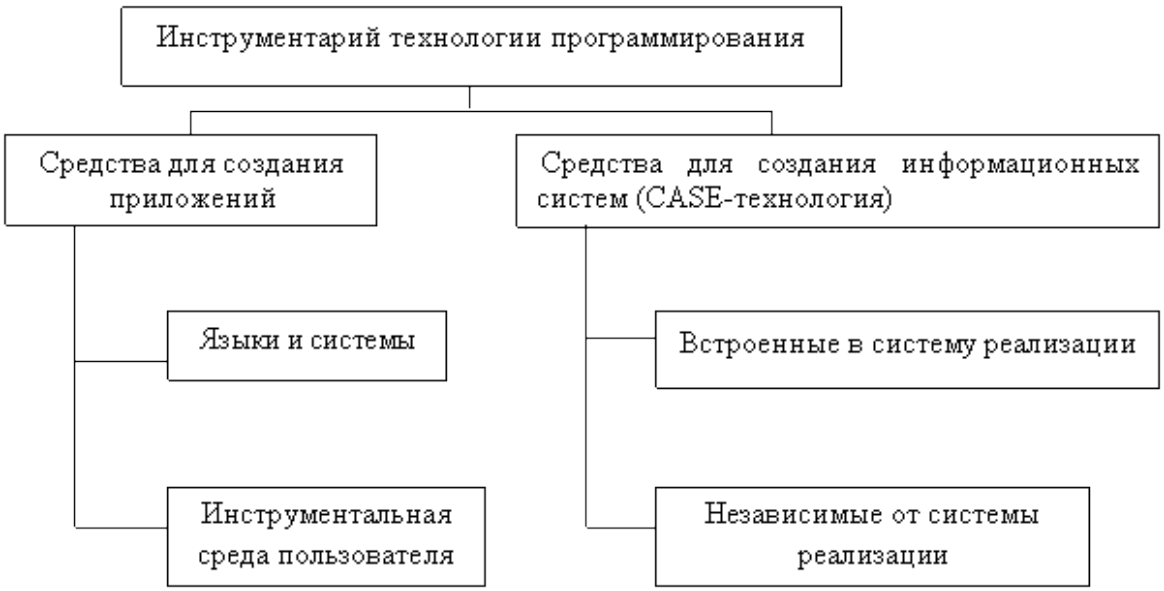

Рис.1.1. Группы программных продуктов

1.5.1. Средства для создания приложений

Средства для создания приложений – локальные средства, обеспечивающие выполнение отдельных видов работ по созданию программ, делятся на:

- языки и системы программирования;
- инструментальная среда пользователя.

Язык программирования – формализованный язык для описания алгоритма решения задачи на компьютере. Они делятся на классы:

- *машинные языки* – языки программирования, воспринимаемые аппаратной частью компьютера (машинные коды);
- *машинно-ориентированные языки* – языки программирования, которые отражают структуру конкретного типа компьютера (ассемблеры);
- *алгоритмические языки* – не зависящие от архитектуры компьютера языки программирования для отражения структуры алгоритма (Паскаль, бейсик, Фортран и др.);
- *процедурно–ориентированные языки* – языки программирования, где имеется возможность описания программы как совокупности процедур (подпрограмм).
- *проблемно–ориентированные языки* – предназначены для решения задач определенного класса (Lisp);

Другой классификацией языков является их деление на языки, ориентированные па реализацию основ структурного программирования, основанного на модульной структуре программного продукта и типовых управляющих структурах алгоритмов обработки данных различных программных модулей, и объектно-ориентированные языки, поддерживающие понятие объектов, их свойств и методов обработки [4, 24].

Системы программирования включают:

- компилятор (транслятор);
- интегрированную среду разработки программ (не всегда);
- отладчик;
- средства оптимизации кода программ;
- набор библиотек;
- редактор связей;
- сервисные средства (утилиты) (для работы с библиотеками, текстовыми и двоичными файлами);
- справочные системы;
- систему поддержки и управления продуктами программного комплекса.

Компилятор транслирует всю программу без ее выполнения. **Трансляторы** (интерпретаторы) выполняют пооперационную обработку и выполнение программы. **Отладчики** – специальные программы, предназначенные для трассировки и анализа выполнения других программ. **Трассировка** – это обеспечение выполнения в пооператорном варианте.

Инструментальная среда пользователя – это специальные средства, встроенные в пакеты прикладных программ, такие, как:

- библиотека функций, процедур, объектов и методов обработки;
- макрокоманды;
- клавишные макросы;
- языковые макросы;
- конструкторы экранных форм и объектов;
- генераторы приложений;
- языки запросов высокого уровня;
- конструкторы меню и др.

Интегрированные среды разработки программ объединяют набор средств для их комплексного применения на технологических этапах создания программы.

1.5.2. Средства для создания информационных систем (Case–технология)

CASE-технология (CASE – Computer-Aided System Engineering) – программный комплекс, автоматизирующий весь технологический процесс анализа, проектирования, разработки и сопровождения сложных программных систем. Средства CASE-технологий делятся на:

- встроенные в систему реализации – все решения по проектированию и реализации привязки к выбранной СУБД;
- независимые от системы реализации – все решения по проектированию ориентированы на унификацию (определение) начальных этапов жизненного цикла программы и средств их доку-ментирования, обеспечивают большую гибкость в выборе средств реализации.

Основное достоинство CASE-технологии – это поддержка коллективной работы над проектом за счет возможности работы в локальной сети разработчиков, экспорта (импорта) любых фрагментов проекта, организованного управления проектами. В некоторых CASE-системах поддерживается создание каркаса программ и создание полного продукта.

Вопросы и задания для самоконтроля

1. Что такое технология разработки ПО?
2. Что явилось предпосылкой становления дисциплины «Технология разработки ПО»? Что явилось причиной стремительного развития ПО?
3. Чем отличаются программа и программное обеспечение?
4. Достаточно ли при работе над проектом большой программной системы быть компетентным в области вычислительной техники и программировании. Почему?
5. Может ли большая программная система быть отлажена до конца и почему?
6. При каких условиях созданный программный комплекс может быть назван программным продуктом?
7. Что такое системное программное обеспечение?
8. Что такое инструментарий технологии программирования?

Тема 2
ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Жизненный цикл ПО определяется как период времени, который начинается с момента принятия решения о необходимости ПО и заканчивается в момент его полного изъятия из эксплуатации. Основным нормативным документом, регламентирующим состав ЖЦ ПО, является международный стандарт ISO/IEC 12207: 2008 «System and software engineering – Software life cycle processes». Данный стандарт, используя устоявшуюся терминологию, устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии. Стандарт определяет процессы, виды деятельности и задачи, которые используются при приобретении программного продукта или услуги, а также при поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов. (его российский аналог ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств»). Каждый процесс (процесс – совокупность взаимосвязанных действий, преобразующий некоторые входные данные в выходные) разделен на набор действий, каждое действие – на набор задач. В соответствии с ГОСТ Р ИСО/МЭК 12207-2010 различные виды деятельности, которые могут выполняться в течение жизненного цикла программных систем в семь групп процессов:

1) процессы соглашения – 2;
2) процессы организационного обеспечения проекта – 5;
3) процессы проекта – 7;
4) технические процессы – 11;
5) **процессы реализации программных средств – 7;**
6) процессы поддержки программных средств – 8;
7) процессы повторного применения программных средств – 3.

Каждый из процессов жизненного цикла в пределах этих групп описывается в терминах цели и желаемых выходов, список действий и задач, которые необходимо выполнять для достижения этих результатов. В рамках курса «Технология разработки программного обеспечения» детально будет рассмотрена группа процессов, описывающих реализацию программных средств [5].

2.1. Процессы реализации программных средств

Процессы реализации программных средств используются для создания конкретного элемента системы (составной части), выполненного в виде программного средства. Эти процессы преобразуют заданные характеристики поведения, интерфейсы и ограничения на реализацию в действия, результатом которых становится системный элемент, удовлетворяющий требованиям, вытекающим из системных требований. [ГОСТ Р ИСО/МЭК 12207-2010].

Примечание. В курсе «Технология разработки программного обеспечения» программное средство рассматривается ни как составная часть системы, а как независимое автономное программное обеспечение, состоящее из программных модулей [17].

2.1.1. Процесс реализации

При реализации проекта необходимо осуществлять следующие виды деятельности в соответствии с принятыми в организации политиками и процедурами в отношении процесса реализации программных средств:

1) Если не оговорено в контракте, разработчик должен определить или выбрать модель жизненного цикла, соответствующую области применения, размерам и сложности проекта. Модель жизненного цикла должна содержать стадии, цели и выходы каждой стадии. Виды деятельности и задачи процесса реализации программных средств должны быть выбраны и отражены в модели жизненного цикла. *Подробно существующие модели и методологии будут рассмотрены во второй теме текущего документа.* Эти виды деятельности и задачи могут пересекаться или взаимодействовать друг с другом, могут выполняться итеративно или рекурсивно. В идеальном случае рассматриваемые виды деятельности и задачи выполняются и решаются с использованием определенной организационной модели жизненного цикла.
2) Исполнитель должен:
   - документировать результаты в соответствии с процессом менеджмента программной документации;
   - передавать результаты в процесс менеджмента конфигурации программных средств и выполнять управление изменениями в соответствии с ним;
   - документировать, решать проблемы и снимать несоответствия, найденные в программных продуктах и задачах в соответствии с процессом решения проблем в программных средствах;

- выполнять поддержку процессов в соответствии с контрактом;
- устанавливать базовые линии и соединять элементы конфигурации в сроки, определенные приобретающей стороной и поставщиком.

3) Исполнитель должен выбирать, адаптировать и применять те стандарты, методы, инструментарий и языки программирования (если не оговорено в контракте), которые документально оформлены, являются подходящими и установлены организацией для выполнения деятельности в рамках процесса реализации программных средств и поддерживающих процессов.

4) Исполнитель должен разрабатывать планы проведения действий процесса реализации программных средств. Планы должны включать в себя конкретные стандарты, методы, инструментарий, действия и обязанности, связанные с разработкой и квалификацией всех требований, включая безопасность и защиту. При необходимости могут разрабатываться отдельные планы. Эти планы должны документироваться и выполняться.

5) При разработке или сопровождении программных продуктов могут применяться не поставляемые элементы. Однако должно гарантироваться, что функционирование и сопровождение поставляемых программных продуктов после поставки приобретающей стороне не зависит от таких элементов; другими словами, эти элементы следует также рассматривать как поставляемые.

Результатом процесса является создание программной составной части, удовлетворяющей как требованиям к архитектурным решениям, что подтверждается посредством верификации, так и требованиям правообладателей, что подтверждается посредством валидации.

В результате успешного осуществления процесса реализации программных средств:

1) определяется стратегия реализации;
2) определяются ограничения по технологии реализации проекта;
3) изготавливается программная составная часть;
4) программная составная часть упаковывается и хранится в соответствии с соглашением о ее поставке.

Процесс реализации программных средств включает в себя несколько специальных процессов более низкого уровня:

1) процесс анализа требований к программным средствам;
2) процесс проектирования архитектуры программных средств;
3) процесс детального проектирования программных средств;
4) процесс конструирования программных средств;

5) процесс комплексирования программных средств;
6) процесс квалификационного тестирования программных средств.

2.1.2. Процесс анализа требований к программным средствам

Цель процесса анализа требований к программным средствам заключается в установлении и документировании требований к программному обеспечению. В результате успешного выполнения процесса определяется перечень требований к функциональным модулям программного обеспечения и их интерфейсам, определяются приоритеты реализации требований, требования к ПО оцениваются по стоимости, графикам работ и техническим воздействиям. Подробно о способах выявления и видах требований будет описано в третьей теме текущего документа [6, 17].

2.1.3. Процессы проектирования (детального проектирования) архитектуры программных средств

Цель процесса заключается в обеспечении проекта для программных средств, которые реализуются и могут быть проверены относительно требований сформулированных в ходе процесса анализа требований. В рамках процесса исполнитель осуществляет преобразование выявленных требований в архитектуру, которая описывает верхний уровень структуры программного средства и идентифицирует программные компоненты. Исполнитель должен разработать проект, описывающий внешние и внутренние интерфейсы, структуру и метод доступа к базе данных (БД), так же исполнитель оформляет предварительные версии пользовательской документации и требования к предварительному тестированию.

В результате успешной реализации процесса разрабатывается проект архитектуры программных средств, определяются внутренние и внешние интерфейсы, устанавливается соответствие между требованиями и программным проектом. Подробно о методах проектирования программных средств рассказывается в четвертой теме.

