МОДУЛЬ 1

РАЗДЕЛ 1. ВВЕДЕНИЕ В ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНЫХ СРЕДСТВ

1.1. Основные понятия и определения

Существуют различные определения технологии разработки ПО. Ниже приведены наиболее распространенные из них.

Технология разработки программного обеспечения – 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 [23]. Данное определение имеет частный характер, поскольку учитывает только две из существующих характеристик качества ПО – надежность и эффективность [12]. С учетом этого можно сформулировать более общее и точное определение.

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

Одно из определений современных технологий разработки ПО приведено в подразд. 5.1.

Близким по смыслу к технологии разработки ПО является широко используемый в настоящее время термин ***программная инженерия*** (software engineering).

Любая технология разработки ПО базируется на некоторой методологии или совокупности методологий.

Под ***методологией*** понимается система принципов и способов организации процесса разработки программных средств. *Цель* методологии разработки ПО –

внедрение методов разработки ПС, обеспечивающих достижение соответствующих характеристик качества.

В настоящее время широкую известность приобрели два базовых принципа разработки ПС: *модульный* и *объектно-ориентированный*.

Разработка модульных ПС основывается на использовании *структурных методов проектирования*, целью которых является разбиение по некоторым правилам проектируемого программного средства на структурные компоненты. К структурным методам проектирования относятся такие классические методы, как структурное программирование, нисходящее проектирование, расширение ядра, восходящее проектирование и их комбинации, а также ряд современных методов и методологий разработки ПО.

Объектно-ориентированная разработка базируется на применении объектных методов, к которым относятся методологии объектно-ориентированного анализа, проектирования и программирования.

В учебном пособии используются следующие термины [7, 9].

Программные средства, программное обеспечение (software) – полный набор (программное обеспечение) или часть (программные средства) программ, процедур, правил и связанной с ними документации системы обработки информации. *Программное средство* – ограниченная часть программного обеспечения системы обработки информации, имеющая определенное функциональное назначение.

Программный модуль (software unit) – отдельно компилируемая часть программного кода (программы).

Программный продукт (software product) – набор компьютерных программ, процедур, а также связанных с ними документации и данных. Продукты включают промежуточные продукты и продукты, предназначенные для пользователей типа разработчиков и персонала сопровождения.

Система (system) – комплекс, состоящий из процессов, технических и программных средств, устройств и персонала, обладающий возможностью удовлетворять установленным потребностям или целям.

Нотация (notation) – система графических обозначений для записи промежуточных и конечного результатов разработки ПС (в том числе предметной области, требований, результатов проектирования и т.п.).

Базовыми понятиями технологии разработки ПС являются понятия жизненного цикла, стратегии разработки и модели жизненного цикла, реализующей данную стратегию. Эти понятия рассматриваются в подразд. 1.2 и разд. 2 ЭРУД.

Резюме

Технология разработки программного обеспечения – это система инженерных принципов для создания экономичного ПО с заданными характеристиками качества. Методология разработки программного обеспечения – это система принципов и способов организации процесса разработки программных средств. Целью методологии является внедрение методов разработки ПО, обеспечивающих достижение соответствующих характеристик качества.

1.2. Жизненный цикл программных средств

В настоящее время базовым стандартом в области ЖЦ ПС и систем является международный стандарт ***ISO/IEC 12207:2008 – Системная и программная инженерия – Процессы жизненного цикла программных средств***. В Республике Беларусь с 2004 г. действует национальный стандарт ***СТБ ИСО/МЭК 12207–2003 – Информационная технология – Процессы жизненного цикла программных средств***, являющийся аутентичным аналогом предыдущей редакции международного стандарта ***ISO/IEC 12207:1995***.

В соответствии со стандартом *СТБ ИСО/МЭК 12207–2003* под ***жизненным циклом*** программного средства или системы подразумевается совокупность процессов, работ и задач, включающая в себя разработку, эксплуатацию и сопровождение программного средства или системы и охватывающая их жизнь от формулирования концепции до прекращения использования. ЖЦ ПС состоит из *процессов*. Каждый процесс ЖЦ разделен на набор *работ*. Каждая работа разделена на набор *задач*.

Процессы ЖЦ ПС делятся на три *группы*:

- основные;
- вспомогательные;
- организационные.

Основные процессы жизненного цикла – это процессы, которые реализуются под управлением основных сторон, участвующих в жизненном цикле программных

средств. Основными сторонами являются заказчик, поставщик, разработчик, оператор и персонал сопровождения программных продуктов. К *основным процессам* относится пять процессов:

- заказ;
- поставка;
- разработка;
- эксплуатация;
- сопровождение.

Процесс заказа определяет работы и задачи заказчика и состоит из определения потребностей заказчика в системе или программном продукте, подготовки и выпуска заявки на подряд, выбора поставщика и управления процессом заказа до завершения приемки системы или программного продукта.

Процесс поставки определяет работы и задачи поставщика. Данный процесс начинается с решения о подготовке предложения в ответ на заявку на подряд, присланную заказчиком, или с подписания договора с заказчиком на поставку системы или программного продукта. Затем определяются процедуры и ресурсы, необходимые для управления и обеспечения проекта, включая разработку проектных планов и их выполнение.

Процесс разработки состоит из работ и задач, выполняемых разработчиком. Данный процесс содержит *тринадцать работ*:

1) подготовка процесса разработки;
2) анализ требований к системе;
3) проектирование системной архитектуры;
4) анализ требований к программным средствам;
5) проектирование программной архитектуры;
6) техническое проектирование программных средств;
7) программирование и тестирование программных средств;
8) сборка программных средств;
9) квалификационные испытания программных средств;
10) сборка системы;
11) квалификационные испытания системы;
12) ввод в действие программных средств;

13) обеспечение приемки программных средств.

При выполнении работы 1 «Подготовка процесса разработки» выбирается модель ЖЦ ПС и систем. В данную модель структурируются процессы, работы и задачи стандарта *СТБ ИСО/МЭК 12207–2003*. Выбираются и адаптируются стандарты, методы, инструментальные средства разработки, языки программирования. Формируется план проведения работ процесса разработки.

При выполнении работы 2 «Анализ требований к системе» анализируется область применения системы. На основании результатов анализа предметной области определяются требования к ней. Выполняется оценка разработанных требований.

При выполнении работы 3 «Проектирование системной архитектуры» определяется общая архитектура системы. Осуществляется распределение требований к системе между объектами технических и программных средств архитектуры и ручными операциями. Производится дальнейшее уточнение требований. Осуществляется оценка архитектуры системы и требований к объектам архитектуры.

При выполнении работы 4 «Анализ требований к программным средствам» анализируется назначение программного средства. Исходя из результатов анализа определяются и уточняются требования к программному средству. Выполняется оценка разработанных требований.

При выполнении работы 5 «Проектирование программной архитектуры» разрабатывается эскизный проект программного средства. Требования к программному средству преобразуются в его архитектуру, распределяются между его компонентами и уточняются далее. Производится оценка результатов эскизного проектирования.

При выполнении работы 6 «Техническое проектирование программных средств» осуществляется детальное проектирование программного средства (разрабатывается технический проект для компонентов программного объекта). Компоненты проектируются до уровня их представления в виде набора программных модулей. Сложность модулей должна обеспечить возможность их непосредственного кодирования в следующей работе (работе 7). Производится распределение технических требований к компонентам между программными модулями и дальнейшее уточнение требований. Выполняется оценка технического проекта.

При выполнении работы 7 «Программирование и тестирование программных средств» производится кодирование и тестирование программных модулей.

Осуществляется оценка полученных результатов программирования и тестирования.

При выполнении работы 8 «Сборка программных средств» производится сборка программных модулей и компонентов в программное средство и тестирование промежуточных и конечных результатов сборки. Выполняется оценка результатов сборки и тестирования.

При выполнении работы 9 «Квалификационные испытания программных средств» проводятся квалификационные испытания программного средства в моделируемой среде с моделируемыми исходными данными. Оцениваются результаты испытаний. При необходимости производится доработка программного продукта.

При выполнении работы 10 «Сборка системы» осуществляется сборка объектов программной и технической конфигурации, ручных операций, других подсистем в единую систему. Проводятся испытания собранной системы. Выполняется оценка собранной системы.

При выполнении работы 11 «Квалификационные испытания системы» проводятся квалификационные испытания собранной системы в моделируемой среде с моделируемыми исходными данными. По результатам испытаний выполняется оценка системы и при необходимости доработка программного продукта.

При выполнении работы 12 «Ввод в действие программных средств» программный продукт вводится в действие в среде эксплуатации.

При выполнении работы 13 «Обеспечение приемки программных средств» обеспечивается проведение заказчиком приемочных испытаний. Программный продукт поставляется заказчику.

В приведенной последовательности различают ***два вида работ***: системные и программные.

Системные работы начинают и завершают процесс разработки. К ним относятся работы с номерами 2, 3, 10, 11.

Работы процесса разработки с 4-й (анализ требований к программным средствам) по 9-ю (квалификационные испытания программных средств) представляют собой *программные работы*. Они выполняются над ПС, выделенными из системы.

Таким образом, системные работы являются расширением совокупности программных работ.

Процесс эксплуатации определяет работы и задачи оператора. Данный процесс включает эксплуатацию программного продукта и поддержку пользователей в процессе эксплуатации.

Процесс сопровождения определяет работы и задачи персонала сопровождения и реализуется при модификациях программного продукта. Назначением процесса является изменение существующего программного продукта при сохранении его целостности. Процесс охватывает вопросы переносимости и снятия программного продукта с эксплуатации.

Вспомогательные процессы жизненного цикла – это процессы, являющиеся целенаправленными составными частями других процессов и предназначенные для обеспечения успешной реализации и качества выполнения программного проекта. К *вспомогательным процессам* относится восемь процессов:

- документирование;
- управление конфигурацией;
- обеспечение качества;
- верификация;
- аттестация;
- совместный анализ;
- аудит;
- решение проблем.

Вспомогательные процессы вызываются другими процессами ЖЦ.

Процесс документирования предназначен для формализованного описания информации, созданной в процессе или работе жизненного цикла. Он включает планирование, проектирование, разработку, выпуск, редактирование, распространение и сопровождение документов по программному продукту.

Процесс управления конфигурацией предназначен для определения состояния (базовой линии) программных объектов в системе, управления их изменениями и выпуском.

Процесс обеспечения качества предназначен для обеспечения гарантий того, что программные продукты и процессы в жизненном цикле проекта соответствуют требованиям и планам.

Процесс верификации предназначен для определения соответствия функционирования программных продуктов требованиям и условиям, реализованным в предшествующих работах. В процессе разработки верификация связана с экспертизой результатов конкретной работы с целью определения их соответствия установленным на входе данной работы требованиям.

Процесс аттестации предназначен для определения полноты соответствия установленных требований, созданной системы или программного продукта их функциональному назначению. В процессе разработки аттестация связана с экспертизой промежуточного или конечного продукта в целях определения его соответствия потребностям пользователя (то есть исходным требованиям к проекту).

Процесс совместного анализа предназначен для оценки состояния и результатов работ по проекту. Данный процесс может выполняться двумя сторонами, участвующими в договоре, когда одна сторона (анализирующая) проверяет другую (анализируемую).

Процесс аудита предназначен для определения соответствия требованиям, планам и условиям договора. Данный процесс может выполняться двумя сторонами, участвующими в договоре, когда одна сторона (ревизующая) проверяет другую сторону (ревизуемую).

Процесс решения проблем предназначен для анализа и решения проблем (включая найденные несоответствия), которые обнаружены в ходе выполнения разработки, эксплуатации, сопровождения или других процессов.

Организационные процессы жизненного цикла – это процессы, предназначенные для создания в некоторой организации и совершенствования организационных структур, охватывающих процессы жизненного цикла и соответствующий персонал. К *организационным процессам* относится четыре процесса:

- управление;
- создание инфраструктуры;
- усовершенствование;
- обучение.

Процесс управления состоит из общих работ и задач, которые могут быть использованы стороной, управляющей соответствующим процессом. В данном процессе разрабатываются планы выполнения процессов ЖЦ ПС, осуществляется управление и текущий надзор за ходом процессов ЖЦ, обеспечивается управление оценкой планов,

программных продуктов, работ и задач.

Процесс создания инфраструктуры предназначен для создания и сопровождения инфраструктуры, необходимой для любого другого процесса. *Инфраструктура* содержит технические и программные средства, инструментальные средства, методики, стандарты и условия для разработки, эксплуатации или сопровождения.

Процесс усовершенствования предназначен для создания, оценки, измерения, контроля и улучшения любого процесса жизненного цикла программных средств.

Процесс обучения является процессом обеспечения обучения персонала работам по заказу, поставке, разработке, эксплуатации или сопровождению программного проекта.

С понятием ЖЦ ПС и систем тесно связано понятие модели ЖЦ. ***Модель жизненного цикла*** – это совокупность процессов, работ и задач жизненного цикла, отражающая их взаимосвязь и последовательность выполнения.

Очевидно, что существуют тесные взаимосвязи между моделью ЖЦ, выбранной при реализации процесса разработки ПС, используемыми стратегиями и технологиями разработки ПС и уровнем качества разработанного программного продукта.

Резюме

Процессы ЖЦ ПС и систем регламентируются международным стандартом *ISO/IEC 12207*:*2008* и соответствующими национальными стандартами. В Республике Беларусь в настоящее время действует национальный стандарт *СТБ ИСО/МЭК 12207–2003*, являющийся аутентичным переводом стандарта *ISO/IEC 12207*:*1995*. В соответствии со стандартом *СТБ ИСО/МЭК 12207–2003* процессы ЖЦ ПС делятся на три группы: основные, вспомогательные, организационные. К основным процессам относятся процессы заказа, поставки, разработки, эксплуатации и сопровождения. Процесс разработки включает тринадцать работ.

РАЗДЕЛ 2. СТРАТЕГИИ РАЗРАБОТКИ ПРОГРАММНЫХ СРЕДСТВ И СИСТЕМ И РЕАЛИЗУЮЩИЕ ИХ МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА

2.1. Стратегии разработки программных средств и систем

2.1.1. Базовые стратегии разработки программных средств и систем

На начальном этапе развития вычислительной техники ПС разрабатывались по принципу «кодирование – устранение ошибок» [27]. Модель такого процесса разработки ПС иллюстрирует рис. 2.1.

Рис. 2.1. Модель «Делать, пока не будет сделано»

Очевидно, что *недостатками* данной модели являются003A

- неструктурированность процесса разработки ПС;
- ориентация на индивидуальные знания и умения программиста;
- сложность управления и планирования проекта;
- большая длительность и стоимость разработки;
- низкое качество программных продуктов;

- высокий уровень рисков проекта.

Для устранения или сокращения вышеназванных недостатков к настоящему времени созданы и широко используются ***три базовые стратегии*** разработки ПО:

- каскадная;
- инкрементная;
- эволюционная.

Некоторые характеристики каскадной, инкрементной и эволюционной стратегий разработки ПС и предъявляемые к ним требования приведены в стандарте ***ГОСТ Р ИСО/МЭК ТО 15271–2002 – Информационная технология – Руководство по применению ГОСТ Р ИСО/МЭК 12207 (Процессы жизненного цикла программных средств)***.

Выбор той или иной стратегии определяется характеристиками:

- проекта;
- требований к продукту;
- команды разработчиков;
- команды пользователей.

Данные характеристики подробно рассмотрены в подразд. 3.1.

Каждая из стратегий разработки имеет как достоинства, так и недостатки, определяемые правильностью выбора данной стратегии для реализации конкретного проекта. Следует подчеркнуть, что одни и те же свойства стратегии могут проявлять себя как достоинства при правильном выборе стратегии и как ее недостатки, если стратегия выбрана неверно [27].

Три базовые стратегии могут быть реализованы с помощью различных моделей ЖЦ. В подразд. 2.2 – 2.5 приведены некоторые из наиболее известных и используемых моделей ЖЦ, большинство из которых рекомендовано к использованию Институтом качества программного обеспечения SQI (Software Quality Institute) [27] или стандартом *ГОСТ Р ИСО/МЭК ТО 15271–2002*. Модели, рекомендованные Институтом SQI, в данном ЭУМКД адаптированы с учетом положений стандарта *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2).

Резюме

Существуют три базовые стратегии разработки ПС: каскадная, инкрементная, эволюционная. Данные стратегии могут быть реализованы с помощью различных

моделей ЖЦ.

2.1.2. Каскадная стратегия разработки программных средств и систем

Каскадная стратегия представляет собой однократный проход этапов разработки. Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству или системе в начале процесса разработки. Каждый этап разработки начинается после завершения предыдущего этапа. Возврат к уже выполненным этапам не предусматривается. Промежуточные продукты разработки в качестве версии программного средства (системы) не распространяются.

Представителями моделей, реализующих каскадную стратегию, являются каскадная и V-образная модели.

Основными достоинствами каскадной стратегии, проявляемыми при разработке соответствующего ей проекта, являются:

1) стабильность требований в течение ЖЦ разработки;

2) необходимость только одного прохода этапов разработки, что обеспечивает простоту применения стратегии;

3) простота планирования, контроля и управления проектом;

4) доступность для понимания заказчиками.

К *основным недостаткам каскадной стратегии*, проявляемым при ее использовании в проекте, ей несоответствующем, следует отнести:

1) сложность полного формулирования требований в начале процесса разработки и невозможность их динамического изменения на протяжении ЖЦ;

2) линейность структуры процесса разработки; разрабатываемые ПС или системы обычно слишком велики и сложны, чтобы все работы по их созданию выполнять однократно; в результате возврат к предыдущим шагам для решения возникающих проблем приводит к увеличению финансовых затрат и нарушению графика работ;

3) непригодность промежуточных продуктов для использования;

4) недостаточное участие пользователя в процессе разработки ПС – только в самом начале (при разработке требований) и в конце (во время приемочных испытаний); это приводит к невозможности предварительной оценки пользователем качества программного средства или системы.

Области применения каскадной стратегии определяются ее достоинствами и ограничены ее недостатками. Использование данной стратегии наиболее эффективно в следующих случаях [27]:

1) при разработке проектов с четкими, неизменяемыми в течение ЖЦ требованиями и понятной реализацией;

2) при разработке проектов невысокой сложности, например:

- создание программного средства или системы такого же типа, как уже разрабатывались разработчиками;
- создание новой версии уже существующего программного средства или системы;
- перенос уже существующего продукта на новую платформу;

3) при выполнении больших проектов в качестве составной части моделей ЖЦ, реализующих другие стратегии разработки (см., например, модели, приведенные на рис. 2.5 и рис. 2.12).

В подразд. 2.2 рассмотрены модели ЖЦ, реализующие каскадную стратегию разработки ПС и систем.

Резюме

Каскадная стратегия представляет собой однократный проход этапов разработки. Данная стратегия основана на полном определении всех требований к программному средству или системе в начале процесса разработки. Возврат к уже выполненным этапам не предусматривается. Промежуточные результаты в качестве версии программного средства (системы) не распространяются. Каскадная стратегия имеет достоинства и недостатки, определяемые правильностью выбора данной стратегии по отношению к конкретному проекту.

2.1.3. Инкрементная стратегия разработки программных средств и систем

Инкрементная стратегия представляет собой многократный проход этапов разработки с запланированным улучшением результата.

Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству (системе) в начале процесса разработки.

Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

Результат каждого цикла называется ***инкрементом***.

Первый инкремент реализует базовые функции программного средства. В последующих инкрементах функции программного средства постепенно расширяются, пока не будет реализован весь набор требований. Различия между инкрементами соседних циклов в ходе разработки постепенно уменьшаются.

Результат каждого цикла разработки может распространяться в качестве очередной поставляемой версии программного средства или системы.

Особенностью инкрементной стратегии является большое количество циклов разработки при незначительной продолжительности цикла и небольших различиях между инкрементами соседних циклов. Например, данная стратегия разработки ПС и систем используется в компании Microsoft. Здесь на каждую версию программного средства разрабатывается около тысячи инкрементов. Период разработки инкремента составляет одни сутки (например, днем инкремент разрабатывается, ночью тестируется) [14]. В ряде организаций используется недельный период разработки инкремента (чаще всего пять дней – разработка, два дня – тестирование).

Инкрементная стратегия обычно основана на объединении элементов каскадной модели и прототипирования. При этом использование прототипирования позволяет существенно сократить продолжительность разработки каждого инкремента и всего проекта в целом.

Под ***прототипом*** понимается легко поддающаяся модификации и расширению рабочая модель разрабатываемого программного средства (или системы), позволяющая пользователю получить представление о его ключевых свойствах до полной реализации [27].

Современной реализацией инкрементной стратегии является экстремальное программирование [14, 31]. Различные модификации моделей, реализующих инкрементную стратегию, рассмотрены в подразд. 2.4.

Основными достоинствами инкрементной стратегии, проявляемыми при разработке соответствующего ей проекта, являются:

1) возможность получения функционального продукта после реализации каждого инкремента;

2) короткая продолжительность создания инкремента; это приводит к сокращению сроков начальной поставки, позволяет снизить затраты на первоначальную и последующие поставки программного продукта;

3) предотвращение реализации громоздких спецификаций требований; стабильность требований во время создания определенного инкремента; возможность учета изменившихся требований;

4) снижение рисков по сравнению с каскадной стратегией;

5) включение в процесс пользователей, что позволяет оценить функциональные возможности продукта на более ранних этапах разработки и в конечном итоге приводит к повышению качества программного продукта, снижению затрат и времени на его разработку.

К *основным недостаткам инкрементной стратегии*, проявляющимся в результате ее несоответствующего применения, следует отнести:

1) необходимость полного функционального определения системы или программного средства в начале ЖЦ для обеспечения планирования инкрементов и управления проектом;

2) возможность текущего изменения требований к системе или программному средству, которые уже реализованы в предыдущих инкрементах;

3) сложность планирования и распределения работ;

4) проявление человеческого фактора, связанного с тенденцией к оттягиванию решения трудных проблем на поздние инкременты, что может нарушить график работ или снизить качество программного продукта.

Области применения инкрементной стратегии определяются ее достоинствами и ограничены ее недостатками. Использование данной стратегии наиболее эффективно в следующих случаях [27]:

1) при разработке проектов, в которых большинство требований можно сформулировать заранее, но часть из них могут быть уточнены через определенный период времени;

2) при разработке сложных проектов с заранее сформулированными требованиями; для них разработка системы или программного средства за один цикл связана с большими трудностями;

3) при необходимости быстро поставить на рынок продукт, имеющий базовые функциональные свойства;

4) при разработке проектов с низкой или средней степенью рисков;

5) при выполнении проекта с применением новых технологий.

Резюме

Инкрементная стратегия представляет собой многократный проход этапов разработки с запланированным улучшением результата. Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству или системе в начале процесса разработки. Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки. При инкрементной стратегии часто используется прототипирование. Инкрементная стратегия имеет достоинства и недостатки, определяемые правильностью выбора данной стратегии по отношению к конкретному проекту.

2.1.4. Эволюционная стратегия разработки программных средств и систем

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных циклах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы.

Следует отметить, что в общем случае для эволюционной стратегии характерно существенно меньшее количество циклов разработки при большей продолжительности цикла по сравнению с инкрементной стратегией. При этом результат каждого цикла разработки (очередная версия программного средства или системы) гораздо сильнее отличается от результата предыдущего цикла.

Как и при инкрементной стратегии, при реализации эволюционной стратегии зачастую используется прототипирование.

В данном случае основной целью прототипирования является обеспечение полного понимания требований. Оно позволяет итеративно уточнять требования к продукту при достижении предельно высокой производительности разработки проекта и

одновременном снижении затрат. Использование прототипирования наиболее эффективно в тех случаях, когда в проекте применяются новые концепции или новые технологии, так как в этих случаях достаточно сложно полностью и корректно разработать детальные технические требования к системе или программному средству на ранних стадиях цикла разработки.

Для итеративного уточнения требований при применении прототипирования в цикле разработки должен участвовать заказчик.

Представителями моделей, реализующих эволюционную стратегию, являются, например, спиральные модели (см. подразд. 2.5).

Основными достоинствами эволюционной стратегии, проявляемыми при разработке соответствующего ей проекта, являются:

1) возможность уточнения и внесения новых требований в процессе разработки;
2) пригодность промежуточного продукта для использования;
3) возможность управления рисками;
4) обеспечение широкого участия пользователя в проекте, начиная с ранних этапов, что минимизирует возможность разногласий между заказчиками и разработчиками и обеспечивает создание продукта высокого качества;
5) реализация преимуществ каскадной и инкрементной стратегий.

К *недостаткам эволюционной стратегии*, проявляемым при ее несоответствующем выборе, следует отнести:

1) неизвестность точного количества необходимых итераций и сложность определения критериев для продолжения процесса разработки на следующей итерации; это может вызвать задержку реализации конечной версии системы или программного средства;
2) сложность планирования и управления проектом;
3) необходимость активного участия пользователей в проекте, что реально не всегда осуществимо;
4) необходимость в мощных инструментальных средствах и методах прототипирования;
5) возможность отодвигания решения трудных проблем на последующие циклы, что может привести к несоответствию полученных продуктов требованиям заказчиков.

Очевидно, что ряд недостатков эволюционной стратегии (см. недостатки 3 – 5) характерны и для инкрементной стратегии.

Области применения эволюционной стратегии определяются ее достоинствами и ограничены ее недостатками. Использование данной стратегии наиболее эффективно в следующих случаях [27]:

1) при разработке проектов, для которых требования слишком сложны, неизвестны заранее, непостоянны или требуют уточнения;
2) при разработке сложных проектов, в том числе:
   - больших долгосрочных проектов;
   - проектов по созданию новых, не имеющих аналогов ПС или систем;
   - проектов со средней и высокой степенью рисков;
   - проектов, для которых нужна проверка концепции, демонстрация технической осуществимости или промежуточных продуктов;
3) при разработке проектов, использующих новые технологии.

В подразд. 2.5 рассмотрены модели ЖЦ, реализующие эволюционную стратегию разработки ПС и систем.

Резюме

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных цик-лах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы. При эволюционной стратегии часто используется прототипирование. Эволюционная стратегия имеет достоинства и недостатки, определяемые правильностью выбора данной стратегии по отношению к конкретному проекту.

2.2. Модели жизненного цикла, реализующие каскадную стратегию разработки программных средств и систем

2.2.1. Общие сведения о каскадных моделях

Моделью ЖЦ, пришедшей на смену принципу разработки ПС «кодирование – устранение ошибок», явилась классическая каскадная модель. Первые публикации о ней появились в 1970 г. Данная модель впервые формализовала структуру этапов разработки ПС. Она поддерживает *каскадную стратегию однократного прохода этапов разработки ПС* и базируется на полном формулировании требований в начале ЖЦ. К их уточнению или изменению на следующих шагах ЖЦ возврата не происходит.

Процесс разработки ПС и систем реализуется с помощью упорядоченной последовательности независимых шагов. Модель предусматривает, что каждый последующий шаг начинается после полного завершения выполнения предыдущего шага.

Существуют различные варианты каскадной модели ЖЦ.

Резюме

Каскадные модели реализуют каскадную стратегию однократного прохода этапов разработки ПС. Каждый последующий шаг разработки начинается после полного завершения выполнения предыдущего шага.

2.2.2. Классическая каскадная модель

В общем случае каскадные модели могут представлять процесс разработки с различной степенью детализации. При этом в качестве шага модели могут быть приняты некоторые фазы, этапы или работы процесса разработки.

Рис. 2.2 представляет вариант классической каскадной модели, ориентированный на работы процесса разработки, структура которого определена в *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2).