2.1.4. Процесс конструирования программных средств

Целью процесса является создание исполняемых программных блоков (модулей), которые созданы на основе архитектурного проекта. При реализации процесса исполнитель разрабатывает документацию на каждый программный модуль и базу данных, процедуры и данные для

тестирования модулей и базы данных. В данном процессе также происходит тестирование модулей исполнителем, гарантируя, что они удовлетворяют требованиям. В ходе тестирования ведется журнал тестирования, фиксирующий информацию о соответствующих работах (когда проводится, какой тест, кем проводится и т.п.). Неожиданные или некорректные результаты тестов могут записываться в специальной подсистеме ведения отчетности по сбоям. Исполнитель должен оценивать программный код и результаты испытаний, учитывая следующие критерии:

1) прослеживаемость к требованиям и проекту программных элементов;
2) внешнюю согласованность с требованиями и архитектурным проектом для программных модулей;
3) тестовое покрытие модулей;
4) соответствие методов кодирования и используемых стандартов;
5) осуществимость функционирования и сопровождения.

В результате успешного осуществления процесса определяется критерий верификации для всех модулей относительно требований, разработка программных модулей, тестирование [17].

2.1.5. Процесс комплексирования программных средств

В ходе процесса комплексирования программных средств осуществляется объединение функциональных программных модулей, создание интегрированных программных элементов, согласованных с проектом программного средства, которые демонстрируют, что функциональные и нефункциональные требования к программному средству удовлетворяются.

Для каждого модуля программного средства исполнитель должен разработать план комплексирования для объединения программных модулей. План должен включать в себя требования к тестированию, данные для тестирования, обязанности и графики работ. Так же исполнителю необходимо объединить программные модули в соответствии с планом комплексирования и разработать комплекс тестов. Результаты комплексирования и тестирования должны быть оформлены документально. Любое изменение в пользовательском интерфейсе и функциональности сопровождается обновлением пользовательской документации по мере необходимости [17].

2.1.6. Процесс квалификационного тестирования программного средства

Цель процесса квалификационного тестирования программного средства заключается в подтверждении того, что комплектованный программный продукт удовлетворяет установленным требованиям. В рамках процесса исполнитель должен провести квалификационное тестирование (согласно требованиям). Исполнителю необходимо провести оценку проекта, кода, тестов и их результаты, а также пользовательской документации, учитывая следующие критерии:

1) тестовое покрытие требования к программному средству;
2) соответствие с ожидаемыми результатами;
3) осуществимость функционирования и сопровождения.

После успешного тестирования программный продукт готов к передаче заказчику. После чего в действие вступают процессы поддержки программного средства [17].

Заключение

Эталонная модель, описанная в стандарте, не представляет конкретного подхода к осуществлению процесса, как и не определяет модель жизненного цикла системы (программного средства), методологию или технологию. В зависимости от принятой методологии разработки ПО или модели ЖЦ в конкретной компании используется различный набор процессов. В следующей теме рассмотрены основные модели и методологии разработки программного обеспечения.

Вопросы и задания для самоконтроля

1. Понятие жизненного цикла ПО. Что понимается под процессом жизненного цикла? Назовите основные группы процессов согласно ГОСТ Р ИСО/МЭК 12207-2010.
2. Основная цель процесса анализа требований к программным средствам. Что является результатом успешного осуществления процесса?
3. Процесс реализации. Какие виды деятельности и задачи входят в состав процесса реализации?
4. Процесс проектирования архитектуры программных средств. Что является результатом успешной реализации процесса. Что понимается под базовой линией?

Тема 3
МОДЕЛИ И МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Процесс жизни любой системы или программного продукта может быть описан посредством модели жизненного цикла, состоящей из стадий. Модели могут использоваться для представления всего жизненного цикла от замысла до прекращения применения или для представления части жизненного цикла, соответствующей текущему проекту. Модель жизненного цикла представляется в виде последовательности стадий, которые могут перекрываться и (или) повторяться циклически в соответствии с областью применения, размером, сложностью, потребностью в изменениях и возможностях. Каждая стадия описывается формулировкой цели и выходов. Процессы и действия жизненного цикла отбираются и исполняются на этих стадиях для полного удовлетворения цели и результатам каждой стадии. Различные организации могут использовать различные стадии в пределах жизненного цикла. Однако каждая стадия реализуется организацией, ответственной за эту стадию, с надлежащим рассмотрением информации, имеющейся в планах жизненного цикла и решениях, принятых на предшествующих стадиях. Аналогичным образом организация, ответственная за текущую стадию, ведет записи принятых решений и записи допущений, относящихся к последующим стадиям данного жизненного цикла [2, 25].

3.1. Модели жизненного цикла программного обеспечения

Под моделью жизненного цикла ПО понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Модель ЖЦ зависит от спецификации, масштаба и сложности проекта и спецификации условий, в которых система создается и функционирует. Модель ЖЦ ПО включает в себя: стадии, результаты выполнения работ на каждой стадии, ключевые события – точки завершения работ и принятия решений. Модель ЖЦ любого конкретного ПО определяет характер процесса его создания, который представляет собой совокупность упорядоченных во времени, взаимосвязанных и объединенных в стадии работ, выполнение которых необходимо и достаточно для создания ПО, соответствующего заданным требованиям. Под стадией понимается часть процесса создания ПО, ограниченная определенными временными рамками и

заканчивающаяся выпуском конкретного продукта (моделей, программных компонентов, документации), определяемого заданными для данной стадии требованиями. На каждой стадии могут выполнятся несколько процессов, определённых в стандарте ГОСТ Р ИСО/МЭК 12207-2010, и наоборот один и тот процесс может выполняться на различных стадиях. Соотношение между стадиями и процессами также определяется используемой моделью ЖЦ ПО. Далее рассмотрим модели и их классификации [2, 25].

3.1.1. Каскадная модель

Первой моделью, получившей широкую известность и действительно структурирующей процесс разработки, является каскадная (водопадная) модель. Каждая стадия каскадной модели заканчивается получением некоторых результатов, которые служат в качестве исходных данных для следующей стадии. Требования к разрабатываемому ПО, определенные на стадии формирования требований, строго документируются в виде технического задания и фиксируются на все время разработки проекта.

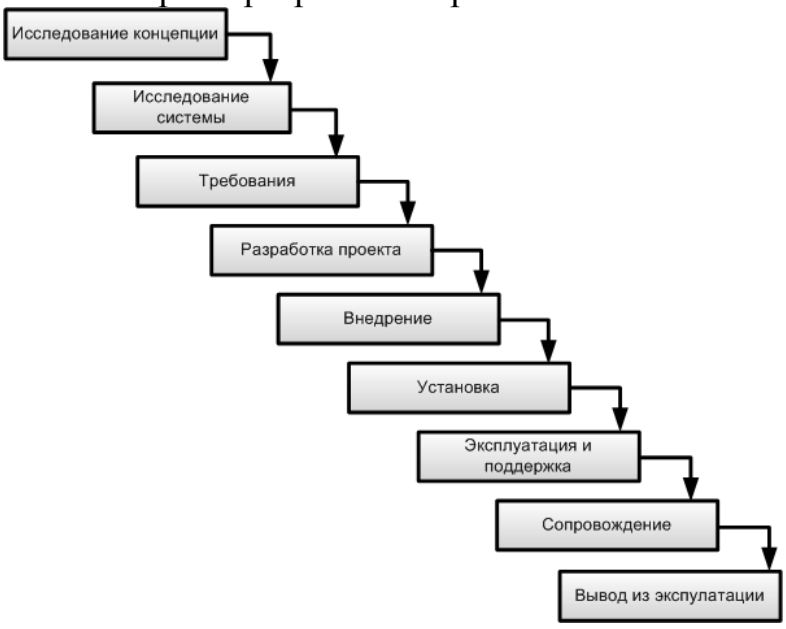

Рис. 3.1 Стандартная водопадная модель

Преимущества применения каскадной модели заключаются в следующем:

- на каждой стадии формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;

– выполняемые в логичной последователь стадии работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадная модель может использоваться при создании ПО, для которого в самом начал разработки можно достаточно точно и полно сформулировать все требования. В то же время этот подход обладает рядом недостатков, вызванных прежде всего тем, чьл реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему.

Спустя непродолжительное время после появления на свет каскадная модель была доработана Уинстом Ройсом с учетом взаимозависимости этапов и необходимости возврата на предыдущие ступени, что может быть вызвано, например, неполнотой требований или ошибками в формировании задания. Процесс создания ПО носит как правило, итерационный характер: результаты очередной стадии часто вызывают изменения в проектных решениях, выработанных на более ранних стадиях. Таким образом, постоянно возникает потребность в возврате к предыдущим стадиям и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимает вид, изображенный на рисунке *.1*.

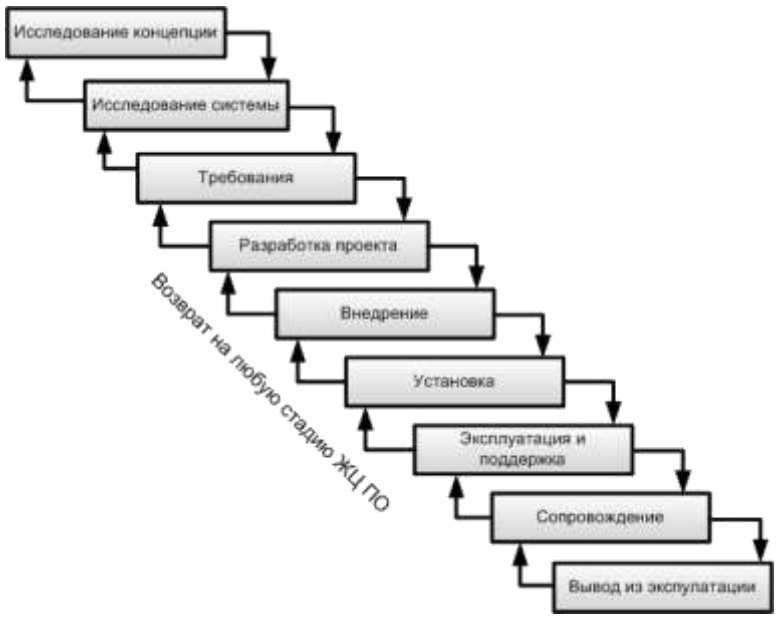

Рис. 3.1. Модифицированная водопадная модель

Наиболее распространенным результатом каскадного похода к разработке ПО является поздняя неудача. Кажется, что проекты выполняются нормально, но только до тех пор, пока работы не вступят в

завершающий этап, и тогда выясняется, что потребители недовольны созданным продуктом [25].

3.1.2. V-образная модель, как разновидность каскадной модели

Основной принцип V-образной модели заключается в том, что детализация проекта возрастает при движении слева направо, одновременно с течением времени, и ни то, ни другое не может повернуть вспять. Итерации в проекте производятся по горизонтали, между левой и правой сторонами буквы.

V-модель – вариация каскадной модели, в которой задачи разработки идут сверху вниз по левой стороне буквы V, а задачи тестирования – вверх по правой стороне буквы V. Внутри V проводятся горизонтальные линии, показывающие, как результаты каждой из стадий разработки влияют на развитие системы тестирования на каждой из стадий тестирования. Модель базируется на том, что приемо-сдаточные испытания основываются, прежде всего, на требованиях, системное тестирование – на требованиях и архитектуре, комплексное тестирование – на требованиях, архитектуре и интерфейсах, а компонентное тестирование – на требованиях, архитектуре, интерфейсах и алгоритмах.

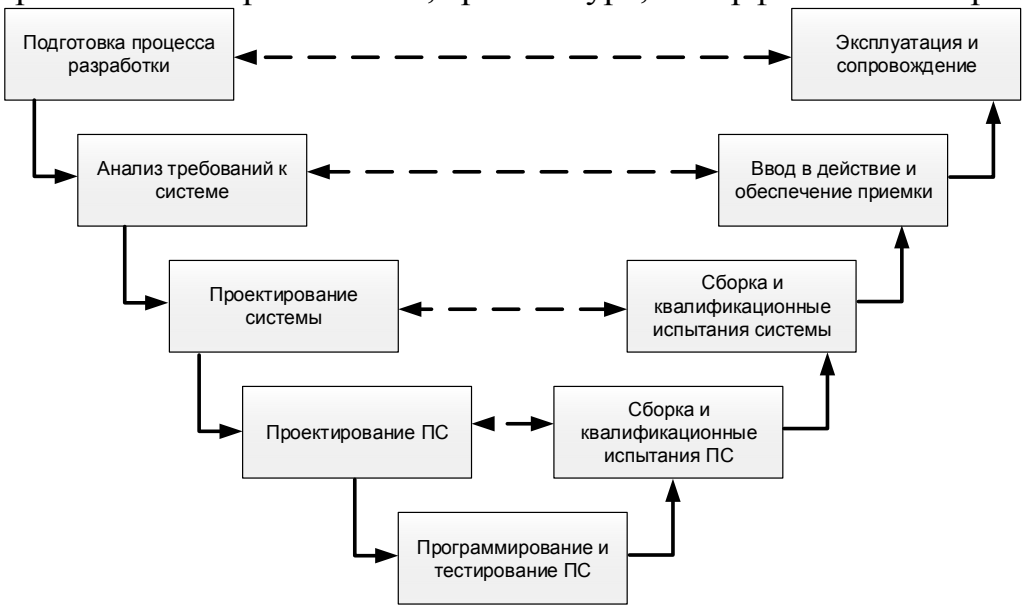

Рис. 3.2 – V-образная модель

Особенностью данной модели является разбиение стадий на три логических этапа: проектирование (детализация требований), реализация, тестирование.

V-модель дает организациям и проектным группам руководство по выполнению и завершению проектов последовательным и воспроизводимым образом. Применение принципов V-модели гарантирует выявление и фиксацию требований пользователей. Утвержденные требования могут быть переведены в функции готового приложения, (и) приложение отражает требования пользователей [2, 25].

3.1.3. Итеративный инкрементный подход к разработке (эволюционная модель)

3.1.3.1 Итеративная модель

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.

Шансы успешного создания сложной системы будут максимальными, если она реализуется в серии небольших шагов и если каждый шаг заключает в себе четко определенный результат, а также возможность возврата к результатам предыдущей успешной итерации, в случае неудачи. Перед тем, как пустить в дело все ресурсы, предназначенные для создания ПО, разработчик имеет возможность получать обратную связь из реального мира (заказчиков, пользователей) и исправлять возможные ошибки в проекте.

Итеративная модель подразумевает возможность не только сборки работающей (с точки зрения результатов тестирования) версии системы ***- прототипа****, но и её развертывания в реальных операционных условиях с анализом откликов пользователей для определения содержания и планирования следующей итерации.* Поскольку на каждом шаге мы имеем работающую систему, то можно:

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

3.1.3.2 Инкрементная модель

Идея, лежащая в основе инкрементной модели, состоит в том, что программную систему следует разрабатывать по принципу приращений, так, чтобы разработчик мог использовать данные, полученные при разработке более ранних версий ПО. Новые данные получаются как в ходе разработки ПО, так и в ходе его использования, где это возможно. Ключевые этапы этого процесса – простая реализация подмножества требований к программе и совершенствование модели в серии последовательных релизов до тех пор, пока не будет реализовано ПО во всей полноте. В ходе каждой итерации организация модели изменяется, и к ней добавляются новые функциональные возможности.

Для организации инкрементной разработки обычно выбирается характерный временной интервал, например, неделя. Затем в течение этого интервала происходит обновление проекта: добавляется новая документация как текстовая, так и графическая, расширяется набор тестов, добавляются новые программные коды и т. д. Теоретически шаги разработки могут выполняться и параллельно, но такой процесс очень сложно скоординировать. Инкрементная разработка проходит лучше всего, если следующая итерация начинается после того, как обновление всех артефактов в предыдущей итерации закончено, и существенно хуже, если время, требуемое на обновление артефактов, значительно превышает выбранный интервал [27].

В результате каждой итерации получается работающее, но не полнофункциональное ПО, которое еще не является программным продуктом и не подлежит распространению. В результате каждой итерации создается версия некоторой части ПО. Необходимо заметить, но как правило на каждой итерации определяются и реализуются новые требования, некоторые итерации могут быть целиком посвящены усовершенствованию существующей программы, например, с целью повышения ее производительности.

Вывод

С точки зрения структуры жизненного цикла эволюционную модель называют итеративной. С точки зрения развития продукта – инкрементальной. **Опыт 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). Значимость эволюционной модели на основе организации итераций особо проявляется в снижении

неопределенности с завершением каждой итерации. В свою очередь, снижение неопределенности позволяет уменьшить риски. Рисунок 3.3 иллюстрирует идею эволюционной модели, предполагая, что итеративному разбиению может быть подвержен не только жизненный цикл в целом, включающий перекрывающиеся стадии – формирование требований, проектирование, конструирование и т.п., но и каждая стадия может, в свою очередь, разбиваться на уточняющие итерации, связанные, например, с детализацией структуры декомпозиции проекта – например, архитектуры модулей системы [25].

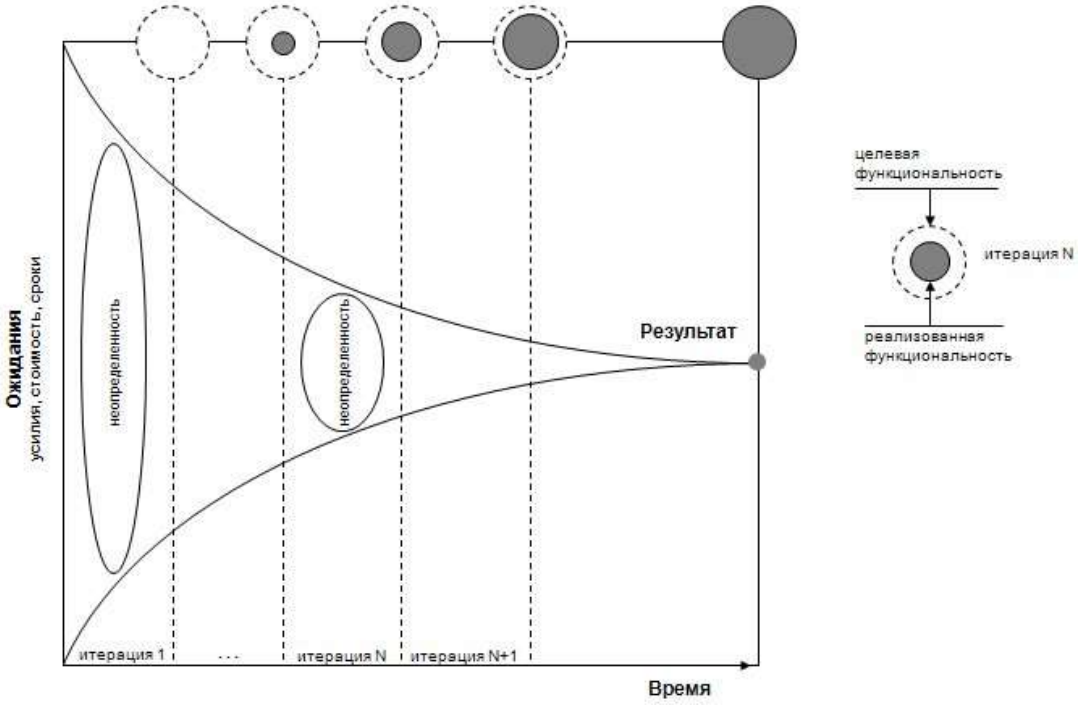

Рисунок 3.3 – Снижение неопределенности и инкрементальное расширение функциональности при итеративной организация жизненного цикла.

3.1.4. Спиральная модель, как разновидность эволюционной модели

В середине 1980-х годов Барри Боэм предложил свой вариант итерационной модели итеративной модели под названием «Спиральная модель». При использовании спиральной модели прикладное ПО создается в несколько итераций (витков спирали) методом прототипирования. Создание прототипов осуществляется в несколько итераций, или витков спирали. Каждая итерация соответствует созданию фрагмента или версии ПО, на ней уточняются цели и характеристики проекта, оценивается качество полученных результатов и планируются

работы следующей итерации. На каждой итерации производится тщательная оценка риска превышения сроков и стоимости проекта, чтобы определить необходимость выполнения еще одной итерации, степень полноты и точности понимания требований к системе, а также целесообразность прекращения проекта [2, 27].

Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Боэм формулирует 10 наиболее распространенных (по приоритетам) рисков:

- дефицит специалистов;
- нереалистичные сроки и бюджет;
- реализация несоответствующей функциональности;
- разработка неправильного пользовательского интерфейса;
- «золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей;
- непрекращающийся поток изменений;
- нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию;
- недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами;
- недостаточная производительность получаемой системы;
- «разрыв» в квалификации специалистов разных областей знаний.

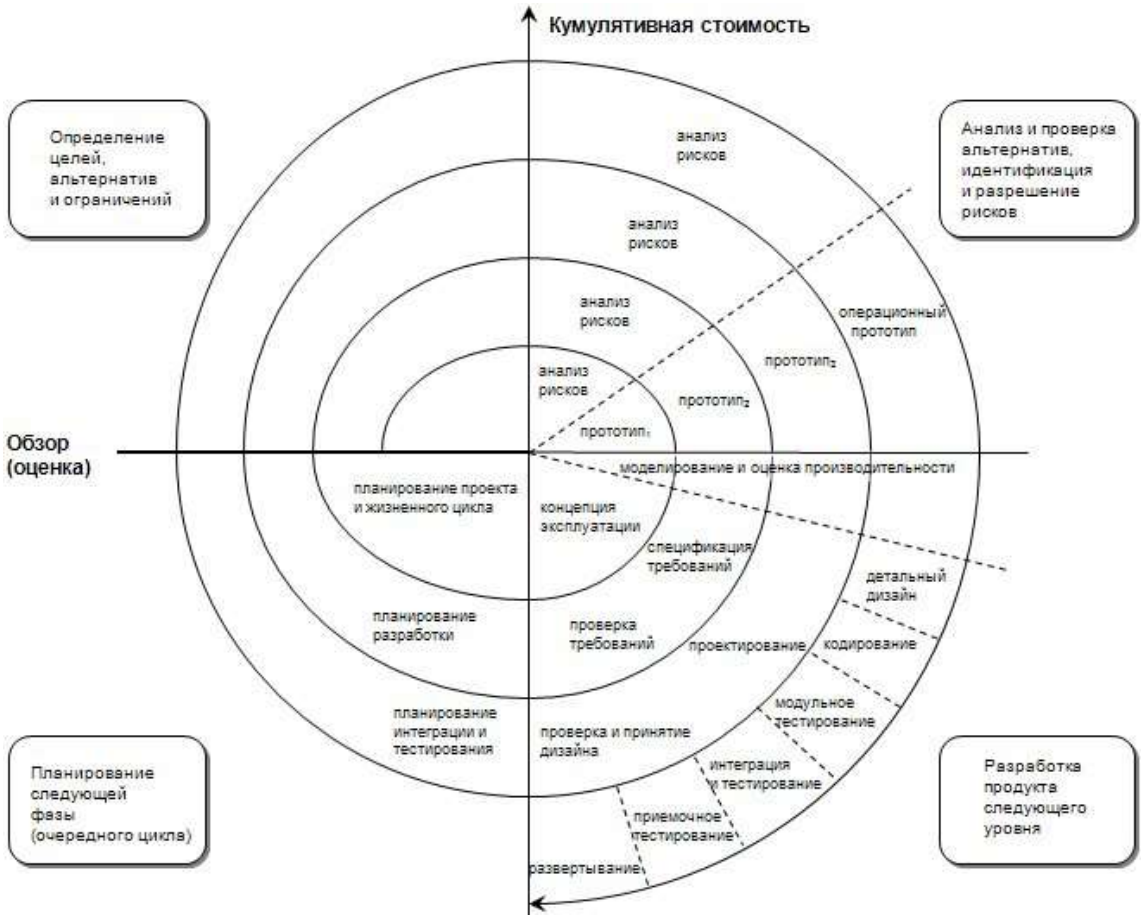

Рис. 3.4 Оригинальная спиральная модель жизненного цикла разработки по Боэму

Основная проблема спирального цикла – определение момента перехода на следующую стадию. Для её решения необходимо ввести временные ограничения на каждую из стадий ЖЦ. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статических данных, полученных в предыдущих проектах, и личного опыта разработчика.

Достоинствами спиральной модели являются: ускорение разработки (ранее получение результата за счет прототипирования), постоянное участие заказчика в процессе разработки, разбиение большого объема работы на небольшие части, снижение риска [2, 25].

Заключение

Естественное развитие каскадной и эволюционной модели привело к их сближению и появлению современных подходов - методологий, которые по существу представляет собой рациональное сочетание вышеописанных моделей. Различные варианты эволюционного подхода реализованы в большинстве современных технологий и методов: Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Agile и другие.

3.2. Методологии разработки ПО

Методологии представляют собой ядро теории управления разработкой ПО. К существующей классификации в зависимости от используемой в ней модели жизненного цикла (каскадные и эволюционные) добавилась более общая классификация на прогнозируемые и адаптивные методологии.

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

Адаптивные (гибкие) методологии нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. Когда меняются требования, команда разработчиков тоже меняется. Команда, участвующая в адаптивной разработке, с трудом может предсказать будущее проекта. Существует точный план лишь на ближайшее время. Более удаленные во времени планы существуют лишь как декларации о целях проекта, ожидаемых затратах и результатах. Среди адаптивных методологий: (Scrum, Crystal, Extreme Programming, Adaptive Software Development, DSDM, Feature Driven Development, Lean software development). Рассмотрим самые основные и популярные методологии [26].

3.2.1. RUP (Rational Unified Process)

Один из самых известных процессов, использующих итеративную модель разработки – RUP. Он был создан во второй половине 1990-х годов в компании Rational Software. Термином RUP обозначает как методологию, так и продукт компании IBM (ранее Rational) для управления процессом разработки. Методология RUP описывает абстрактный общий процесс, на основе которого организация или

проектная команда должна создать специализированный процесс, ориентированный на ее потребности.