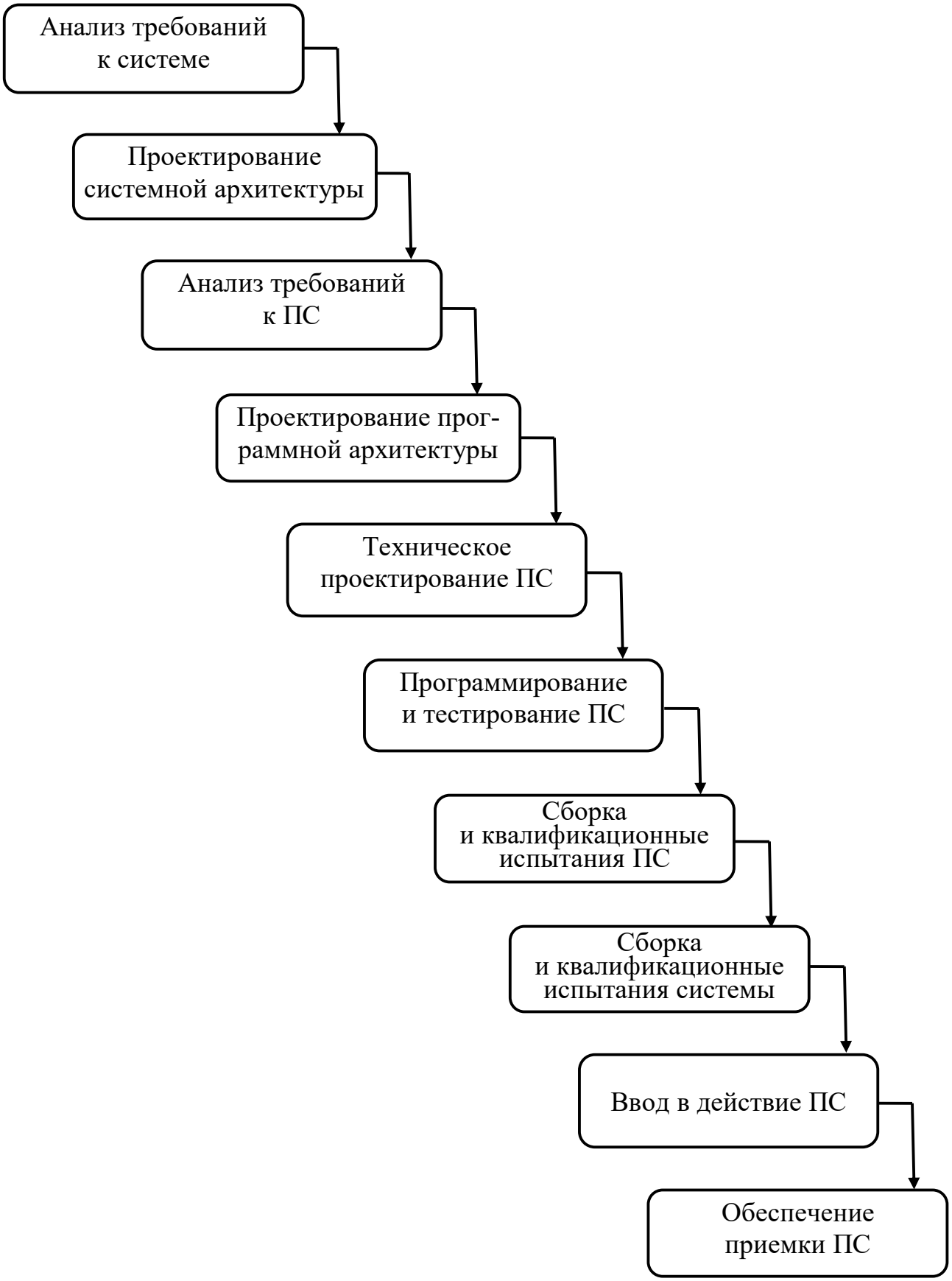

Рис. 2.2. Классическая каскадная модель,

ориентированная на работы процесса разработки СТБ ИСО/МЭК 12207–2003

Для данного варианта модели понятие шага разработки программного средства совпадает с понятием одной или нескольких работ процесса разработки вышеназванного стандарта.

С учетом внешнего вида каскадной модели (см. рис. 2.2) ее называют также *водопадной моделью*.

На всех шагах модели по необходимости выполняются вспомогательные и организационные процессы, например, управление проектом, обеспечение качества, верификация, аттестация, управление конфигурацией, документирование (см. подразд. 1.2).

Результатами завершения шагов модели являются промежуточные продукты разработки, которые не могут изменяться на последующих шагах и не могут сдаваться заказчику в качестве версий программного средства.

Классическая каскадная модель обладает всеми достоинствами и недостатками, характерными для каскадной стратегии разработки (см. п. 2.1.2).

Резюме

В каскадных моделях возможна различная степень детализации процесса разработки ПС. Рассмотренный вариант классической каскадной модели базируется на работах процесса разработки, определенного в стандарте *СТБ ИСО/МЭК 12207–2003*. В каскадной модели продукты промежуточных шагов разработки не могут изменяться на последующих шагах и не могут сдаваться заказчику.

2.2.3. Каскадная модель с обратными связями

Реализовать классическую каскадную модель ЖЦ в чистом виде затруднительно ввиду сложности разработки ПС без возвратов к предыдущим шагам и изменения их результатов для устранения возникающих проблем. В этой связи разработаны варианты каскадной модели с обратными связями между ее отдельными шагами.

Рис. 2.3 отражает организацию обратных связей между соседними шагами модели, ориентированной на работы стандарта *СТБ ИСО/МЭК 12207–2003*.

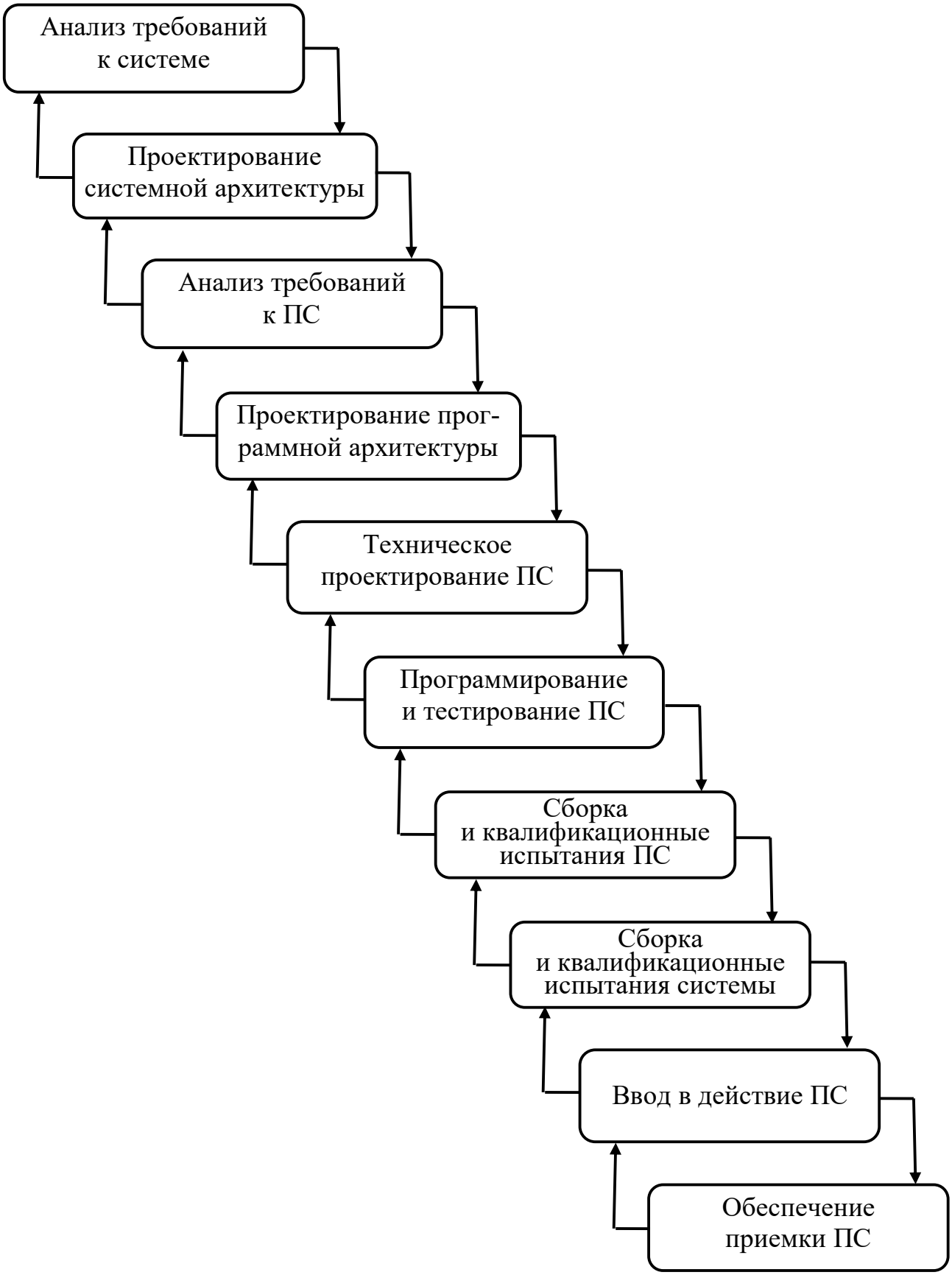

Рис. 2.3. Каскадная модель с обратными связями, ориентированная на работы процесса разработки СТБ ИСО/МЭК 12207–2003

Возможна организация обратных связей между любыми шагами каскадной модели. Рис. 2.4 схематично иллюстрирует фрагмент каскадной модели с возможностью возврата от некоторого шага процесса разработки к различным шагам, выполненным ранее.

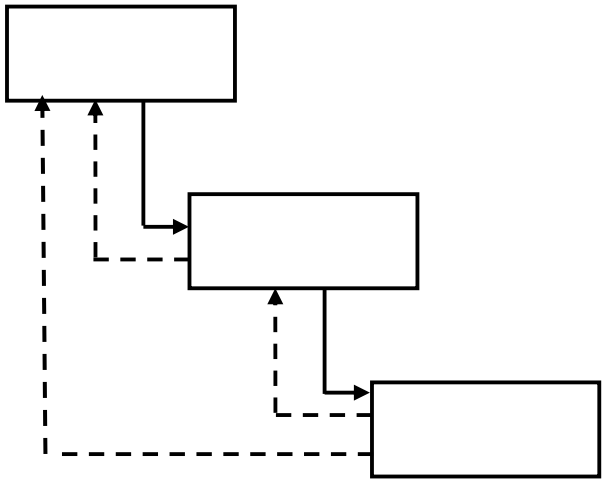

Рис. 2.4. Структура фрагмента каскадной модели с возможностью возврата к различным шагам

Достоинством каскадной модели с обратными связями по сравнению с классической каскадной моделью является возможность исправления продуктов предыдущих шагов процесса разработки.

К *недостаткам* каскадной модели с обратными связями по сравнению с классической каскадной моделью относятся сложности планирования и финансирования проекта, достаточно высокий риск нарушения графика разработки.

Резюме

Возможна организация обратных связей между любыми шагами каскадной модели. Это позволяет выполнять исправление промежуточных продуктов предыдущих шагов процесса разработки. При этом возрастает сложность пла нирования и финансирования проекта, достаточно высок риск нарушения графика разработки.

2.2.4. Вариант каскадной модели по ГОСТ Р ИСО/МЭК ТО 15271–2002

В *ГОСТ Р ИСО/МЭК ТО 15271–2002* приведен вариант каскадной модели, ориентированный на коллективную разработку систем (рис. 2.5). В данном варианте, как и в рассмотренных ранее вариантах модели, понятие шага разработки совпадает с понятием работы процесса разработки, определенного в стандарте *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2).

Приведенный вариант базируется на возможности параллельной разработки ПС, входящих в состав системы, различными командами разработчиков, а также выбора программных объектов из существующих или разработанных ранее.

Данный вариант учитывает также необходимость разработки или выбора технических компонентов системы.

Резюме

На основе каскадной модели возможна организация параллельной разработки ПС, входящих в состав системы, различными командами разработчиков.

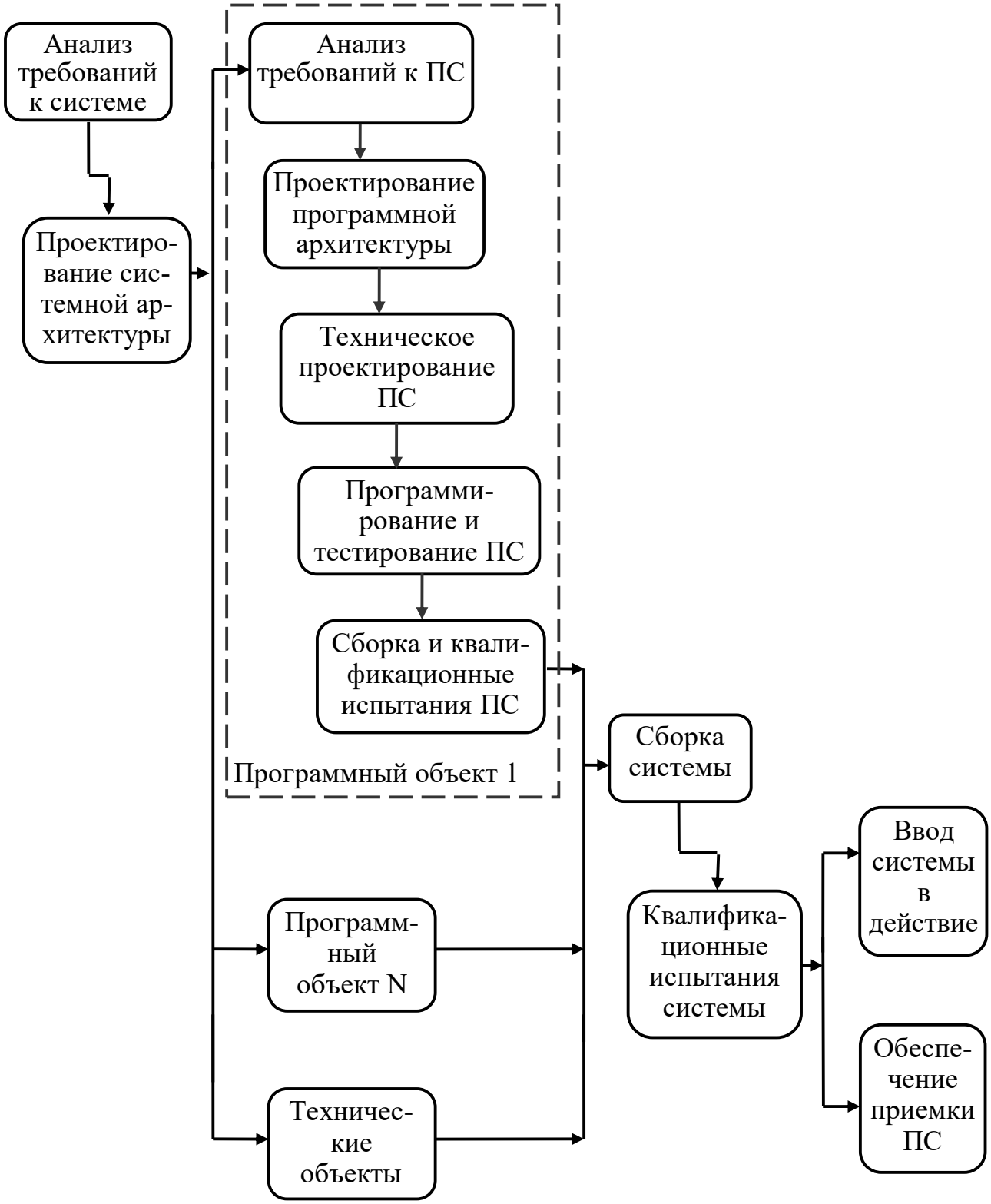

Рис. 2.5. Вариант каскадной модели по ГОСТ Р ИСО/МЭК ТО 15271–2002

2.2.5. V-образная модель

Основное назначение V-образной модели – обеспечение планирования тестирования (испытаний) системы и программного средства на ранних стадиях проекта.