Основные характеристики:

- разработка требований, для описания требований в RUP используются прецеденты использования (use cases). Полный набор прецедентов использования системы вместе с логическими отношениями между ними называется моделью прецедентов использования. Каждый прецедент использования – это описание сценариев взаимодействия пользователя с системой, полностью выполняющего конкретную пользовательскую задачу. Согласно RUP все функциональные требования должны быть представлены в виде прецедентов использования.
- итеративная разработка, проект RUP состоит из последовательности итераций с рекомендованной продолжительностью от 2 до 6 недель. Перед началом очередной итерации определяется набор прецедентов использования, которые будут реализованы к её завершению.

3.2.1.1 Архитектура

Можно сказать, что RUP – ориентированная на архитектуру методология. Считается, что реализация и тестирование архитектуры системы должны начинаться на самых ранних стадиях проекта. RUP использует понятие исполняемой архитектуры (executable architecture) – основы приложения, позволяющей реализовать архитектурно значимые прецеденты использования. Основы исполняемой архитектуры должны быть реализованы как можно раньше. Это позволяет оценить адекватность принятых архитектурных решений и внести необходимые коррективы еще в начале проекта. Таким образом, для первых нескольких итераций необходимо выбирать прецеденты, которые требуют реализации большей части архитектурных компонентов.

RUP поощряет использование визуальных средств для анализа и проектирования. Как правило, используется нотация и, соответственно, средства моделирования UML (такие как Rational Rose). Модель предметной области документируется в виде диаграммы классов, модель прецедентов использования – при помощи диаграммы прецедентов, взаимодействие компонентов системы между собой описывается диаграммой последовательности и т д.

3.2.1.2 Жизненный цикл проекта

Жизненный цикл проекта RUP состоит из четырех фаз. Последовательность этих фаз фиксирована, но число итераций, необходимых для завершения каждой фазы, определяется индивидуально для каждого конкретного проекта. Фазы RUP нельзя отождествлять с фазами водопадной модели – их назначение и содержание принципиально различны.

Начало (Inception)

Стадия «начало» обычно состоит из одной итерации. В ходе выполнения этой стадии необходимо:

- определить видение и границы проекта;
- создать экономическое обоснование;
- идентифицировать большую часть прецедентов использования и подробно описать несколько ключевых прецедентов;
- найти хотя бы одно возможное архитектурное решение;
- оценить бюджет, график и риски проекта.

Если после завершения первой итерации заинтересованные лица приходят к выводу о целесообразности выполнения проекта, проект переходит в следующую стадию. В противном случае проект может быть отменен или проведена еще одна итерация стадия «начало».

Проектирование (Elaboration)

В результате выполнения этой стадии на основе требований и рисков проекта создается основа архитектуры системы. Проектирование может занимать до двух-трех итераций или быть полностью пропущенным (если в проекте используется архитектура существующей системы без изменений). Целями этой фазы являются:

- детальное описание большей части прецедентов использования;
- создание оттестированной (при помощи архитектурно значимых прецедентов использования) базовой архитектуры;
- снижение основных рисков и уточнение бюджета и графика проекта.

В отличие от каскадной модели, основным результатом этой стадии является не множество документов со спецификациями, а действующая система с 20-30% реализованных прецедентов использования [25].

Построение (Construction)

В этой стадии (длящейся от двух до четырех итераций) происходит разработка окончательного продукта. Вовремя ее выполнения создается основная часть исходного кода системы и выпускаются промежуточные демонстрационные прототипы.

Внедрение (Transition)

Целями стадии «внедрения» являются проведение бета-тестирования и тренингов пользователей, исправление обнаруженных дефектов, развертывание системы на рабочей площадке, при необходимости – миграция данных. Кроме того, на этой стадии выполняются задачи, необходимые для проведения маркетинга и продаж.

Стадия «внедрения» занимает от одной до трех итераций. После ее завершения проводится анализ результатов выполнения всего проекта: что можно изменить для улучшения эффективности в будущих проектах.

Рабочий процесс

В терминах RUP участники проектной команды создают так называемые артефакты (work products), выполняя задачи (tasks) в рамках определенных ролей (roles). Артефактами являются спецификации, модели, исходный код и т.п. Задачи разделяются по девяти процессным областям, называемым дисциплинами (discipline). В RUP определены шесть инженерных и три вспомогательные дисциплины. В них входят:

- **Бизнес-моделирование (Business Modeling**) – исследование и описание существующих бизнес-процессов заказчика, а также поиск их возможных улучшений.
- **Управление требованиями (Requirements Management)** – определение границ проекта, разработка функционального дизайна будущей системы и его согласование с заказчиком.
- **Анализ и проектирование (Analysis and Design)** – проектирование архитектуры системы на основе функциональных требований и ее развитие на протяжении всего проекта.
- **Реализация (Implementation)** – разработка, юнит-тестирование и интеграция компонентов системы.
- **Тестирование (Test)** – поиск и отслеживание дефектов в системе, проверка корректности реализации требований.
- **Развертывание (Deployment)** – создание дистрибутива, установка системы, обучение пользователей.
- **Управление конфигурациями и изменениями (Configuration and Change Management)** – управление версиями исходного кода и документации, процесс обработки запросов на изменение (Change requests).
- **Управление проектом (Project Management)** – создание проектной команды, планирование фаз и итераций, управление бюджетом и рисками.

– **Среда (Environment)** – создание инфраструктуры для выполнения проекта, включая организацию и настройку процесса разработки.

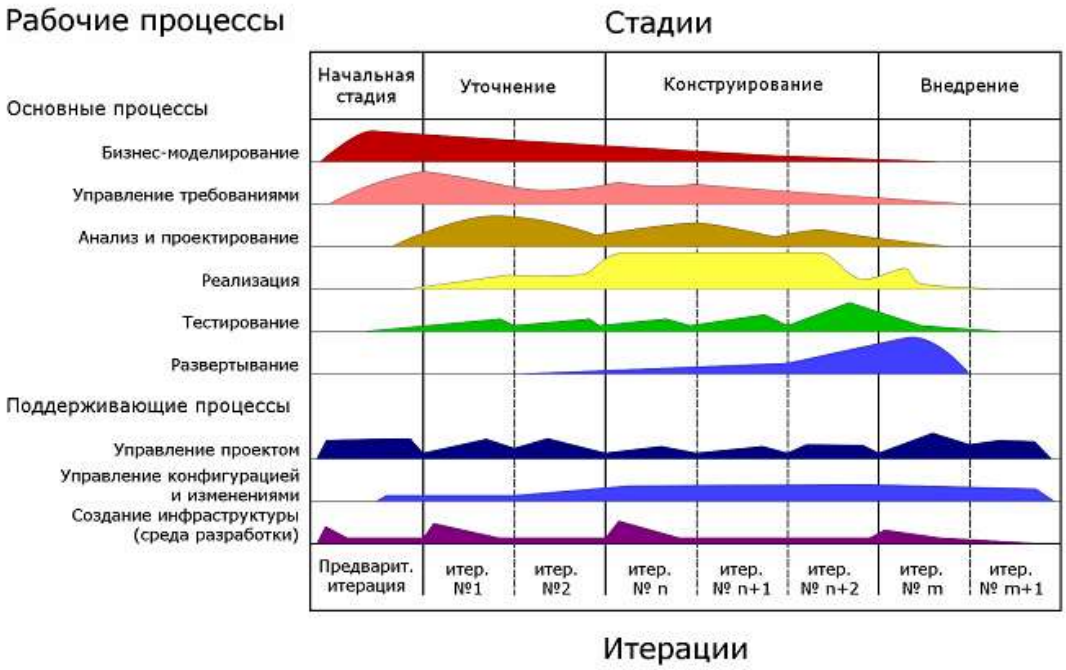

Рис. 3.5 Распределение усилий при выполнении проекта

Заключение

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

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

3.2.2. Microsoft Solutions Framework (MSF)

Данная методология описывает подход и организацию работы при создании программных продуктов. Подробно про методологию MSF вы

можете прочитать в переводе Microsoft Solutions Frameworks for Agile Software Development, которая входит в поставку Microsoft Team Foundation Server [23].

3.2.3. Scrum

Scrum предоставляет эмпирический подход к разработке ПО. Этот процесс быстр, адаптивен, умеет подстраиваться и отличен от каскадной модели. Scrum основан на повторяющихся циклах, это делает его более гибким и предсказуемым.

Для начала определим роли, которые участвуют в процессе: Scrum мастер (Scrum Master), Владелец продукта (Product Owner), Команда (Team).

Scrum Мастер - самая важная роль в методологии. Scrum Мастер отвечает за успех Scrum в проекте. Как правило, эту роль в проекте играет менеджер проекта или лидер команды (Team Leader). Важно подчеркнуть, что Scrum Мастер не раздает задачи членам команды. В Scrum команда является самоорганизующейся и самоуправляемой.

Основные обязанности Scrum Мастера таковы:

- создает атмосферу доверия,
- участвует в митингах в качестве фасилитатора - человека, обеспечивающий успешную групповую коммуникацию
- устраняет препятствия
- делает проблемы и открытые вопросы видимыми
- отвечает за соблюдение практик и процесса в команде

Scrum Мастер отслеживает прогресс команды при помощи Sprint Backlog, отмечая статус всех задач в спринте. Scrum Мастер может также помогать заказчику создавать список задач для команды

Product Owner - это человек, отвечающий за разработку продукта. Как правило представитель заказчика для заказной разработки. Владелец продукта - это единая точка принятия окончательных решений для команды в проекте, именно поэтому это всегда один человек, а не группа или комитет.

Команда (Team) - в методологии Scrum команда является самоорганизующейся и самоуправляемой. Команда берет на себя обязательства по выполнению объема работ на спринт перед Владельцем продукта. Работа команды оценивается как работа единой группы. В Scrum вклад отдельных членов проектной команды не оценивается, так как это разваливает самоорганизацию команды.

Размер команды ограничивается размером группы людей, способных эффективно взаимодействовать лицом к лицу. Типичные

размер команды - 7 плюс минус 2. Команда в Scrum кроссфункциональна. В нее входят люди с различными навыками - разработчики, аналитики, тестировщики. Нет заранее определенных и поделенных ролей в команде, ограничивающих область действий членов команды. Команда состоит из инженеров, которые вносят свой вклад в общий успех проекта в соответствии со своими способностями и проектной необходимостью.

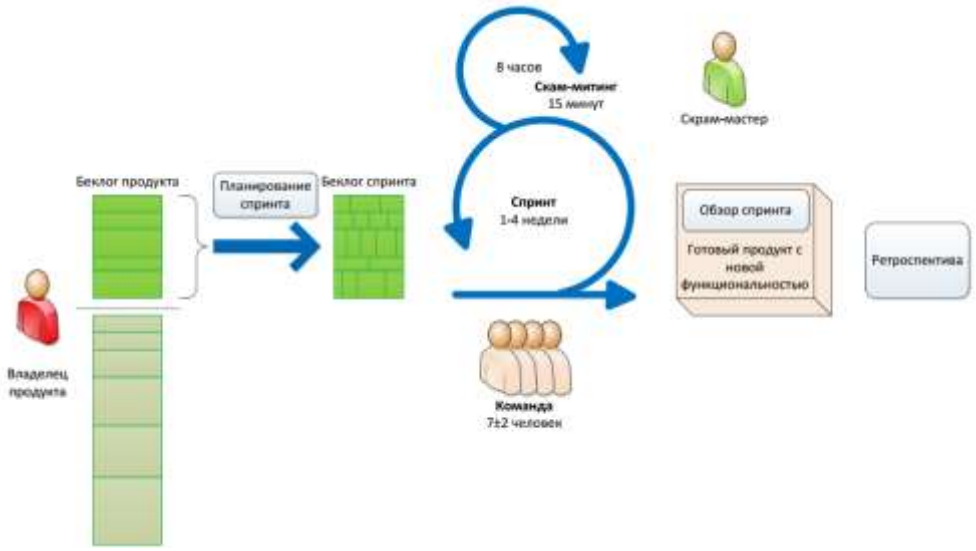

Рис. 3.6 Scrum

В основе лежат короткие ежедневные встречи – Scrum и циклические 30-дневные встречи, называемые спринтом. Результатом спринта является готовый продукт, который можно передавать заказчику (по крайней мере, система должна быть готова к показу заказчику) [25, 26].

Короткие спринты обеспечивают быструю обратную связь проектной команды с заказчиком. Заказчик получает возможность гибко управлять системой, оценивая результат спринта и предлагая улучшения к созданной функциональности. Такие улучшения попадают в список имеющихся на данный момент бизнес-требований и технических требований к системе (Product Backlog), приоритезируются наравне с прочими требованиями и могут быть запланированы на следующий (или на один из следующих) спринтов. Каждый спринт представляет собой маленький «водопад». В течение спринта делаются все работы по сбору требований, дизайну, кодированию и тестированию продукта.

Цель спринта должна быть фиксированным. Это позволяет команде давать обязательства на тот объем работ, который должен быть сделан в

спринте. Это означает, что Sprint Backlog не может быть изменен никем, кроме команды.

3.2.4. Экстремальное программирование (eXtreme Programming)

Методология XP, разработанная Кентом Беком (Kent Beck), Уордом Каннингемом (Ward Cunningham) и Роном Джеффрисом (Ron Jeffries), является сегодня одной из самых популярных гибких методологий. Она описывается как набор практик: игра в планирование, короткие релизы, метафоры, простой дизайн, переработки кода (refactoring), разработка «тестами вперед», парное программирование, коллективное владение кодом, 40-часовая рабочая неделя, постоянное присутствие заказчика и стандарты кода.

Интерес к XP рос снизу вверх – от разработчиков и тестировщиков, замученных тягостным процессом, документацией, метриками и прочим формализмом. Они не отрицали дисциплину, но не желали бессмысленно соблюдать формальные требования и искали новые быстрые и гибкие подходы к разработке высококачественных программ.

При использовании XP тщательное предварительное проектирование ПО заменяется, с одной стороны, постоянным присутствием в команде заказчика, готового ответить на любой вопрос и оценить любой прототип, а с другой – регулярными переработками кода (так называемый рефакторинг). Основой проектной документации считается тщательно прокомментированный код. Очень большое внимание в методологии уделяется тестированию. Как правило, для каждого нового метода сначала пишется тест, а потом уже разрабатывается собственно код метода до тех пор, пока тест не начнет выполняться успешно. Эти тесты сохраняются в наборах, которые автоматически выполняются после любого изменения кода.

Хотя парное программирование и 40-часовая рабочая неделя и являются, возможно, наиболее известными чертами XP, но все же носят вспомогательный характер и способствуют высокой производительности разработчиков и сокращению количества ошибок при разработке [26].

3.2.5. Crystal Clear

Легковесная гибкая методология, созданная Алистером Коуберном, которая предназначена для небольших команд в 6-8 человек для разработки некритичных бизнес-приложений. Как и все гибкие методологии, Crystal Clear больше опирается на людей, чем на процессы

и артефакты. Crystal Clear использует семь методов/практик, три из которых являются обязательными:

- **частая поставка продукта;**
- **улучшения через рефлексию;**
- **личные коммуникации;**
- чувство безопасности;
- фокусировка;
- простой доступ к экспертам;
- качественное техническое окружение.

Методология Crystal Clear уступает XP по производительности, зато максимально проста в использовании. Она требует минимальных усилий для внедрения, поскольку ориентирована на человеческие привычки. Считается, что эта методология описывает тот естественный порядок разработки ПО, который устанавливается в достаточно квалифицированных коллективах, если в них не занимаются целенаправленным внедрением другой методологии.

Основные характеристики Crystal Clear:

- итеративная инкрементная разработка;
- автоматическое регрессионное тестирование;
- пользователи привлекаются к активному участию в проекте;
- состав документации определяется участниками проекта;
- как правило, используются средства контроля версий кода.

В графическом виде практики Crystal Clear можно изобразить таким образом:

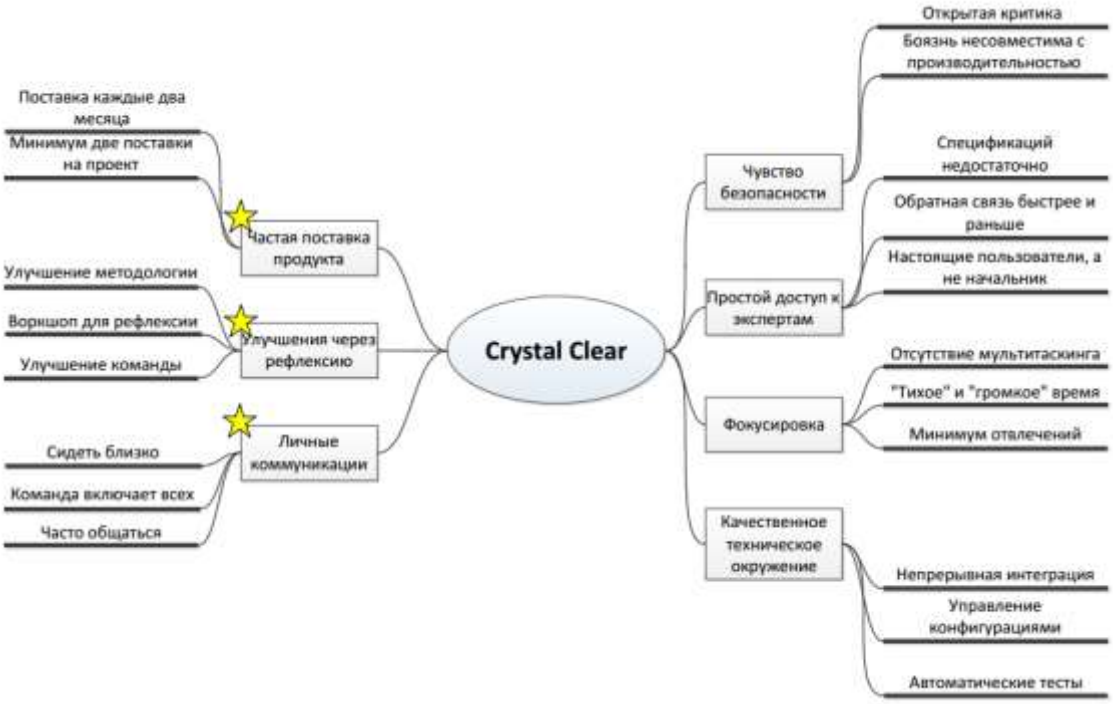

Рис. 3.7 Графическое представление Crystal Clear

Заключение

В данной главе были рассмотрены только несколько самых основных методологий, на самом деле их намного больше, например ICONIX, Канбан, Feature-driven development и другие. Каждая методология применяется в зависимости от типа программного продукта, а также определяет процесс проектирования ПО.

Вопросы и задания для самоконтроля

1. Что понимается под моделью ЖЦ ПО? Назовите существующие модели ЖЦ ПО.
2. Чем модель ЖЦ ПО отличается от методологии разработки ПО? Назовите существующие гибкие методологии разработки ПО.
3. Назовите основные особенности и стадии «Каскадной модели».
4. Назовите основные особенности и стадии «Эволюционной модели».
5. Методология Scrum. Что такое Спринт в рамках методологии Scrum? Какие группы ролей определены в данной методологии?

Тема 4
КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Процессы разработки, приобретения и внедрения сложных систем, к которым относятся в частности программные комплексы, должны находиться под жестким управленческим контролем. В настоящее время практически во всех организациях обеспечивается контроль важнейших характеристик, связанных с производством и использованием программных продуктов, таких как время, финансовые средства, ресурсы и т.п. Однако в большинстве случаев вне пределов сферы контроля оказывается наиболее важная характеристика программных продуктов, ради которой, собственно и осуществляются затраты времени, финансовых средств и ресурсов – это качество продукта, поскольку «невозможно контролировать то, что нельзя измерить» («You cannot control what you cannot measure»). Отсутствие возможности установки полного контроля вызывает рост количества необоснованных решений, увеличивает финансовые и проектные риски, связанные с разработкой и внедрением систем. Однако в настоящее время уже существуют организации, в которых накоплен достаточно большой опыт использования метрик в управлении качеством разрабатываемых и внедряемых программных продуктов. Использование апробированных подходов в управлении качеством разработки и внедрения крупных программных систем значительно повышает предсказуемость проектов, снижает финансовые и ресурсные издержки. Сейчас существует несколько определений качества, которые в целом совместимы друг с другом. К числу наиболее распространенных относятся [7]:

Определение 1 (ISO): *Качество – это полнота свойств и характеристик продукта, процесса или услуги, которые обеспечивают способность удовлетворять заявленным или подразумеваемым потребностям.*

Определение 2 (IEEE): *Качество программного обеспечения – это степень, в которой оно обладает требуемой комбинацией свойств.*

Анализ всех составляющих качества должен проводиться с учетом сфер ответственности заинтересованных сторон, как внутренних участников исполняемого процесса (in-process stakeholder), так и пользователей процесса (end-of-process stakeholders).

Очевидно, что управление качеством требует контроля всех измерений и оценок качества в жизненном цикле ПС (рисунок).

4.1. Измерение и оценка характеристик качества ПО

Программное обеспечение, в зависимости от особенностей разработки и применения, может представлять программу, программный комплекс, программное средство или программный продукт (изделие). Введем и будем в дальнейшем использовать следующие определения.

Качество программного обеспечения – это совокупность свойств, характеризующих способность программного обеспечения удовлетворять потребностям пользователя в соответствии с предназначением.

Рис. 4.1 Анализ Измерения и оценки качества в контуре управления качеством

Управление качеством – это система организационных, экономических, технологических и правовых мероприятий, осуществляемых для удовлетворения требований к качеству программного обеспечения в течение жизненного цикла.

Свойства программы – это особенности, объективно присущие программе, которые проявляются в ее жизненном цикле (разработке, применении, сопровождении).

Характеристика программы – это понятие, отражающее проявление отдельного измеримого фактора присущего программе свойства. Иначе говоря, характеристика – это проявляемый и измеримый атрибут свойства.

Измерение (оценка) одной или нескольких характеристик программы дает представление о том, насколько программе присуще то или иное свойство.

Каждому свойству соответствует одна или несколько характеристик программного обеспечения.

Для решения задачи количественной оценки характеристик программного обеспечения необходимо наличие системы измерений и

методов оценки [7, 18].

Система измерений характеристик программного обеспечения – это совокупность измеряемых характеристик, единиц измерения, измерительных шкал и связей, установленных между ними. Если между измеряемыми характеристиками установлены иерархические связи, систему измерений называют иерархической, в противном случае – одноранговой.

Измерительная шкала устанавливает границы (диапазон) и точность измерений характеристик свойств в установленных единицах.

Результаты измерений в избранной измерительной шкале позволяют обнаружить сходство и различие в свойствах программного обеспечения с целью последующей оценки и классификации. Применительно к ПО используют главным образом следующие известные виды измерительных шкал: *номинальные (категорийные), порядковые, интервальные.*

Номинальная (категорийная) шкала фиксирует наличие или отсутствие некоторой характеристики свойства без учета градаций и позволяет классифицировать программы по этому принципу.

Порядковая шкала фиксирует отношение порядка и позволяет ранжировать программы относительно некоторого опорного значения характеристик свойств.

Интервальная шкала фиксирует не только отношение порядка, но и величину, отличающую одно значение характеристики от другого (интервал между значениями).

Методы оценки характеристик программного обеспечения делят на следующие шесть групп: измерительные, регистрационные, органолептические, расчетные, экспертные, социологические, традиционные.

Измерительные методы основаны на получении информации о характеристиках программного обеспечения с использованием специальных инструментальных средств (технических или программных средств, обеспечивающих проведение измерений и их автоматизацию).

Регистрационные методы основаны на получении информации о характеристиках программного обеспечения во время испытаний или функционирования путем регистрации и подсчета определенных событий (например, моментов и количества ошибок, времени начала и окончания расчетов и т.д.), регистрируемых извне программы с помощью средств измерений общего назначения.

Органолептические методы основаны на получении информации о характеристиках программного обеспечения путем их восприятия органами чувств – в первую очередь зрения, слуха и осязания.

Расчетные методы основаны на получении информации о характеристиках программного обеспечения за счет использования теоретических или эмпирических зависимостей.

Экспертные методы используют опыт экспертов-специалистов, компетентных в оценке характеристик программного обеспечения.

Социологические методы используют обработку специальных анкет-опросников, содержащих качественные оценки характеристик программного обеспечения социальными группами, имеющими отношение к применению программного обеспечения.

Традиционные методы объединяют группу сформировавшихся и традиционно используемых в организациях, на предприятиях и иных учреждениях методов количественной оценки характеристик программного обеспечения.

Проявляемые свойства программного обеспечения условно можно разделить на две группы (рис.): функциональные (внешние), конструктивные (внутренние).

Для разработчиков и пользователей программы представляют интерес определенные функциональные и конструктивные свойства, (например, надежность, эффективность, модульность, структурность). Как правило, пользователя (заказчика) интересуют те функциональные свойства, которые характеризуют полезность программного обеспечения. Именно эти внешние свойства, отражающие точку зрения пользователя, обуславливают качество программного обеспечения, то есть являются его факторами (рис. 4.2). Заметим, что для разработчиков представляют интерес не только внешние, но и внутренние, или конструктивные свойства, от которых зависит выполнение требований к программному обеспечению и восприятие его пользователем.

Характеристики качества отражают свойства, определяющие качество программного обеспечения. В силу сложной природы количественной оценки характеристик качества программного обеспечения для их оценки используют иерархические системы измерений. Иерархию характеристик качества образуют факторы, критерии, метрики и оценочные элементы (рис.3). Факторы и критерии, составляющие два верхних уровня иерархии измерений, отражают функциональные характеристики программного обеспечения, а нижние (метрики и оценочные элементы) – конструктивные характеристики, от которых зависит качество программного обеспечения. Измеримость характеристик качества обеспечивается составом характеристик самого нижнего уровня – оценочных элементов.