V-образная модель представляет собой разновидность каскадной модели. Данная модель поддерживает каскадную стратегию однократного выполнения этапов процесса разработки ПС или систем и базируется на предварительном полном формировании требований. В классической V-образной модели каждый шаг начинается после завершения предыдущего шага.

Отличием V-образной модели от каскадной является то, что в ней выделены связи между шагами, предшествующими программированию, и соответствующими видами тестирования и испытаний.

Рис. 2.6 иллюстрирует вариант V-образной модели, адаптированный к работам процесса разработки, определенного в *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2). Данная модель состоит из последовательных этапов.

Этапы *подготовки процесса разработки, анализа требований к системе* и *программирования и тестирования программных средств* соответствуют работам 1, 2 и 7 процесса разработки (см. подр. 1.2).

Кроме того, на этапе *анализа требований к системе* разрабатывается план ввода в действие и обеспечения приемки системы.

На этапе проектирования системы выполняются работы 3 (проектирование системной архитектуры) и 4 (анализ требований к программным средствам) процесса разработки. На данном этапе, помимо этого, разрабатываются планы сборки и квалификационных испытаний системы.

На этапе проектирования программных средств выполняются работы 5 (проектирование программной архитектуры) и 6 (техническое проектирование программных средств). Одновременно составляются план сборки ПС и план квалификационных испытаний ПС.

На этапе сборки и квалификационных испытаний программных средств выполняются работы 8 (сборка программных средств) и 9 (квалификационные испытания программных средств). Данный этап выполняется в соответствии с планами, разработанными на этапе проектирования программных средств, и имеет одной из основных целей подтверждение результатов названного этапа.

Рис. 2.6. V-образная модель жизненного цикла

На этапе сборки и квалификационных испытаний системы выполняются работы 10 (сборка системы) и 11 (квалификационные испытания системы).

Данный этап выполняется в соответствии с планами, разработанными на этапе проектирования системы. Подтверждение результатов последнего является одной из основных целей данного этапа.

На этапе ввода в действие и обеспечения приемки осуществляются работы 12 (ввод в действие программных средств), 13 (обеспечение приемки программных средств), а также при необходимости работы по вводу в действие и обеспечению

приемки всей системы в целом. Этот этап выполняется в соответствии с планом, разработанным на этапе анализа требований к системе. На данном этапе выполняются приемочные испытания, целью которых является проверка пользователем соответствия системы исходным требованиям.

Связи между деятельностью по разработке планов испытаний и тестирования и деятельностью по подтверждению результатов соответствующих этапов на рис. 2.6 обозначены пунктирными линиями.

Как и в каскадной модели, на всех этапах V-образной модели выполняются необходимые вспомогательные процессы ЖЦ ПС (см. подразд. 1.2), например, управление проектом, оценка качества, верификация, аттестация, управление конфигурацией, документирование.

С целью сокращения недостатков каскадной стратегии разработан ряд модификаций V-образной модели, обусловленных разновидностью обратных связей, которые обеспечивают возможность изменения результатов предыдущих этапов разработки.

Например, рис. 2.7 иллюстрирует V-образную модель с организацией обратных связей между соседними этапами процесса разработки.

Поскольку V-образная модель поддерживает каскадную стратегию разработки ПС и систем, то она обладает всеми достоинствами данной стратегии. Кроме того, при подходящем использовании V-образная модель обладает следующими дополнительными ***достоинствами***:

1) планирование тестирования и испытаний на ранних стадиях разработки системы и программного средства;

2) упрощение аттестации и верификации промежуточных результатов разработки;

3) упрощение управления и контроля хода процесса разработки.

При использовании V-образной модели для несоответствующего ей проекта выявляются следующие ее ***недостатки***:

1) поздние сроки тестирования требований в жизненном цикле, что оказывает существенное влияние на график выполнения проекта при необходимости изменения требований;

2) отсутствие, как и в остальных каскадных моделях, действий, направленных на анализ рисков.

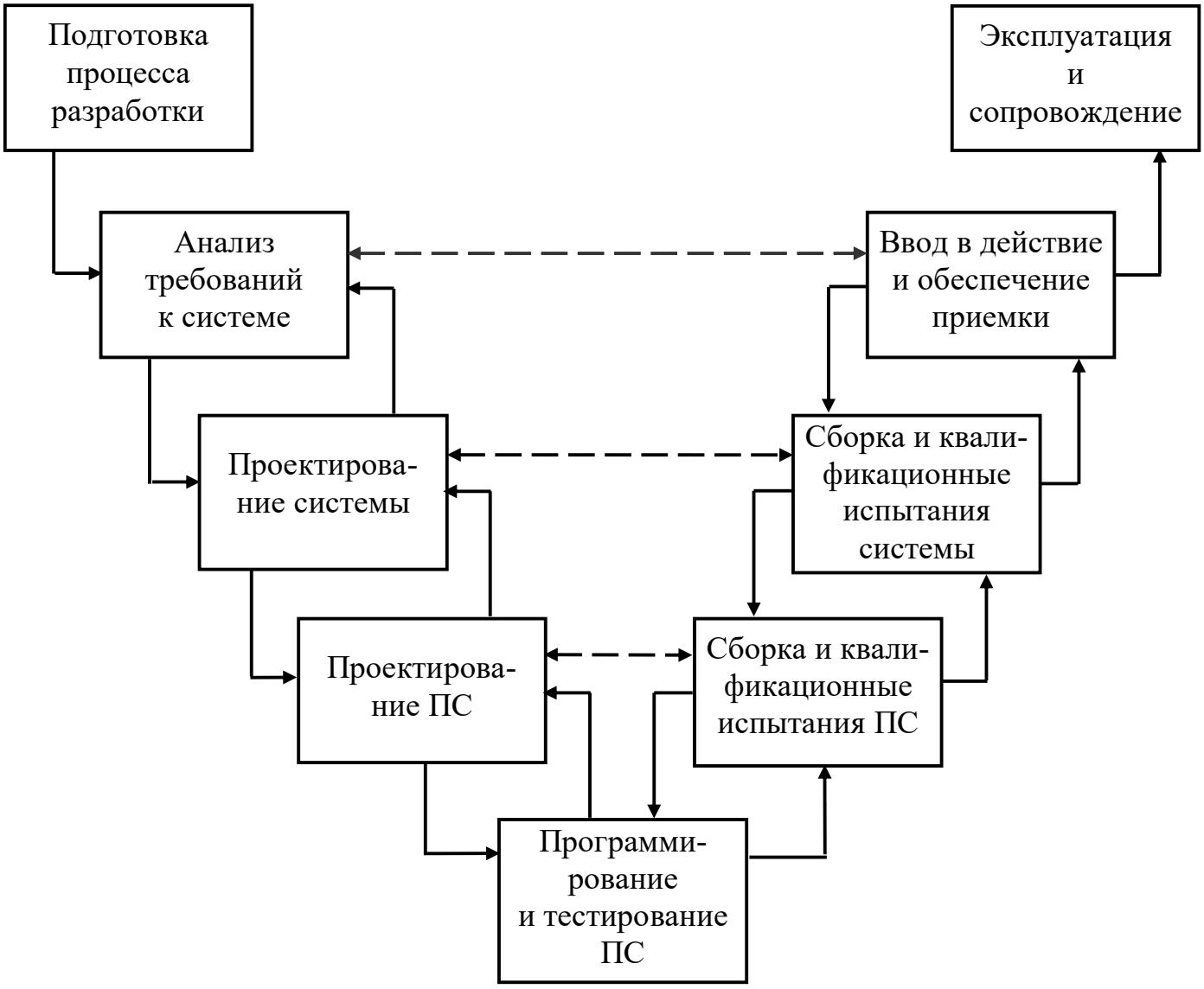

Рис. 2.7. V-образная модель жизненного цикла
с организацией обратных связей между соседними этапами

Резюме

V-образная модель поддерживает каскадную стратегию разработки. В ней выделены связи между шагами, предшествующими программированию, и соответствующими видами тестирования и испытаний. Данные связи увязывают деятельность по разработке планов испытаний и тестирования с деятельностью по подтверждению результатов соответствующих этапов. В V-образной модели возможна организация обратных связей между этапами модели.

2.3. Модели быстрой разработки приложений

Модель быстрой разработки приложений (Rapid Application Development, RAD) появилась в 80-е гг. XX в. в связи с бурным развитием мощных технологий и инструментальных средств разработки программных продуктов. Данная модель, исходя из особенностей ее реализации и целей ее использования, может поддерживать как инкрементную, так и эволюционную стратегию разработки систем или ПС. Как правило, RAD-модели используются в составе другой модели для ускорения цикла разработки прототипа (версии) системы или программного средства (см. пп. 2.5.3, 2.5.4). При невысокой сложности проектов RAD-модели могут применяться как независимые модели.

Как было отмечено в подразд. 2.1, модели ЖЦ, реализующие инкрементную или эволюционную стратегию разработки, широко применяют понятие быстрого прототипирования. RAD-модель представляет собой модель, на использовании которой прототипирование базируется.

Основу RAD-модели составляет использование мощных *инструментальных средств разработки*. Такими средствами являются языки четвертого поколения *4GL* (Fourth Generation Language – язык программирования четвертого поколения) и *CASE-средства* (Computer Aided Software Engineering – компьютерная поддержка проектирования ПО), благодаря наличию в них сред визуальной разработки и кодогенераторов (подробно понятия CASE-средств и CASE-технологий рассмотрены в подразд. 5.1 и разд. 6). Поэтому в процессе быстрой разработки приложений основное внимание уделяется не программированию и тестированию, а анализу требований и проектированию.

Использование инструментальных средств позволяет задействовать пользователя, а следовательно, дать оценку продукту на всех этапах его разработки.

Характерной чертой RAD-модели является короткое время перехода от анализа требований до создания полной системы или программного средства. Разработка прототипа, как правило, ограничивается четко определенным периодом времени (*временным блоком*; обычно 60 дней) [23, 27]. При полностью определенных

требованиях и не очень сложной проектной области использование RAD-модели позволяет за временной блок создать полную функциональную систему.

Резюме

Модель быстрой разработки приложений может поддерживать как инкрементную, так и эволюционную стратегию разработки систем или ПС. Обычно RAD-модель применяется в составе другой модели для ускорения цикла разработки прототипа системы или программного средства. Основу RAD-модели составляет использование мощных инструментальных средств разработки. Разработка прототипа ограничивается временным блоком.

2.3.1. Базовая RAD-модель

Рис. 2.8 представляет вариант базовой модели быстрой разработки приложений. Данный вариант отражает зависимость трудозатрат по разработке и участия пользователя от этапов RAD-модели [27].

На этапе анализа требований к системе совместно с заказчиком (пользователем) выполняется анализ предметной области, сбор и разработка требований к системе (работа 2 процесса разработки, определенного в стандарте *СТБ ИСО/МЭК 12207–2003,* см. подразд. 1.2). При этом используются соответствующие инструментальные средства (см. разд. 6).

На этапе проектирования выполняются работы 3 – 5 процесса разработки (проектирование системной архитектуры, анализ требований к программным средствам, проектирование программной архитектуры). При этом используются инструментальные средства, обеспечивающие сбор и уточнение пользовательской информации, визуальные анализ и проектирование.

На этапе конструирования выполняются работы 6 – 11 процесса разработки (техническое проектирование программных средств, программирование и тестирование программных средств, сборка и квалификационные испытания программных средств и системы). При этом используются соответствующие инструментальные средства проектирования, автоматической кодогенерации и тестирования.

На этапе ввода в действие и обеспечения приемки выполняются установка системы или программного средства, приемочные испытания и обучение пользователей (работы 12, 13 процесса разработки).

Данный вариант RAD-модели является обобщенным и не учитывает специфику выполнения отдельных этапов процесса разработки. Этот вариант может использоваться в отдельных итерациях некоторой эволюционной модели и в качестве независимой модели в небольших проектах.

Резюме

Базовая RAD-модель отражает укрупненные этапы процесса разработки (анализ требований, проектирование, конструирование, ввод в действие и обеспечение приемки). Наибольшие трудозатраты разработчика и наименьшее участие пользователя приходятся на этап конструирования.

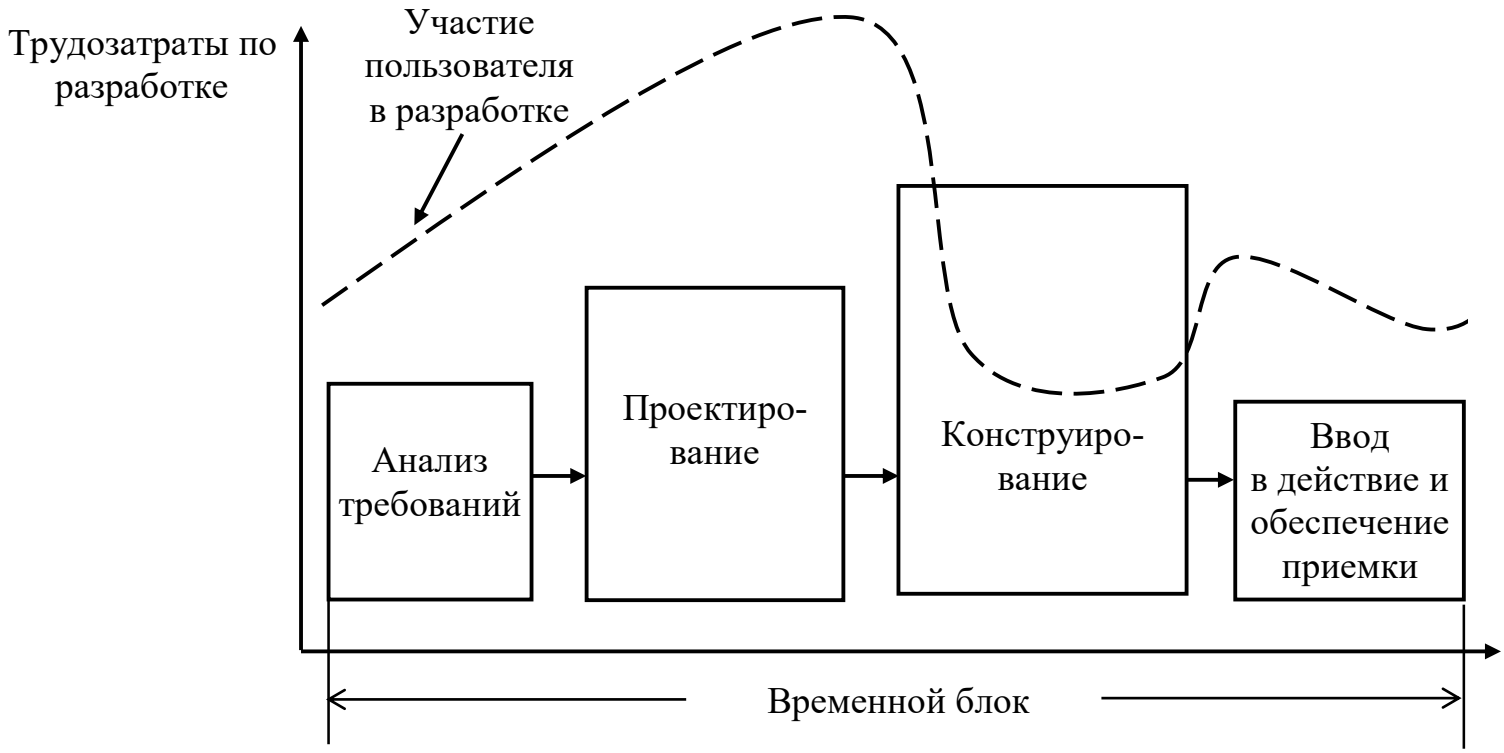

Рис. 2.8. Базовая модель быстрой разработки приложений

2.3.2. RAD-модель, основанная на моделировании предметной области

В RAD-модели для ускорения процесса разработки обычно используются различные виды моделирований предметной области, например, функциональное

моделирование, моделирование данных, моделирование процесса (поведения). С этой целью широко используются CASE-средства (см. разд. 6). Затем на основе созданных моделей выполняется автоматическая кодогенерация программного средства. С учетом этого разработаны варианты RAD-модели, базирующиеся на моделировании предметной области. Один из вариантов приведен на рис. 2.9.

В данной модели выделяется пять этапов.

На *этапе функционального моделирования* определяются и анализируются функции и информационные потоки предметной области. В подразд. 5.2. и 5.3 рассмотрены методологии функционального моделирования IDEF0 и DFD.

На этапе моделирования данных на базе информационных потоков, определенных на предыдущем этапе, разрабатывается информационная модель предметной области. В подразд. 5.4 рассмотрена методология информационного моделирования IDEF1X.

На этапе моделирования поведения выполняется динамическое (поведенческое) моделирование предметной области. Одной из известных методологий динамического моделирования является методология IDEF3.

На этапе автоматической кодогенерации на основе информационной, функциональной и поведенческой моделей выполняется генерация текстов программных компонентов. При этом используются языки программирования четвертого поколения (Fourth Generation Language – 4GL) и CASE-средства. Широко применяются повторно используемые программные компоненты.

На этапе сборки и квалификационных испытаний выполняется сборка и испытания результирующей системы, подсистемы или программного средства.

При независимом использовании рассмотренный вариант RAD-модели поддерживает стратегию однократного прохода этапов процесса разработки, то есть фактически каскадную стратегию. В этом случае он может применяться для проектов невысокой сложности.

В проектах более высокой сложности данный вариант может встраиваться в инкрементные и эволюционные модели как основа реализации отдельных итераций.

Резюме

В RAD-модели, основанной на моделировании предметной области, этапы анализа требований, проектирования и программирования (работы 2 – 7 процесса разработки, см. подразд. 1.2) заменены этапами функционального моделирования, моделирования

данных, моделирования поведения разрабатываемого программного средства или системы и автоматической кодогенерации. Это существенно ускоряет процесс разработки.

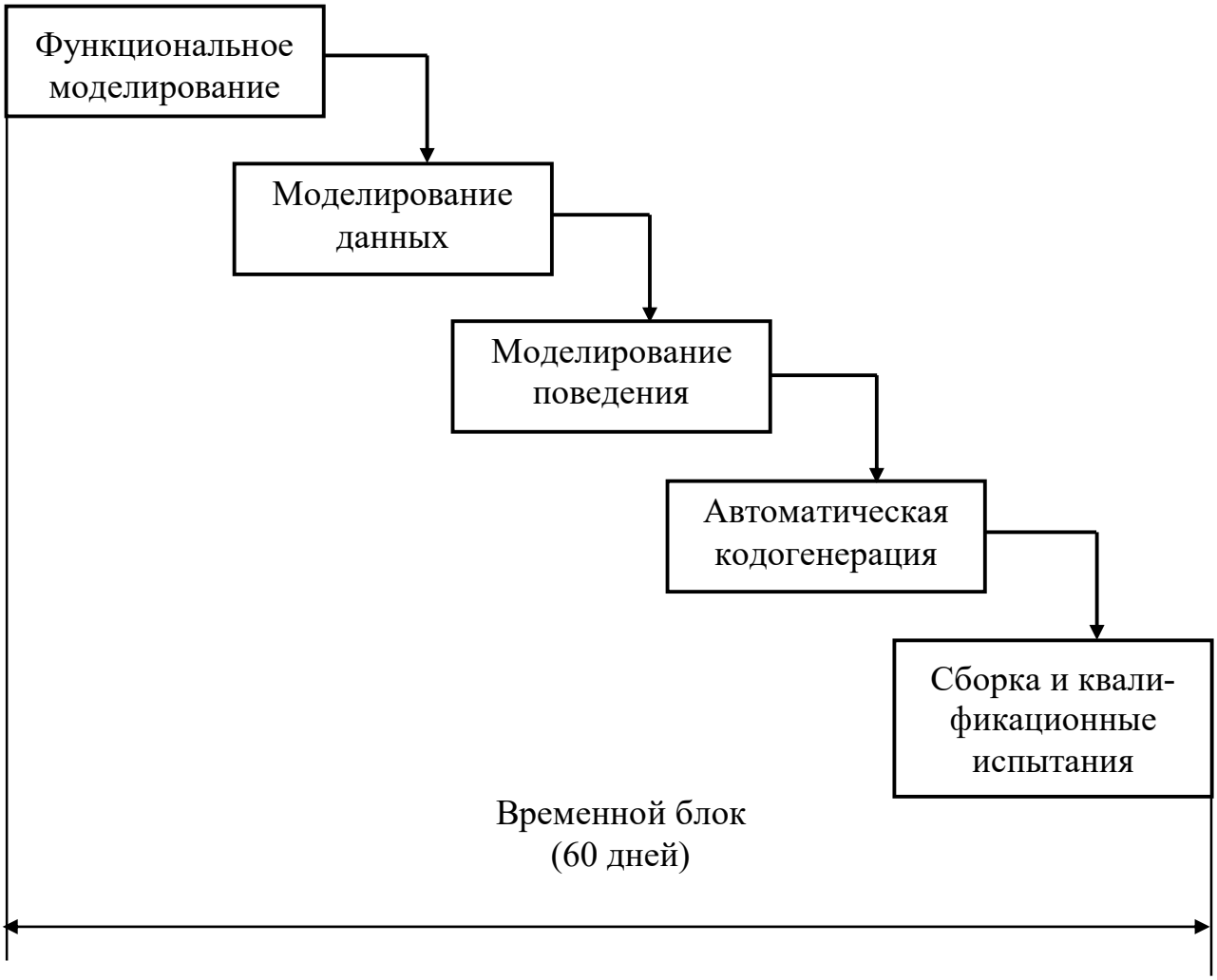

Рис. 2.9. Вариант модели быстрой разработки приложений, основанной на моделировании предметной области

2.3.3. RAD-модель параллельной разработки приложений

При разработке сложных проектов с организацией коллективной разработки ПС могут использоваться различные варианты RAD-модели [23]. Один из вариантов представлен на рис. 2.10.

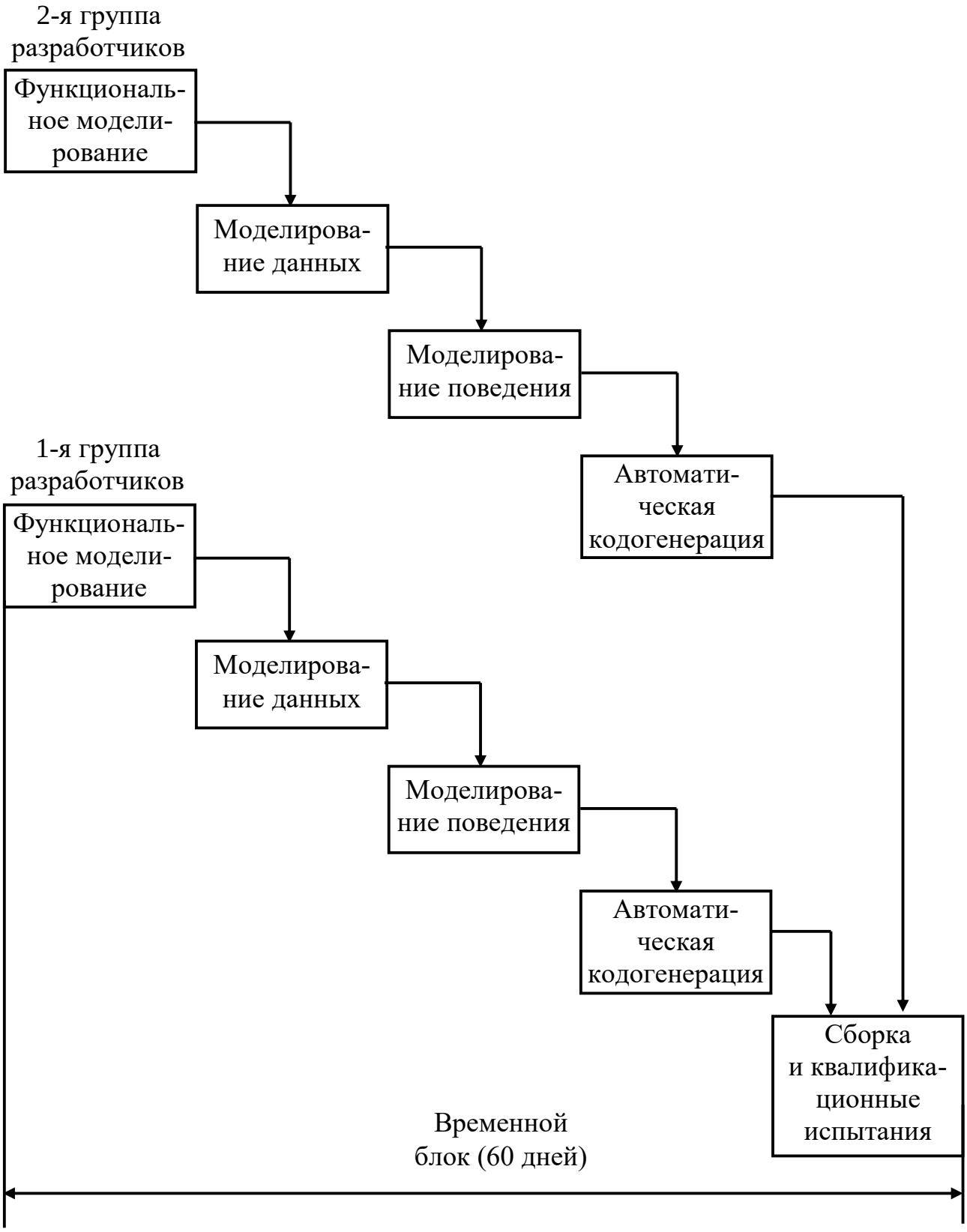

Рис. 2.10. Вариант RAD-модели,
основанный на независимой работе групп разработчиков

Данный вариант модели поддерживает параллельную разработку программных компонентов, реализующих базовые функции программного средства различными группами разработчиков, с дальнейшей интеграцией разработанных компонентов в единую систему или программное средство.

2.3.4. Модель быстрой разработки приложений по ГОСТ Р ИСО/МЭК ТО 15271–2002

На рис. 2.11 представлен вариант RAD-модели, приведенный в стандарте *ГОСТ Р ИСО/МЭК ТО 15271–2002* и адаптированный под требования стандарта *ISO/IEC 12207*:*1995* (*СТБ ИСО/МЭК 12207–2003*). Числами в скобках обозначены номера работ процесса разработки, используемые на соответствующих этапах модели (см. подразд. 1.2), ***** обозначает процесс эксплуатации.