Фактором качества будем называть свойство, в той или иной степени обуславливающее качество программного обеспечения. При

оценке качества учитывают несколько факторов. Для получения численной оценки фактора качества используют один или несколько критериев качества.

Критерий качества – это понятие, признак или численный показатель, характеризующий оцениваемый фактор качества. Критерий качества может быть представлен имеющим физический смысл вычислимым выражением, составленным из характеристик качества, значением которого является показатель качества. Для вычисления значения критерия используют одну или несколько метрик.

Метрика – мера количественной оценки качества ПО по заданному критерию, система или способ измерений качества программного обеспечения. Метрика содержит один или несколько оценочных элементов.

Оценочный элемент – измеримая характеристика программного обеспечения, имеющая численное значение в избранной измерительной шкале.

Показатель качества – численное значение критерия качества, определяющее степень, в которой программе присуще определенное критерием свойство. В соответствии с ГОСТ 15467-79 под показателем качества следует понимать количественную характеристику одного или нескольких свойств программной продукции, составляющих ее качество применительно к определенным условиям ее создания и эксплуатации.

Комплексный показатель качества – показатель качества, значение которого получают в результате композиции значений других, в том числе комплексных показателей. Таким образом, качество ПС многомерное понятие.

Базовое значение показателя качества – это реально достижимое значение показателя, отражающее современный уровень развития программного обеспечения.

Совокупность операций, включающих выбор номенклатуры (состава) показателей качества, определения значений этих показателей и сравнения их с базовыми значениями, называют **оценкой качества программного обеспечения.**

Процесс определения соответствия программного обеспечения действующему стандарту качества называют **сертификацией.** Процесс определения соответствия программного обеспечения предназначению называют **верификацией.** Процесс подтверждения функциональной пригодности программного обеспечения называют **аттестацией.**

Очевидно, что сертификация, верификация и аттестация программного обеспечения не исключают, а предполагают проведение количественных оценок характеристик программ.

Учитывая, что по определению программное обеспечение состоит не только из программ, но и документации к ним, одной из задач оценки качества ПО является измерение и оценка характеристик программных и эксплуатационных документов [7, 19].

4.2. Концепция и сущность управления качеством ПС

Основное содержание концепции управления качеством сводится к следующим положениям:

- требования к уровню качества по каждому фактору определяют базовым значением показателя качества;
- требуемый уровень качества обеспечивается процессом и в процессе производства;
- измерение, оценка и контроль уровня качества производится на всех стадиях жизненного цикла;
- управление качеством есть непрерывный, информационный и целенаправленный процесс воздействия на программы и документацию, а также на коллективы разработчиков ПС в целях обеспечения требуемого качества при изменяющихся внешних и внутренних условиях путем принятия управленческих решений.

Следуя этой концепции, можно констатировать, что сущность управления качеством в процессе разработки ПС состоит из трех основных видов деятельности:

- **Обеспечение качества.** Определение множества организационных процедур и стандартов в целях создания ПО высокого качества.
- **Планирование качества.** Выбор из этого множества соответствующего подмножества процедур и стандартов и адаптация их к данному проекту разработки ПО.
- **Контроль качества.** Определение и проведение мероприятий, гарантирующих выполнение нормативных процедур и стандартов качества всеми членами команды разработчиков ПО.

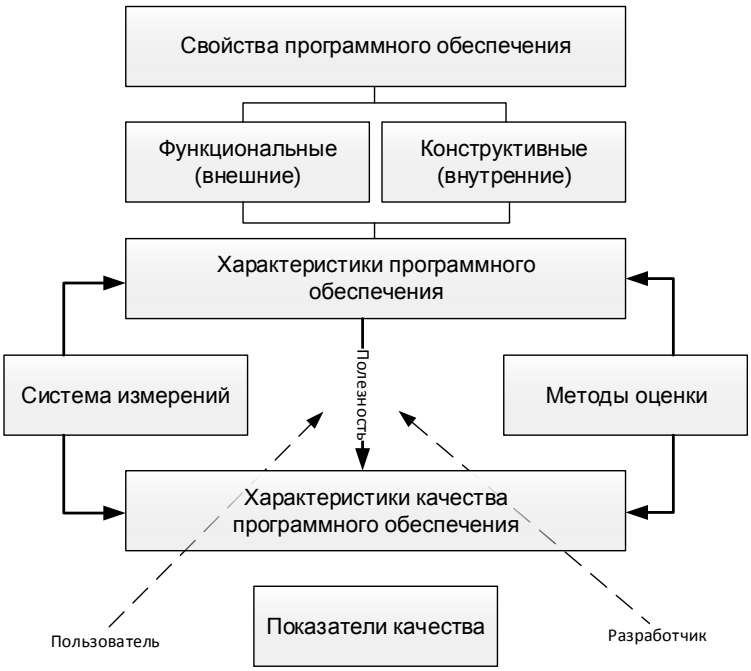

Рис. 4.2 Группы свойств и характеристики программного обеспечения

Управление качеством предполагает возможность независимого контроля за процессом разработки ПС. Контрольные проектные элементы, получаемые в процессе разработки ПС, являются основой контроля качества. Они тщательно проверяются на соответствие стандартам и целям проекта (рис. 4.3**Ошибка! Источник ссылки не найден.**). Так как работы, выполняемые по обеспечению и контролю качества, в определенной степени независимы, это предполагает возможность объективного взгляда на процесс разработки ПС, благодаря чему руководство компании может своевременно получить информацию о проблемах или трудностях, которые возникают в работе над проектом.

По мнению известного специалиста в области программной инженерии Иана Соммервилла, процесс управления качеством необходимо отделять от процесса управления проектом с тем, чтобы не ставить вопрос о компромиссе между качеством создаваемого ПО и бюджетом или графиком выполнения проекта. Над контролем качества должна работать независимая команда, которая отчитывается непосредственно руководству заказчика, минуя звено управляющего проектом (менеджера проекта). Вместе с тем признается факт, что команда контроля качества должна быть связана с группой (группой разработки) и несет ответственность за качество на уровне всей организации разработчика [7, 20].

Рис. 4.3. Факторы качества, отражающие полезность ПО

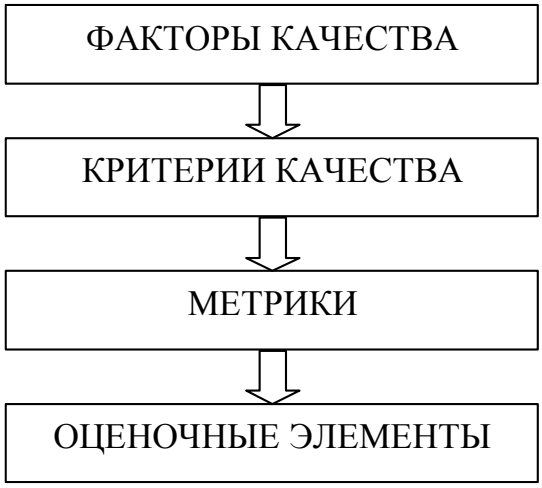

Рис. 4.4 Модель измерений характеристик качества (по ГОСТ 28195-89)

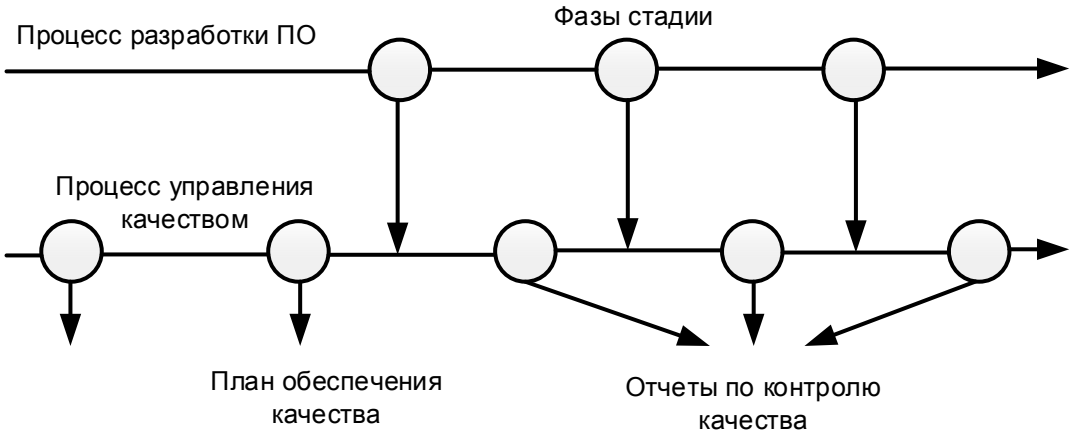

Рис. 4.5. Управление качеством и разработка ПС

4.3. Роль стандартизации и сертификации в управлении качеством ПС

Одним из краеугольных камней современного управления качеством является стандартизация. По определению Международной организации по стандартизации (ISO) стандартизация представляет собой «процесс установления и применения правил с целью упорядочения в данной области на пользу и при участии всех заинтересованных сторон, в частности, для достижения всеобщей максимальной экономии с соблюдением функциональных условий и требований безопасности». В толковом словаре по информатике В.И. Першикова и В.М. Савинкова понятие стандартизация определяется как принятие соглашения по спецификации, производству и использованию аппаратных и программных средств вычислительной техники; установление и применение стандартов, норм, правил и т.п.

Стандартизация выполняет следующие функции:

- упорядочивание объектов (продукции, работ, услуг, процессов), создаваемых людьми в разных странах;
- закрепление в нормативных документах оптимальных требований к упорядоченным объектам;
- установление правил применения этих нормативных документов.

На международном уровне стандартизация:

- обеспечивает взаимозаменяемость элементов сложной продукции;
- сближает уровень качества товаров, производимых в разных странах;

- содействует взаимообмену научно-технической информацией;
- содействует международной торговле.
- ускоряет научно-технический прогресс участников международных организаций.

Необходимость стандартизации разработки ПС на международном уровне, согласно стандарту ИСО/МЭК 12207 (введение), определена следующим образом: Программное обеспечение является неотъемлемой частью информационных технологий и традиционных систем, таких, как транспортные, военные, медицинские и финансовые. Имеется множество разнообразных стандартов, процедур, методов, инструментальных средств и типов операционной среды для разработки и управления программным обеспечением. Это разнообразие создает трудности при проектировании и управлении программным обеспечением, особенно при объединении программных продуктов и сервисных программ. Стратегия разработки программного обеспечения требует перехода от этого множества альтернатив к общему порядку, который позволит специалистам, практикующимся в программном обеспечении, «говорить на одном языке» при разработке и управлении программным обеспечением. Этот международный стандарт обеспечивает такой общий порядок".

Таким образом, в процессе стандартизации вырабатываются нормы, правила, требования, характеристики, касающиеся объекта стандартизации, которые оформляются в виде нормативного документа.

Поскольку сертификация устанавливает соответствие действующему стандарту, без наличия стандартов невозможна и сертификация [7].

Виды нормативных документов, рекомендуемые международными организациями по стандартизации (ИСО/МЭК), а также принятые в государственной системе стандартизации представлены на рисунке 4.6. (стандарты, технические условия, своды правил, регламенты, положения).

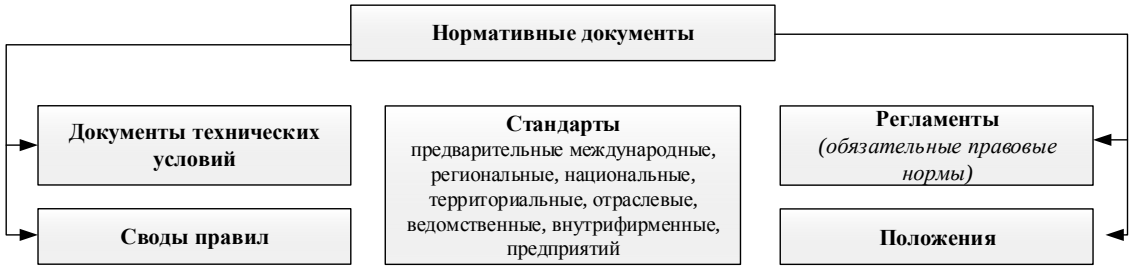

Рис. 4.6. Схема разновидностей нормативных документов

Стандарт – это нормативный документ, разработанный на основе консенсуса, утвержденный признанным органом, направленный на достижение оптимальной степени упорядочения в определенной области. В стандарте устанавливаются для всеобщего и многократного использования общие принципы, правила, характеристики рекомендательного характера, касающиеся различных видов деятельности или их результатов.

Таким образом, стандарты в разработке ПС важны по целому ряду причин. Основными из них являются:

1. Стандарты аккумулируют все лучшее из практической деятельности создания ПС и позволяют избежать повторения прошлых ошибок.

2. Стандарты предоставляют необходимую основу для процесса обеспечения качества: достаточно контролировать соблюдение стандартов.

3. Стандарты позволяют упорядочить процесс разработки, что делает разработку прозрачной и снижает затраты на обучение профессиональной деятельности при ротации кадров.