На этапе осуществимости выполняется анализ проекта на основе критических факторов успешности реализации RAD-модели. К данным факторам могут быть отнесены, например, области применения RAD-модели (см. п. 2.3.5).

На этапе анализа деловой деятельности определяется область применения проекта и разрабатывается план прототипирования.

Назначением *цикла функциональной модели* является уточнение функциональных требований к разрабатываемой системе (программному средству). В данном цикле с помощью инструментальных средств быстрой разработки (CASE-средств) разрабатываются функциональные прототипы. Кроме того, формулируются нефункциональные требования и стратегия реализации полного прототипа.

В цикле проектирования и создания на основе функционального прототипа последовательно проектируются, реализуются и анализируются заказчиком прототипы, все более полно учитывающие нефункциональные требования к разрабатываемой системе или программному средству. В конечном итоге создаются ПС, прошедшие квалификационные испытания и удовлетворяющие всем функциональным и нефункциональным требованиям заказчика.

На этапе реализации выполняется внедрение ПС в среде пользователя и обучение соответствующего персонала.

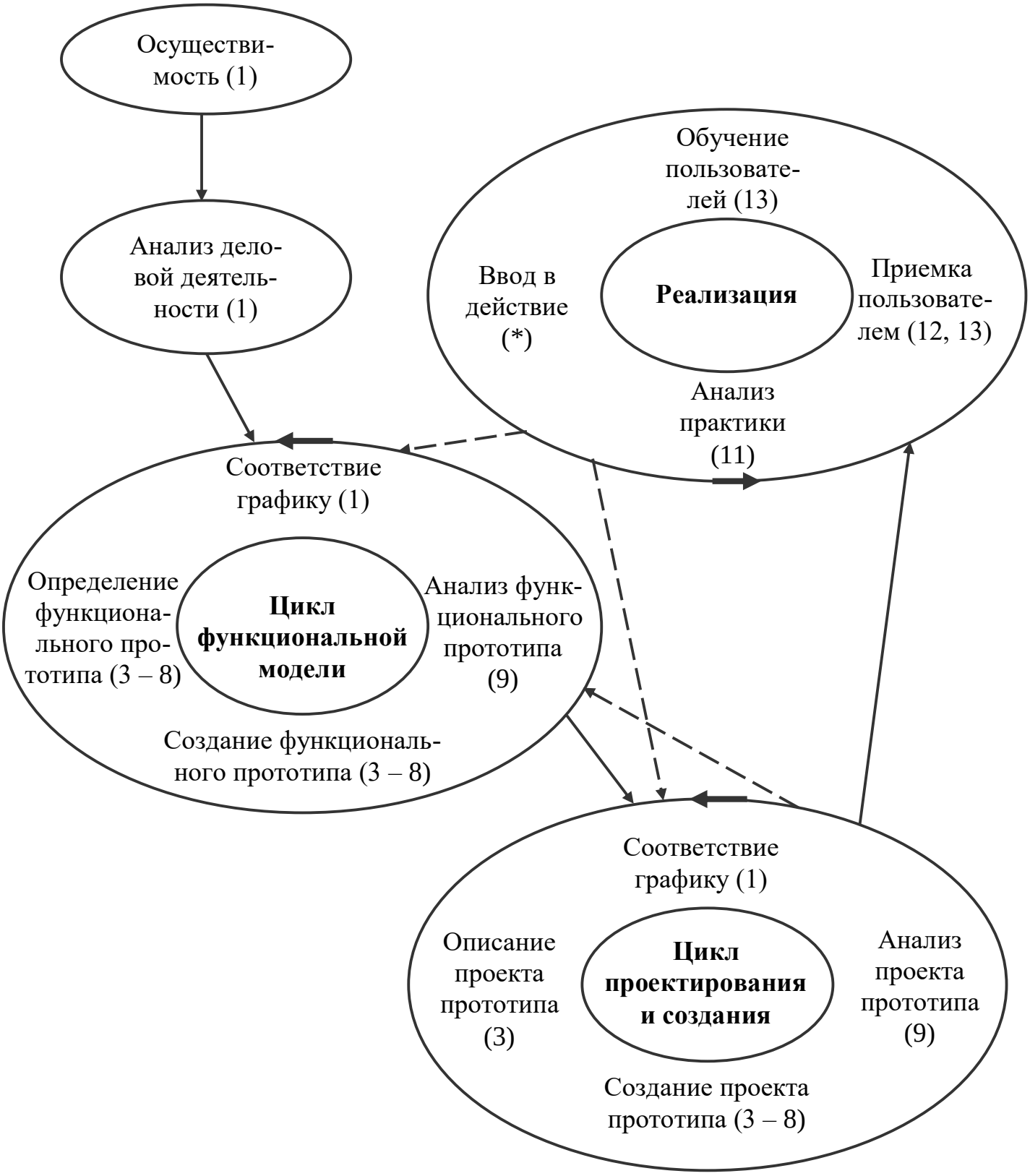

Рис. 2.11. Вариант модели быстрой разработки приложений
по ГОСТ Р ИСО/МЭК ТО 15271–2002

Цикл функциональной модели и цикл проектирования и создания предусматривают последовательную итерацию работ 3 – 9 процесса разработки, регламентированного стандартом *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2). При этом каждая итерация прототипа должна быть создана в соответствии с графиком работ за определенный временной интервал. Конкретное время реализации каждого шага итерационного цикла, как правило, определяется набором *трех итераций* – предварительное исследование, уточнение и утверждение (принятие) каждого прототипа.

Из описания приведенного варианта RAD-модели следует, что он фактически реализует эволюционную стратегию разработки ПС. В каждом цикле разработки реализуется уточнение функциональных и нефункциональных требований, разработка функционального прототипа, разработка версии программного средства, ввод в действие и эксплуатация данной версии.

Резюме

Приведенный вариант RAD-модели адаптирован под требования стандарта *ISO/IEC 12207*:*1995* (*СТБ ИСО/МЭК 12207–2003*). Данный вариант базируется на использовании цикла функциональной модели и цикла проектирования и создания. Данные циклы выполняются итерационно.

2.3.5. Достоинства, недостатки и области использования RAD-моделей

При использовании RAD-модели в соответствующем ей проекте проявляются следующие ее *основные достоинства*:

1) сокращение продолжительности цикла разработки и всего проекта в целом, сокращение количества разработчиков, а следовательно, и стоимости проекта за счет использования мощных инструментальных средств;

2) сокращение риска несоблюдения графика за счет использования принципа временного блока и связанное с этим упрощение планирования;

3) сокращение риска, связанного с неудовлетворенностью заказчика (пользователя) разработанным программным продуктом, за счет его привлечения на постоянной основе к циклу разработки; возрастание уверенности, что продукт будет соответствовать требованиям;

4) возможность повторного использования существующих компонентов; это достоинство проявляется при использовании RAD-модели в составе инкрементной или эволюционной модели; в этом случае наращивание функциональных возможностей осуществляется на базе разработанных ранее компонентов.

*Основными недостатк*ами RAD-модели при использовании в неподходящем для нее проекте являются:

1) необходимость в постоянном участии пользователя в процессе разработки, что часто невыполнимо и в итоге сказывается на качестве конечного продукта;

2) необходимость в высококвалифицированных разработчиках, умеющих работать с инструментальными средствами разработки;

3) возможность применения только для систем или ПС, для которых отсутствует требование высокой производительности;

4) жесткость временных ограничений на разработку прототипа;

5) сложность ограничения затрат и определения сроков завершения работы над проектом в случаях, когда не удается разработать продукт за временной интервал;

6) неприменимость в условиях высоких технических рисков, при использовании новых технологий.

Области применения RAD-модели определяются ее достоинствами и ограничены ее недостатками. RAD-модель может эффективно применяться в следующих случаях:

1) при разработке систем и продуктов, для которых характерно хотя бы одно из следующих свойств:

- поддаются моделированию;
- предназначены для концептуальной проверки;
- являются некритическими;
- имеют небольшой размер;
- имеют низкую производительность;
- относятся к известной разработчикам предметной области;
- являются информационными системами;
- требования для них хорошо известны;
- имеются пригодные к повторному использованию в них компоненты;

2) если пользователь может принимать постоянное участие в процессе разработки;

3) если в проекте заняты разработчики, обладающие достаточными навыками в использовании инструментальных средств разработки;

4) при выполнении проектов в сокращенные сроки (как правило, не более чем за 60 дней);

5) при разработке ПС, для которых требуется быстрое наращивание функциональных возможностей на последовательной основе;

6) при невысокой степени технических рисков;

7) в составе других моделей жизненного цикла.

Резюме

Использование RAD-модели сокращает количество разработчиков, продолжительность и стоимость разработки ПС и систем, сокращает риски. В то же время ее применение требует высокой квалификации разработчиков и постоянного участия пользователей в процессе разработки ПС и систем.

2.4. Модели жизненного цикла, реализующие инкрементную стратегию разработки программных средств и систем

2.4.1. Общие сведения об инкрементных моделях

Инкрементные модели поддерживают инкрементную стратегию разработки ПС и систем. Данная стратегия представляет собой запланированное улучшение продукта в процессе его ЖЦ (см. п. 2.1.3). При использовании инкрементных моделей осуществляется изначальная частичная реализация всей системы (или программного средства). При этом в первую очередь реализуются базовые требования. За этим следует медленное наращивание функциональных возможностей или характеристик качества системы или программного средства в реализуемых последовательно прототипах (инкрементах).

Применение инкрементных моделей ускоряет создание программного средства или системы за счет применяемого принципа компоновки из стандартных или разработанных

ранее программных компонентов. Это позволяет существенно уменьшить затраты на разработку.

Существуют различные варианты реализации инкрементных моделей.

В классических вариантах модель основана на использовании полного заранее сформированного набора требований, реализуемых последовательно в виде небольших инкрементов.

Существуют варианты модели, начинающиеся с формулирования общих требований. Требования постепенно уточняются в процессе разработки прототипов. Данные варианты инкрементных моделей похожи на эволюционные модели. Однако от последних они отличаются существенно большим количеством инкрементов при гораздо меньших различиях между соседними инкрементами. К таким вариантам моделей относится, например, модель ЖЦ, реализущая современную реализацию инкрементной стратегии – экстремальное программирование (см. п. 2.4.4).

При использовании инкрементной модели различия между последовательными прототипами постепенно уменьшаются. В каждой последующей версии системы или программного средства добавляются к предыдущей версии определенные программные компоненты, реализующие соответствующие функциональные возможности, до тех пор, пока не будут реализованы все требования к системе (программному средству). При этом каждая версия системы или программного средства может сдаваться в эксплуатацию.

Резюме

В классических вариантах инкрементная модель основана на использовании полного заранее сформированного набора требований, реализуемых последовательно в виде небольших проектов. Существуют варианты инкрементной модели, начинающиеся с формулирования общих целей, которые постепенно уточняются в процессе разработки прототипов.

2.4.2. Инкрементная модель с уточнением требований на начальных этапах разработки

На рис. 2.12 представлена модель, адаптированная к структуре процесса разработки, определенного в стандарте *СТБ ИСО/МЭК 12207–2003* (см. подразд. 1.2), с

возможностью изменения или уточнения требований на начальных этапах процесса разработки системы или программного средства.

На ранних этапах процесса разработки (анализ требований к системе, проектирование системной архитектуры, анализ требований к программным средствам) выполняется проектирование системы в целом. При этом используется каскадная модель с обратными связями между этапами. Применение обратных связей позволяет производить уточнение требований к системе, выполнять проектирование ее архитектуры с учетом изменившихся требований, уточнять требования к ПС [27]. На этих этапах определяются инкременты и реализуемые ими функции.

Каждый инкремент затем проходит через остальные этапы жизненного цикла (проектирование ПС, программирование и тестирование ПС, сборка и квалификационные испытания ПС и системы, ввод в действие и обеспечение приемки ПС и содержащей их системы, эксплуатация и сопровождение ПС). Выполнение данных этапов соответствует каскадной модели ЖЦ и может быть распределено согласно календарному графику.

В модели в первую очередь реализуется набор функциональных и нефункциональных требований, формирующих основу продукта. Последующие инкременты улучшают функциональные возможности или характеристики про граммного средства. Каждый инкремент верифицируется в соответствии с набором требований, предъявляемых к данному инкременту.

Резюме

В рассмотренном варианте инкрементной модели возможно уточнение требований на начальных этапах процесса разработки системы. Разработка каждого инкремента реализуется на базе каскадной стратегии.

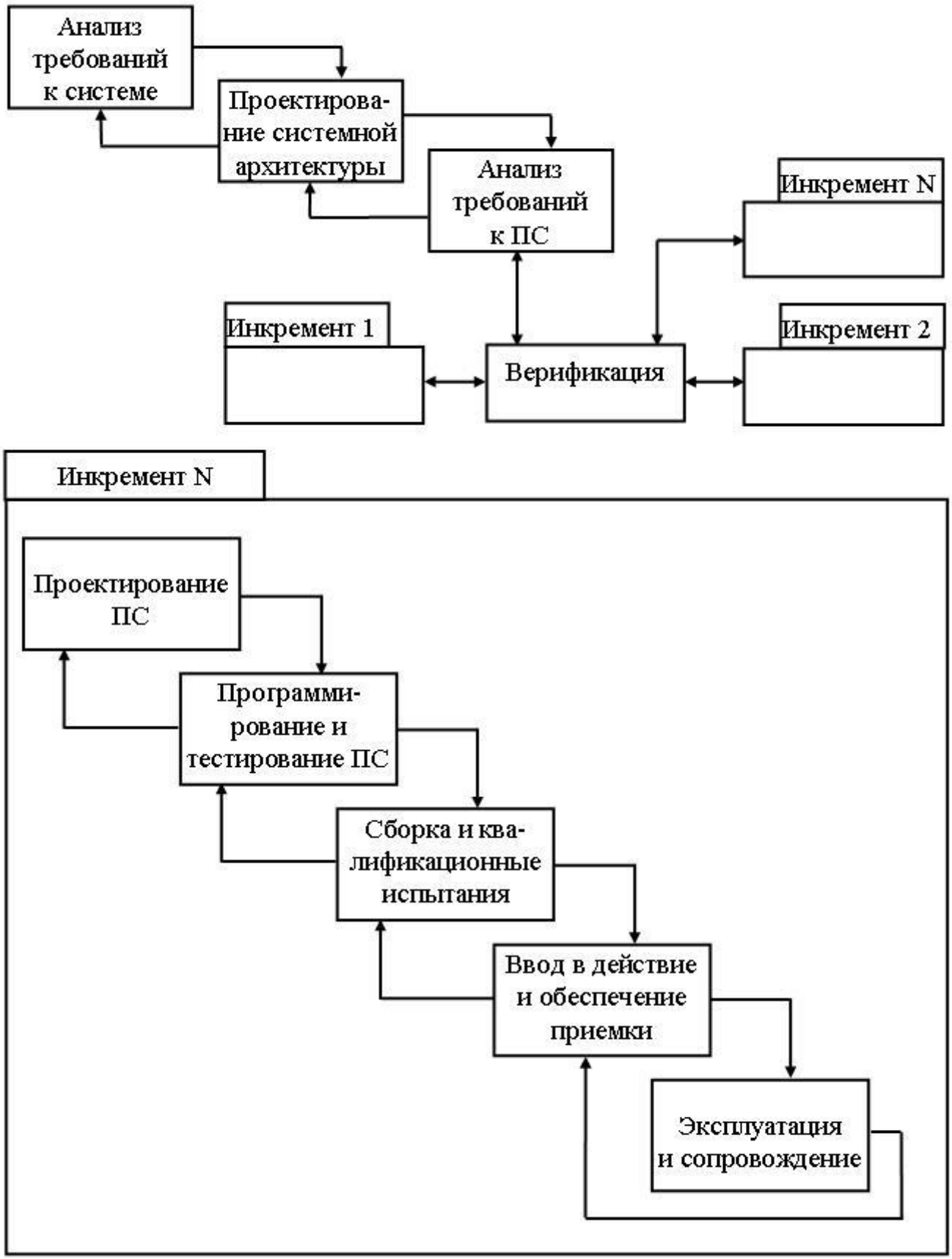

Рис. 2.12. Инкрементная модель жизненного цикла

2.4.3. Вариант инкрементной модели по ГОСТ Р ИСО/МЭК ТО 15271–2002

В стандарте *ГОСТ Р ИСО/МЭК ТО 15271–2002* приводится другой вариант реализации инкрементной модели (рис. 2.13). Он основан на использовании полного заранее сформированного набора требований и их постепенной реализации в отдельных инкрементах. Данный вариант модели учитывает как возможность частично параллельной разработки инкрементов различными группами разработчиков (на рис. 2.13 это возможные информационные потоки между инкрементами 1 и 2), так и возможность последовательной разработки (информационные потоки между инкрементами 2 и N).

В инкременте 1 проектируется архитектура программного средства (или системы) и реализуются его базовые функции. Результаты проектирования инкремента 1 могут быть использованы для проектирования инкремента 2, не дожидаясь окончания реализации инкремента 1.

В дальнейшем различия между инкрементами уменьшаются, что позволяет сократить время их разработки. Поэтому с целью упрощения управления проектом обычно выполняется последовательная разработка инкрементов.

Разработка каждого инкремента состоит из трех укрупненных этапов: проектирование, программирование и тестирование, ввод в действие и обеспечение приемки. Каждый этап объединяет соответствующие работы процесса разработки. Например, при разработке программного средства этап проектирования объединяет работы 5, 6 процесса разработки, этап программирования и тестирования – работы 7, 8, 9, этап ввода в действие и обеспечения приемки – работы 12 и 13 (см. подразд. 1.2).

Инкрементные модели можно комбинировать с другими моделями. Часто их объединяют со спиральной или V-образной моделью. Например, разработка каждого инкремента может выполняться в соответствии с V-образной моделью. Это позволяет снизить затраты и риски при разработке ПС.

Резюме

Рассмотренный вариант инкрементной модели базируется на предварительном полном определении требований и их последовательной планомерной реализации. В модели возможна как последовательная, так и частично параллельная разработка

инкрементов различными группами разработчиков. Возможна комбинация модели с другими моделями. Все это позволяет сократить общие сроки и затраты на разработку ПС и системы в целом.

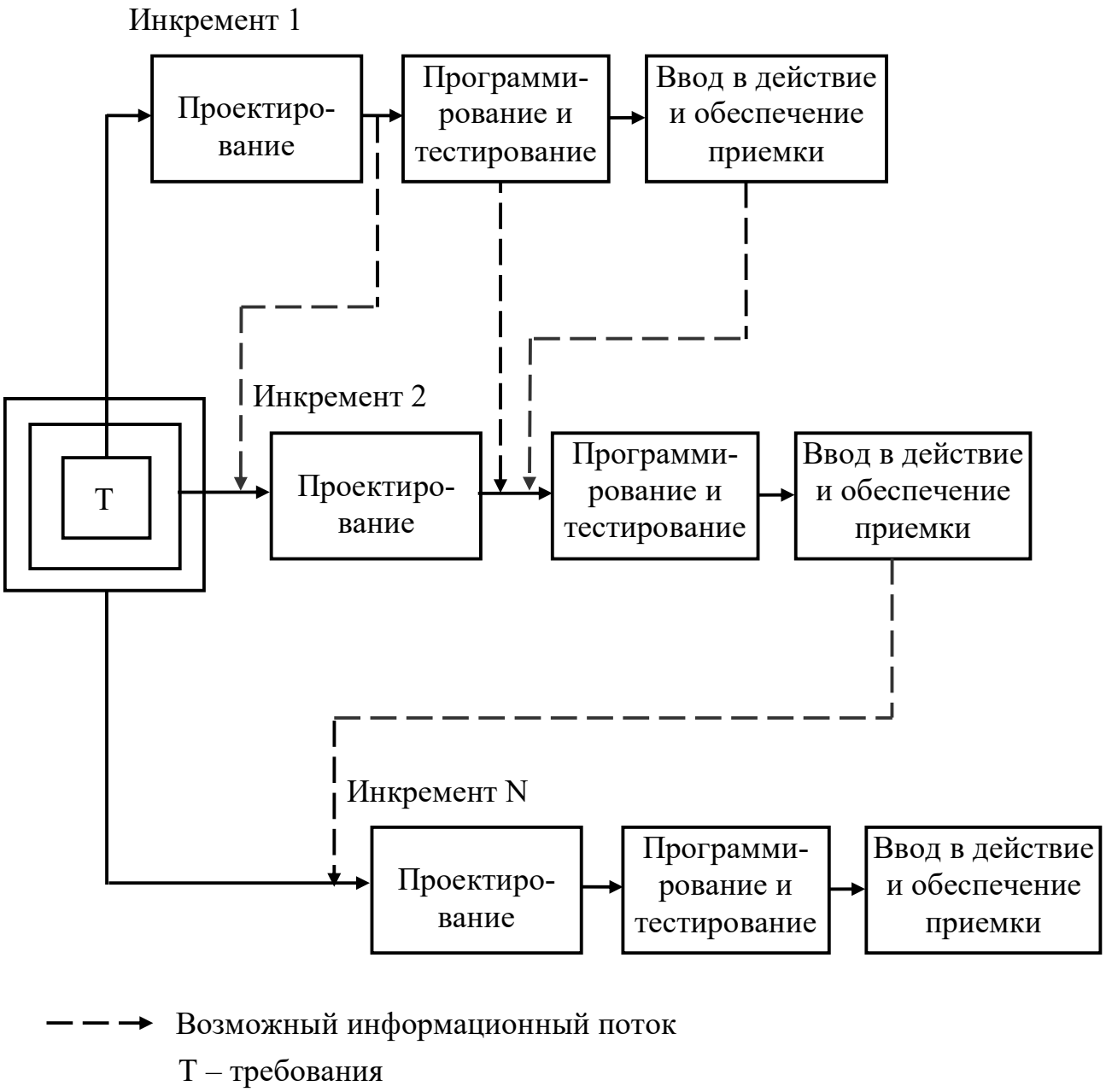

Рис. 2.13. Вариант инкрементной модели по ГОСТ Р ИСО/МЭК ТО 15271–2002

2.4.4. Инкрементная модель экстремального программирования

На рис. 2.14 представлен вариант инкрементной модели жизненного цикла, реализуемой при экстремальном программировании [31]. Данная модель может использоваться в том случае, когда требования заказчика неопределенны или постоянно изменяются. Модель отличается гибкостью, поскольку она ориентирована на высокую степень неопределенности спецификации требований.

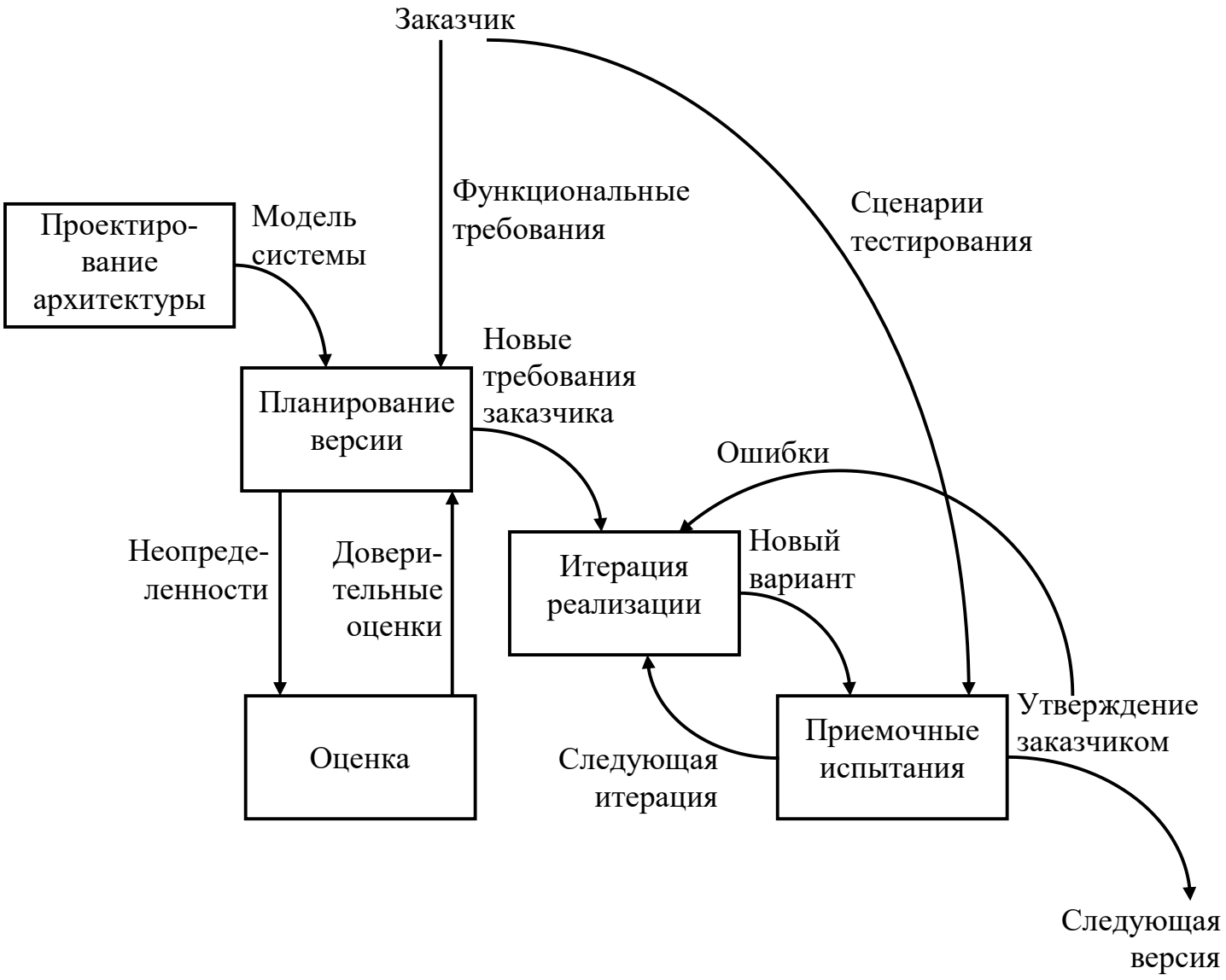

Рис. 2.14. Вариант инкрементной модели экстремального программирования

В соответствии с данной моделью на первом этапе разработки (проектирование архитектуры) усилия разработчиков затрачиваются на тщательное проектирование архитектуры системы и/или программного средства. В дальнейшем уточнение или изменение архитектуры не предусматривается. Результатом данного этапа является модель системы.

На втором этапе разработки выполняется планирование очередной версии системы (программного средства). Каждое новое функциональное требование, поступившее к текущему моменту от заказчика, оценивается с точки зрения стоимости и времени его реализации. При этом учитываются результаты оценки неопределенности данных требований и рисков их реализации. С учетом выполненных оценок заказчик выбирает и утверждает новые требования, которые будут реализовываться в очередной версии системы (программного средства).

Следующие этапы разработки выполняются итерационно. Новые требования заказчика реализуются в новом варианте системы (программного средства), выполняются приемочные испытания данного варианта. Если в очередном варианте системы (программного средства) найдены ошибки, то данный вариант возвращается на следующую итерацию реализации. Процесс продолжается, пока результаты очередных приемочных испытаний не будут утверждены заказчиком. Утвержденный вариант системы (программного средства) поступает в эксплуатацию в качестве очередной версии.

Резюме

Рассмотренный вариант инкрементной модели реализуется при экстремальном программировании. Данный вариант ориентирован на высокую степень неопределенности спецификации требований заказчика. В модели выполняется однократная разработка архитектуры системы и ПС. Реализация каждого требования заказчика выполняется с учетом его стоимости и целесообразности. Каждая версия системы реализуется итерационно.