При современном уровне сложности программных систем и в условиях рыночной конкуренции представляется актуальным задача создания технологии коллективной разработки программных средств (ПС), которая будет отражать реалии процесса разработки и обеспечивать рост уровня производства при соответствующем качестве создаваемых программных изделий (ПИ). Из-за разнообразия типов ПС и способов их разработки технология должна обеспечить механизмы собственной адаптации и автоматизации. Актуальными являются и аспекты экономической эффективности и правовой защиты технологии при ее использовании для создания реальных процессов разработки ПС.

Многие современные исследователи в области процессов разработки программных средств рассматривают процедуру создания программ как сущность, включающую три тесно связанный компоненты: процесс (организацию разработки), коллектив разработчиков и программное изделие. Приводимые в статьях и литературе описания процедуры разработки ПИ подробно рассматривают ее организацию (процесс), игнорируя при этом детальное описание других компонент. Те модели компонент технологии разработки ПС, которые можно встретить в современной литературе, носят в основном описательный, а не формальный характер.

Потребность в создании сложных программных систем приводит к необходимости регламентации творческого процесса. И каждая методология программирования пытается построить процесс разработки

таким образом, чтобы минимизировать творческий элемент в случаях рутинной работы. Иными словами, методологии стремятся сделать так, чтобы сокращалось число ошибок, чтобы как можно раньше переходить если не к производству, то хотя бы к тому, что является аналогом производства при разработке программ. Отсюда попытки разграничить план и конструкцию программы, спецификации пользовательской потребности и план, выбор инструментов для работы программиста и саму работу. Это же приводит к появлению регламентов и предписаний, следование которым уменьшает вероятность ошибочных решений.

По существу, любая методология представляет собой набор регламентов и предписаний. В частности, любая методология выстраивает свою модель жизненного цикла как основу для этих соглашений.

Понятие жизненного цикла само по себе от методологий не зависит. И в «хаотическом» конструировании ранних программных продуктов, и в современных «жестких» методологиях, и в так называемых «облегченных» (lightweight) методологиях можно указать на жизненный цикл. И хотя форма представления жизненных циклов в разных случаях различна до неузнаваемости, мы настаиваем на том, что в основе любых представлений разработки и сопровождения программных изделий лежат общие процессы, которые в конечном итоге ведут проекты от их замыслов к удовлетворению потребностей пользователя. Любая методология предписывает организацию этих общих процессов. Рассмотрим, как проходит предварительная оценка сложности разработки программных систем на основе статистических методов в зависимости от этапов разработки программы.

При использовании интегрированных инструментальных средств у компаний, разрабатывающих типовые решения (под эту категорию попадают так называемые «инхаузеры» – компании, занимающиеся обслуживанием основного бизнеса) появляется возможность строить прогнозы сложности программ, основываясь на собранной статистике. Статистический метод хорошо подходит для решения подобных типовых задач и практически не подходит для прогноза уникальных проектов. В случае уникальных проектов применяются иные подходы, обсуждение которых находится за рамками данного материала.

Типовые задачи падают на отделы разработки из бизнеса, потому предварительная оценка сложности могла бы сильно упростить задачи планирования и управления, тем более что есть накопленная база по проектам, в которой сохранены не только окончательные результаты, но и все начальные и промежуточные [7, 18, 20].

Вопросы и задания для самоконтроля

1. Перечислите основные стандарты, описывающие оценку качества программного обеспечения.
2. Назовите факторы качества программного обеспечения.
3. Что такое модель зрелости? Особенности.
4. Какие модели качества процессов разработки вы знаете?

Тема 5
МЕТОДЫ ВЫЯВЛЕНИЯ ТРЕБОВАНИЙ К ПО. УРОВНИ ТРЕБОВАНИЙ. АНАЛИЗ ТРЕБОВАНИЙ К ПО

Цель анализа требований в проектах – получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски, а также сформировать проектную группу, на которую будет возложена значительная часть предстоящих работ.

На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Анализ требований выполняется на основе совещаний и собеседований с руководителями и специалистами заказчика, а продолжительность этого этапа, в зависимости от сложности задач и масштаба внедрения, может составлять от нескольких дней до нескольких недель.

Определение и описание требований (методологических и технических) – шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы. Практика показывает, что недостаточная проработка требований зачастую проявляется лишь тогда, когда проект почти завершен, а значительная часть ресурсов, выделенных на его реализацию, уже затрачена. К сожалению, устранение проблем на этапе разработки обходится гораздо дороже, чем тщательная проработка на стадии анализа [12, 21].

5.1. Особенности интерпретации требований

IEEE Standard Glossary of Software Engineering Terminology (1990) определяет требования как:

1. Условия или возможности, необходимые пользователю для решения проблем или достижения целей;

2. Условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;

3. Документированное представление условий или возможностей для пунктов 1 и 2.

Это определение охватывает требования как пользователей *(внешнее поведение системы)*, так и разработчиков *(некоторые скрытые*

параметры). Термин пользователи следует распространить на всех заинтересованных лиц, так как не все, кто заинтересован в проекте – пользователи.

Требования – это спецификация того, что должно быть реализовано. В них описано поведение системы, свойства системы или ее атрибуты. Они могут быть ограничены процессом разработки системы.

Уровни требований.

Три уровня требований к ПО:

- бизнес-требования;
- требования пользователей;
- функциональные требования.

5.2. Типы требований.

Каждая система имеет свои **функциональные и нефункциональные требования.**

Бизнес-требования (business requirements) содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Бизнес-требования формулируют в документе об образе и границах проекта – уставе проекта (project charter), или документом рыночных требований (market requirements document). Определение границ проекта представляет собой первый этап управление общими проблемами расползания границ.

Требования пользователей (user requirements) описывают цели и задачи, которые пользователям позволит решить система. К отличным способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие – отклик». Таким образом, в этом документе указано, что клиенты смогут делать с помощью системы.

Функциональные требования (functional requirements) определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Иногда именуемые требованиями поведения (behavioral requirements), они содержат положения с традиционным «должен» или «должна»: «Система должна по электронной почте отправлять пользователю подтверждение о заказе».

Термином системные требования (system requirements) обозначают высокоуровневые требования к продукту, которые содержат

многие подсистемы, то есть система (IEEE, 1998c). Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди – часть системы, поэтому определенные функции системы могут распространяться и на людей.

Бизнес-правила (business rules) включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные варианты использования, или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности. Следовательно, вы можете отследить происхождение конкретных функциональных требований вплоть до соответствующих им бизнес-правил.

Нефункциональные требования содержат цели и атрибуты качества. **Атрибуты качества (quality attributes)** представляют собой дополнительное описание функций продукта, выраженное 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 [21, 25].

Спецификация требований не содержит деталей дизайна или реализации (кроме известных ограничений), данных о планировании проекта или сведений о тестировании.

5.3. Приемы формулирования требований

Обучение аналитиков требований. Всем членам команды, которые будут исполнять функции аналитиков, необходимо научиться приемам формулирования требований – это может занять несколько дней. Квалифицированный аналитик требований терпелив и методичен, обладает навыками межличностного общения и коммуникативными навыками, сведущ в предметной области и знает множество способов формулирования требований к ПО.

Ознакомление пользователей и менеджеров с требованиями. Пользователи, которые будут принимать участие в разработке ПО, должны пройти непродолжительный тренинг (один-два дня), чтоб научиться формулировать требования. Он полезен и для менеджеров по разработке и по работе с клиентами. Обучение поможет понять особое значение выделения требований, суть процесса их разработки, а также опасность пренебрежения ими. Посетив мои семинары по требованиям, некоторые пользователи замечали, что стали теплее относиться к разработчикам ПО.

Ознакомление разработчиков с концепциями предметной области. Чтобы помочь разработчикам в общих чертах понять предметную область, проведите семинар, на котором познакомьте их с бизнесом клиента, терминологией и назначением создаваемого продукта. Это уменьшит вероятность путаницы, непонимания и доработок. Можно также на время проекта назначить каждому разработчику «личного пользователя», который будет разъяснять профессиональные термины и бизнес-концепции. Лучше, если это будет настоящий фанат продукта.

Создание бизнес-словаря. Словарь со специализированными терминами из предметной области снизит вероятность непонимания, включите в него синонимы, термины, имеющие несколько значений, и термины, имеющие в предметной области и повседневной жизни разные значения [12].

5.4. Выявление требований

Определение процесса формулирования требований. Документация этапов выявления, анализа, определения и проверки требований. Наличие инструкций по выполнению ключевых операций поможет аналитикам качественно и согласованно выполнить их работу. Кроме того, будет проще поставить задачи по созданию требований и графики, а также продумать необходимые ресурсы.

Определение образа и границы проекта. Документ об образе и границах проекта содержит бизнес-требования к продукту. Описание образа проекта позволит всем заинтересованным лицам в общих чертах понять назначение продукта. Границы проекта определяют, что следует реализовать в этой версии, а что – в следующих. Образ и границы проекта – хорошая база для оценки предлагаемых требований, Образ продукта должен оставаться от версии к версии относительно стабильным, но для каждого выпуска необходимо составлять отдельный документ о границах.

Определение классов пользователей и их характеристик. Чтобы не упустить из виду потребности отдельных пользователей, необходимо их объединить в группы. Например, по частоте работе с ПО, используемым функциям, уровню привилегий и навыкам работы. Производится описание их обязанности, местоположение и личные характеристики, способные повлиять на архитектуру продукта.

Выбор сторонника продукта (product champion) в каждом классе пользователей. Это человек, который сможет точно передавать настроения и нужды клиентов. Он представляет потребности определенного класса пользователей и принимает решения от их лица. В случае разработки внутрикорпоративных информационных систем, когда все пользователи – ваши коллеги, такого человека выбрать проще. При коммерческой разработке расспросите клиентов или используйте площадки бета-тестирования. Выбранные вами люди должны принимать постоянное участие в проекте и обладать полномочиями для принятия решений, касающихся пользовательских требований.

Создание фокус-групп типичных пользователей. Определите группы типичных пользователей предыдущих версий вашего продукта или похожих. Выясните у них подробности о функциональности и качественных характеристиках разрабатываемого продукта. Фокус-группы особенно значимы при разработке коммерческих продуктов, когда приходится иметь дело с большой и разнородной клиентской базой. В отличие от сторонников продукта, у фокус-групп обычно нет полномочий на принятие решений.

Работа с пользователями для выяснения назначения продукта. Выясните у пользователей, какие задачи им требуется выполнять средствами ПО. Обсудите, как должен клиент взаимодействовать с системой для выполнения каждой такой задачи. Воспользуйтесь стандартным шаблоном для документирования всех задач и для каждой сформулируйте функциональные требования. Похожий способ, часто применяемый в правительственных проектах, — создать документ с концепциями операций (ConOps), где указаны характеристики новой системы сточки зрения пользователя (IEEE, 1998a).

Определение системных событий и реакции на них. Определите возможные внешние события и ожидаемую реакцию системы на них. Это могут быть сигналы и данные, получаемые от внешнего оборудования, а также временные события, вызывающие ответную реакцию, например ежевечерняя передача данных, генерируемых системой, внешнему объекту. В бизнес-приложениях бизнес-события напрямую связаны с задачами.

Проведение совместных семинаров. Совместные семинары по выявлению требований, где тесно сотрудничают аналитики и клиенты – отличный способ выявить нужды пользователей и составить наброски документов с требованиями (Gottesdiener, 2002). Конкретные примеры таких семинаров – Joint Requirements Planning (JRP – совместное планирование требований) (Martin, 1991) и Joint Application Development (JAD – совместная разработка приложений) (Wood и Silver, 1995).

Наблюдение за пользователями на рабочих местах. Наблюдая за работой пользователей, выявляют контекст потенциального применения нового продукта (Beyer и Holtzblatt, 1998). Простые диаграммы рабочих потоков, а также диаграммы потоков данных позволяют выяснить, где, как и какие данные задействовал пользователь. Документируя ход бизнес-процесса, удается определить требования к системе предназначенной для поддержки этого процесса. Иногда даже выясняется, что для выполнения деловых задач клиентам вовсе и не требуется новое ПО (McGraw и Harbison, 1997).

Изучение отчетов о проблемах работающих систем с целью по иска новых идей. Поступающие от клиентов отчеты о проблемах и предложения о расширении функциональности — отличный источник: идей о возможностях, которые можно реализовать в следующей версии или новом продукте. За подобной информацией стоит обратиться и к персоналу службы поддержки.

Повторное использование требований в разных проектах. Если необходимая клиенту функциональность аналогична уже реализованной в другом продукте, подумайте, готовы ли клиенты гибко пересмотреть свои требования для использования существующих компонентов Требования, соответствующие бизнес-правилам компании, можно применить в нескольких проектах. Это требования к безопасности, определяющие порядок доступа к приложениям, и требования, соответствующие постановлениям правительства, например Закон о гражданах США с ограниченными возможностями (Americans with Disabililies Act) [12, 21].

5.5. Анализ требований

Создание контекстной диаграммы. Контекстная диаграмма – простая модель анализа, отображающая место новой системы в соответствующей среде. Она определяет границы и интерфейсы между разрабатываемой системой и сущностями, внешними для этой системы, например пользователями, устройствами и прочими информационными системами.

Создание пользовательского интерфейса и технических прототипов. Если разработчики или пользователи не совсем уверены насчет требований, создайте прототип – частичную, возможную или предварительную версию продукта, которая сделает концепции и возможности более осязаемыми. Оценка прототипа поможет всем заинтересованным лицам достичь взаимопонимания по решаемой проблеме.

Анализ осуществимости требований. Проанализируйте, насколько реально реализовать каждое требование при разумных затратах и с приемлемой производительностью в предполагаемой среде. Рассмотрите риски, связанные с реализацией каждого требования, включая конфликты с другими требованиями, зависимость от внешних факторов и препятствия технического характера.

Определение приоритетов требований. Воспользуйтесь аналитическим подходом и определите относительные приоритеты реализации функций продукта, решаемых задач или отдельных требований. На основании приоритетов установите, в какой версии будет реализована та или иная функция или набор требований. Подтверждая изменения, распределите все их по конкретным версиям и включите в план выпуска этих версий затраты, необходимые на внесение изменений. В ходе работы над проектом периодически корректируйте приоритеты в соответствии с потребностями клиента, условиями рынка и бизнес-целями.

Моделирование требований. В отличие от подробной информации, представленной в спецификации требований к ПО или пользовательского интерфейса прототипа, графическая модель анализа отображает требования на высоком уровне абстракции. Модели позволяют выявить некорректные, несогласованные, отсутствующие и избыточные требования. К таким моделям относятся диаграммы потоков данных, диаграммы «сущность – связь», диаграммы перехода состояний, называемые также автоматами (statecharts), карты диалогов, диаграммы классов, диаграммы последовательностей, диаграммы взаимодействий, таблицы решений и деревья решений.

Создание словаря терминов. В нем соберите определения всех элементов и структур данных, связанных с системой, что позволяет всем участникам проекта использовать согласованные определения данных. На стадии работы над требованиями словарь должен содержать определения элементов данных, относящихся к предметной области, чтобы клиентам и разработчикам было проще общаться.

Распределение требований по подсистемам. Требования к сложному продукту, включающему несколько подсистем, следует

соразмерно распределять между программными, аппаратными и операторскими подсистемами и компонентами (Nelsen, 1990). Как правило, это осуществляет системный инженер или разработчик. **Применение технологий развертывания функций качества.** Технология развертывания функций качества (Quality Function Deployment, QFD) – точная методика, соотносящая возможности и атрибуты продукта с их значимостью для клиента (Zultner, 1993; Pardee, 1996). Она позволяет аналитически выявить функции, которые максимально удовлетворят потребности клиента. Технология развертывания функций качества рассчитана на три класса требований: ожидаемые, о которых клиент может не упомянуть, но будет расстроен, если их не окажется в продукте, обычные требования и отдельные, специальные требования, которые обеспечивают удобство работы клиентам, но отсутствие которых не влечет санкций со стороны клиента [12].

5.6. Спецификации требований

Использование шаблона спецификации требований к ПО. Создайте стандартный шаблон для документирования требований к ПО в вашей организации. Шаблон предоставляет согласованную структуру, позволяющую фиксировать описания нужной функциональности, а также прочую информацию, касающуюся требований. Вместо того чтобы изобретать новый шаблон, модифицируйте один из существующих в соответствии со спецификой проекта. Многие компании начинают с использования шаблона спецификации требований к ПО, описанного в стандарте IEEE 830-1998 (IEEE, 1998b). Если ваша компания занимается разными проектами, например, проектирует новое крупное приложение и параллельно дорабатывает версии старых программ, создайте соответствующие шаблоны для всех типов проектов. Шаблоны и процессы должны быть масштабируемыми.

Определение источников требований. Чтобы гарантировать, что все заинтересованные лица понимают, почему-то или иное требование зафиксировано в спецификации требований к ПО, и упростить последующее прояснение требований, выявите источники всех требований. Это может быть вариант использования или другая информация от пользователей, системное требование высокого уровня, бизнес-правило или иной внешний фактор. Указав всех лиц, заинтересованных в каждом требовании, вы будете знать, к кому обратиться при поступлении запроса на изменение. Источники

требований устанавливают на основе связей или определяют для этой цели атрибут требования.

Присвоение уникальных идентификаторов всем требованиям. Выработайте соглашение о присвоении уникальных идентификаторов требованиям, зафиксированным в спецификации требований к ПО. Соглашение должно быть устойчивым к дополнению, удалению элементов и изменениям, вносимым в требования. Присвоение идентификаторов позволяет отслеживать требования и фиксировать вносимые изменения.

Указание атрибутов качества. Выявляя качественные характеристики, удовлетворяющие потребности клиента, не ограничивайтесь только обсуждением функциональности. Выясните ожидаемые производительность, эффективность, надежность, удобство в использовании др. Информация от клиентов об относительной важности тех или иных качественных характеристиках позволит разработчику принять правильные решения, касающиеся архитектуры приложения.

Документирование бизнес-правил. К бизнес-правилам относятся корпоративные политики, правительственные распоряжения и алгоритмы вычислений. Ведите список бизнес-правил отдельно от спецификации требований к ПО, поскольку правила обычно существуют вне рамок конкретного проекта. Для выполнения некоторых приходится создавать реализующие их функциональные требования, и поэтом необходимо определить связь между этими требованиями и соответствующими правилами [12, 25].

5.7. Проверка требований

Изучение документов с требованиями. Официальная проверка документирования требований – один из наиболее ценных способов проверки качества ПО. Соберите небольшую команду, члены которой представляют различные направления (например, аналитик, клиент, разработчик и специалист по тестированию}, и тщательно изучите спецификацию требований к ПО, модель анализа и соответствующую информацию на предмет недостатков. Также полезно провести в ходе формулирования требований их неофициальный предварительный просмотр. И хотя реализовать это на практике непросто, данный прием – один из самых ценных, так что начинайте внедрять проверку требований в вашей организации прямо сейчас.

Тестирование требований. На основе пользовательских требований создайте сценарии функционального тестирования и задокументируйте ожидаемое поведение продукта в конкретных

условиях. Совместно с клиентами изучите сценарии тестирования и убедитесь, что они отражают нужное поведение системы. Проследите связь сценариев тестирования с функциональными требованиями и удостоверьтесь, что ни одно требование не пропущено и что для всех требований есть соответствующие сценарии тестирования. Запустите сценарии, чтобы удостовериться в правильности моделей анализа и прототипов.

Определение критериев приемлемости. Предложите пользователям описать, как они собираются определять соответствие продукта их потребностям и его пригодность к работе. Тесты на приемлемость следует основывать на сценариях использования (Hsia, Kung и Sell, 1997).

5.8. Управление требованиями

Определение процесса управления изменениями. Определите процесс представления, анализа и утверждения или отклонения изменений. Применяйте его для управления всеми предлагаемыми изменениями. В контексте процесса управления изменениями полезно использовать коммерческие средства отслеживания недостатков.

Создание совета по управлению изменениями. Из представителей заинтересованных в проекте лиц организуйте совет по управлению изменениями, который будет получать информацию о предполагаемых изменениях требований, оценивать ее, решать, какие изменения принять, а какие отклонить, и определять, в какой версии продукта будет внедрена та или иная функция.

Анализ влияния изменений требований. Анализ влияния изменений помогает совету по управлению изменениями принимать обоснованные решения. Оцените, как каждое предлагаемое изменение требований повлияет на проект. На основе матрицы связей выявите другие требования, элементы архитектуры, исходный код и сценарии тестирования, которые, возможно, придется изменить. Определите, что необходимо для реализации изменений, и оцените затраты на реализацию.

Создание базовой версии и управление версиями требований. Базовая версия содержит требования, утвержденные для реализации в конкретной версии продукта. После определения базовых требований изменения можно вносить только в соответствии с процессом управления изменениями. Присвойте всем версиям спецификации требований уникальные идентификаторы, чтобы избежать путаницы между черновыми вариантами и базовыми версиями, а также между

предыдущей и текущей версиями требований. Более надежное решение – управлять версиями документов с требованиями при помощи соответствующих средств управления конфигурацией.

Ведение журнала изменений требований. Фиксируйте даты изменения спецификаций требований, сами коррективы, их причины, а также лиц, вносивших изменения. Автоматизировать эти задачи позволяет утилита управления версиями или коммерческая утилита управления требованиями.

Контроль за состоянием всех требований. Создайте БД, включающую по одной записи для каждого дискретного функционального требования. Занесите в БД ключевые атрибуты каждого требования, включая его состояние (например, «предложено», «одобрено», «реализовано» или «проверено»), чтобы в любой момент вы могли узнать количество требований в каждом состоянии.

Оценка изменяемости требований. Еженедельно фиксируйте количество требований, внесенных в базовую версию, а также число предложенных и одобренных изменений (добавлений, модификаций и удалений). Если требования формируются не самим клиентом, а от его лица, может оказаться, что проблема понята плохо, границы проекта определены нечетко, бизнес стремительно меняется, при сборе информации многие требования были упущены или внутрикорпоративные политики меняются в худшую сторону.

Использование средств управления требованиями. Коммерческие утилиты управления требованиями позволяют хранить различные типы требований в БД. Для каждого требования можно определить атрибуты, отслеживать его состояние, а также выявить связи между требованиями и другими рабочими продуктами. Данный прием поможет вам автоматизировать прочие задачи по управлению требованиями, описанные ниже.

Создание матрицы связей требований. Создайте таблицу, сопоставляющую все функциональные требования с элементами архитектуры и кода, которые реализуют данное требование, и с тестами, проверяющими его. Матрица связей требований позволяет также сопоставить функциональные требования с требованиями более высоких уровней, на основе которых они созданы, и с другими родственными требованиями. Заполняйте эту таблицу входе, а не в конце работы над проектом [12].

5.9. Управление проектом

Выбор цикла разработки ПО. Вашей компании следует определить несколько жизненных циклов разработки для проектов различного типа и различных степеней неопределенности требований (McCormel, 1996). Каждый менеджер проекта должен выбрать и использовать цикл, оптимальным образом подходящий для его проекта. Включите цикл операции по созданию требований. Если на ранних этапах работы над проектом требования или границы проекта определены нечетко, разрабатывайте продукт постепенно (небольшими этапами), начиная с наиболее понятных требований и устойчивых элементов архитектуры. По возможности реализуйте наборы функций, чтобы периодически выпускать промежуточные версии продукта и как можно раньше предоставлять клиенту работоспособные образцы приложение (Gilb, 1988; Cockburn, 2002).

Планы реализации проекта должны быть основаны на требованиях. Разрабатывайте планы и графики работы над проектом постепенно, по мере прояснения границ и подробных требований. Начните с оценки затрат, необходимых на реализацию функциональных требований, определенных на основе первоначальных образа и границ продукт. Графики и оценка затрат, построенные на основе нечетких требований, окажутся крайне неточными, однако по мере детализации требований их следует уточнить.

Пересмотр обязательств по проекту при изменении требований. Добавляя в проект новые требования, оцените, удается ли соблюдать обязательства, касающиеся графика и требований к качеству, при доступном объеме ресурсов. Если нет, обсудите реалии проекта с менеджерами и согласуйте новые, достижимые обязательства (Humphrey, 1997; Fisher, Ury, и Patton, 1991; Wiegers, 2002). Если переговоры не увенчаются успехом, сообщите менеджерам и клиентам о их результатах, чтобы нарушение планов в реализации проекта не стало для них неожиданностью.

Документирование и управление рисками, связанными с требованиями. Одна из составляющих управления рисками проекта – выявление и документирование рисков, связанных с требованиями. Уменьшайте или предотвращайте их посредством мозговых штурмов, реализуйте корректирующие действия и отслеживайте их эффективность.

Контроль объема работ по созданию требований. Фиксируйте усилия, прилагаемые вашей командой на разработку требований и управление проектом. Эти данные позволят оценить соответствие планам и эффективнее спланировать необходимые ресурсы для будущих проектов. Также отслеживайте, как ваши действия по регламентации

требований влияют на проект в целом. Это позволит оценить отдачу от этой работы.

Извлечение уроков из полученного опыта. Для этого в организации следует провести ретроспективу проектов, называемую также изучением законченных проектов (Robertson и Robertson, 1999; Kerth, 2001; Wiegers и Rothman, 2001). Ознакомление с опытом в области проблем и способов создания требований, накопленным в ходе работы над предыдущими проектами, помогает менеджерам и аналитикам требований более эффективно работать в будущем [12].

Вопросы и задания для самоконтроля

1. Назовите основные цели, преследуемые при анализе требований в проектах.
2. Перечислите типы требований.
3. Назовите методы выявления требований.
4. Перечислите задачи, которые решаются на стадии анализа требований.
5. Аналитик требований. Перечислите основные задачи аналитика требований.