2.5. Модели жизненного цикла, реализующие эволюционную стратегию разработки программных средств и систем

2.5.1. Общие сведения об эволюционных моделях

Эволюционные модели поддерживают эволюционную стратегию разработки ПС и систем, при которой в начале жизненного цикла определяются не все требования. Система или программное средство строится в виде последовательности версий. Каждая из версий реализует некоторое подмножество требований. После реализации каждой версии требования уточняются (см. п. 2.1.4).

Как правило, эволюционные модели базируются на использовании прототипирования.

2.5.2. Эволюционная модель по ГОСТ Р ИСО/МЭК ТО 15271–2002

Один из простейших классических вариантов эволюционной модели приведен в стандарте *ГОСТ Р ИСО/МЭК ТО 15271–2002* (рис. 2.15).

В данном случае разработка каждой версии системы (программного средства) выполняется на основе каскадной модели, содержащей четыре этапа: разработка требований, проектирование, программирование и тестирование, ввод в действие и поддержка приемки.

При разработке первой версии формулируются наиболее важные (базовые) требования к продукту. На основе данных требований разрабатывается и вводится в действие первая версия системы (программного средства).

При разработке каждой очередной версии требования уточняются, при необходимости вводятся новые или изменяются уже реализованные требования. На основе уточненных требований разрабатывается и вводится в действие очередная версия продукта.

Процесс продолжается, пока все требования не будут окончательно уточнены и полностью реализованы.

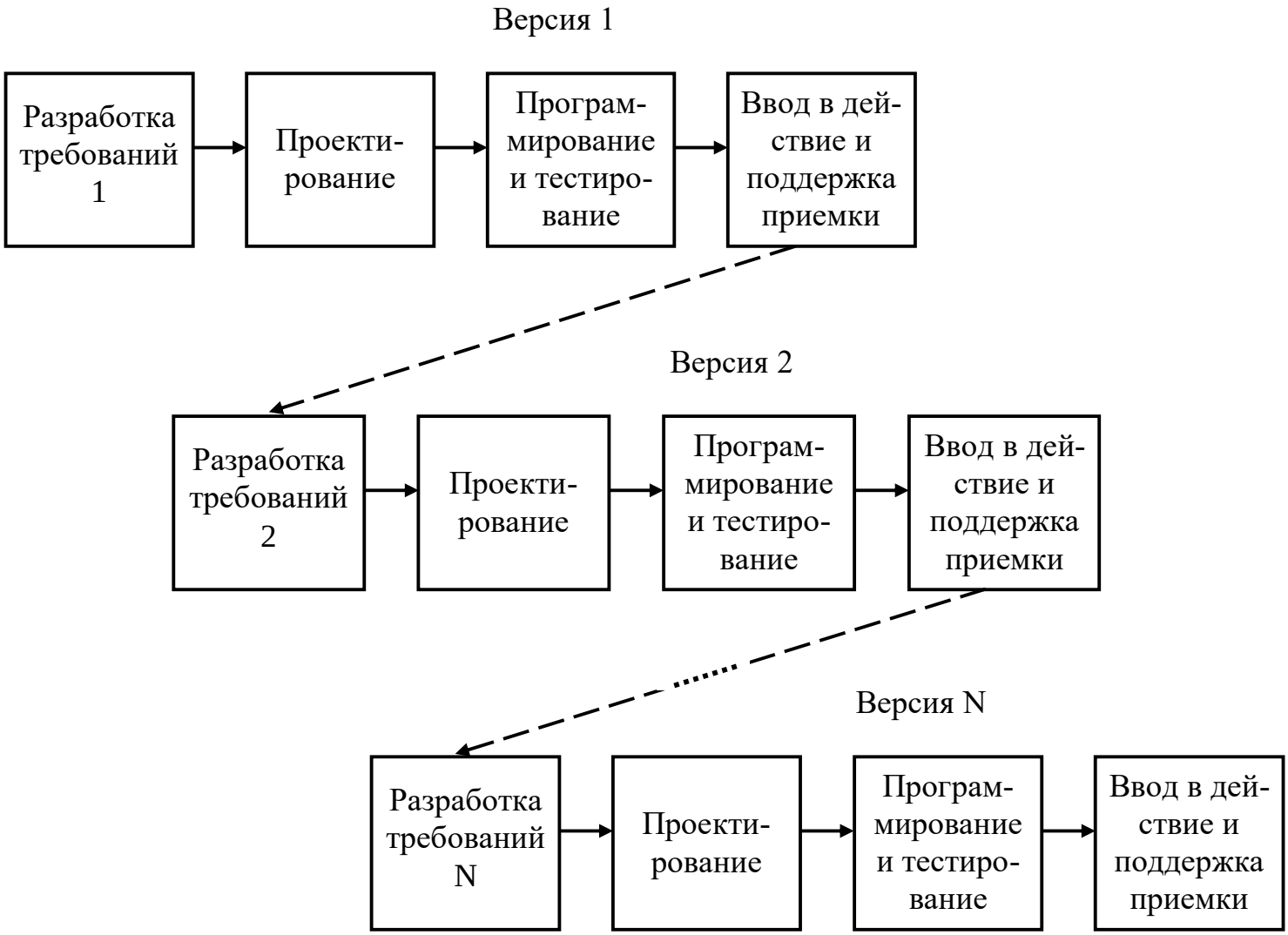

Рис. 2.15. Вариант эволюционной модели по ГОСТ Р ИСО/МЭК ТО 15271–2002

Резюме

Рассмотренный вариант эволюционной модели поддерживает классическую эволюционную стратегию разработки ПС и систем. Для разработки отдельных версий продукта используется каскадная модель.

2.5.3. Структурная эволюционная модель быстрого прототипирования

При использовании структурной эволюционной модели быстрого прототипирования система (программное средство) строится в виде последовательности эволюционных прототипов [27]. Данная модель представлена на рис. 2.16.

Рис. 2.16. Структурная эволюционная модель быстрого прототипирования

Начало ЖЦ разработки находится в центре модели. С учетом предварительных требований пользователями и разработчиками разрабатывается предварительный план проекта.

Затем выполняется быстрый анализ требований к системе (программному средству), во время которого совместно с пользователями разрабатываются *умышленно* неполные требования. На их основе выполняется *укрупненное* проектирование, программирование и тестирование программных компонентов, тестирование системы в

целом. Таким образом, реализуется построение исходного прототипа. Данный прототип оценивается пользователем.

После этого начинается итерационный цикл быстрого прототипирования, содержание которого аналогично циклу построения исходного прототипа (уточнение требований, *укрупненное* проектирование, программирование и тестирование программных компонентов, тестирование системы в целом).

Пользователь оценивает функционирование очередного прототипа. По результатам оценки уточняются требования, на основании которых разрабатывается новый прототип. Этот процесс продолжается до тех пор, пока быстрый прототип не окажется удовлетворительным и не будет принят пользователем.

Затем осуществляется детализированная разработка программного средства, во время которой реализуются его несущественные функции, пользовательские интерфейсы и т.п. После этого выполняется ввод в действие программного средства в среде системы и поддержка его приемки. В результате ускоренный прототип становится полностью действующей системой, которая сдается в эксплуатацию.

Из описания модели быстрого прототипирования видны ее отличия от других эволюционных моделей. Основное отличие заключается в том, что с целью ускорения разработки результаты промежуточных циклов в данной модели представляются в виде прототипов, не доводятся до уровня рабочих версий программного средства или системы и поэтому не сдаются в эксплуатацию. Прототипы предназначены только для уточнения требований.

Зачастую данная модель применяется в комбинации с каскадной моделью. На начальных этапах разработки используется прототипирование, а на последнем (детальная разработка) – этапы каскадной модели с целью обеспечения качества программного средства.

Основным *достоинством* структурной эволюционной модели быстрого прототипирования является ускорение процесса разработки ПС и систем.

К основным *недостаткам* данной модели можно отнести следующее:

1) существует вероятность недостаточного качества результирующего программного средства (и системы в целом) за счет его создания из рабочего прототипа (при детальной разработке программного средства из последнего прототипа может

оказаться сложной или невозможной реализация функций, не реализованных при итерационном прототипировании);

2) возможна задержка реализации конечной версии системы (программного средства) при несочетании языка или среды прототипирования с рабочим языком или средой программирования.

С учетом особенностей и отличий от классической эволюционной стратегии (см. п. 2.1.4) быстрое прототипирование иногда выделяют в отдельную стратегию разработки – так называемую *стратегию быстрого прототипирования*. Существуют различные модели быстрого прототипирования. Общим для них является то, что прототипирование в этих моделях предназначено только для уточнения требований к разрабатываемому продукту.

Резюме

Эволюционная модель быстрого прототипирования используется для ускорения разработки систем или ПС. В данной модели результаты промежуточных циклов представляются в виде прототипов, не доводятся до уровня рабочих версий продукта и не сдаются в эксплуатацию. Прототипы предназначены для уточнения требований. После принятия пользователем (заказчиком) конечного прототипа создается рабочая система или программное средство.

2.5.4. Эволюционная модель прототипирования по ГОСТ Р ИСО/МЭК ТО 15271–2002

В стандарте *ГОСТ Р ИСО/МЭК ТО 15271–2002* приведен пример эволюционной модели, основанной на прототипировании (рис. 2.17). Данная модель адаптирована под требования стандарта *ISO/IEC 12207*:*1995* (*СТБ ИСО/МЭК 12207–2003*, см. подразд. 1.2). Модель предназначена для разработки небольших коммерческих систем.

Основой эффективного применения данной модели жизненного цикла является максимально возможная детализация на ранних этапах процесса разработки (анализ требований к системе и проектирование системной архитектуры). Данные этапы выполняются в модели один раз. Однократное выполнение этих этапов достигается за счет тесных связей разработчиков с пользователями проекта. Требования к системе, в первую очередь, функции системы и внешние интерфейсы определяются пользователями в начале ЖЦ, процессы обработки уточняются при проведении пользователем серии

оценок прототипов системы.

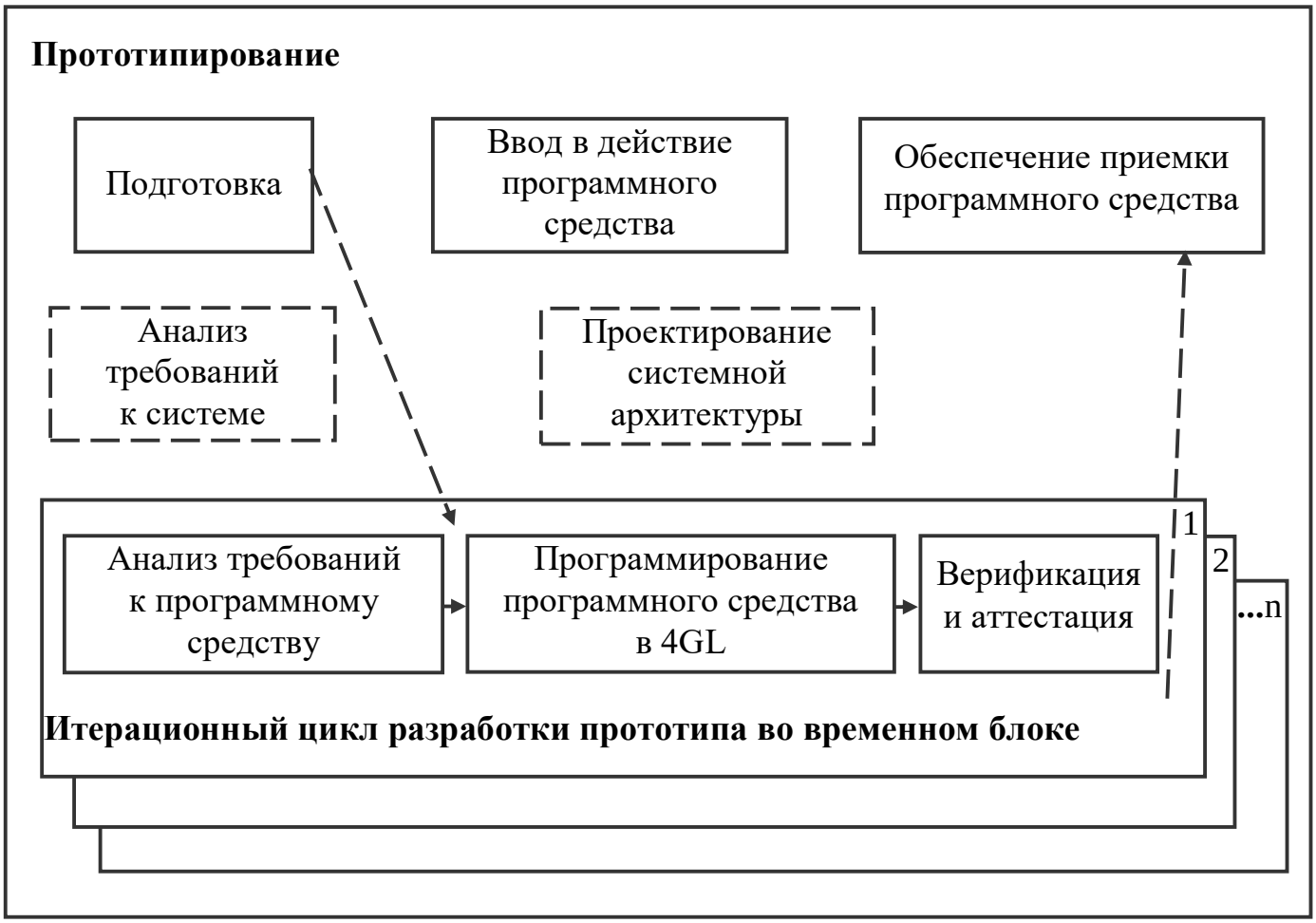

Рис. 2.17. Вариант эволюционной модели прототипирования
по ГОСТ Р ИСО/МЭК ТО 15271–2002

В данной модели при создании версий программного средства используется прототипирование. При разработке каждого прототипа уточняются требования к нему. Затем выполняется программирование прототипа в среде языка программирования четвертого поколения 4GL. При выполнении данного этапа инструментальная среда 4GL используется в первую очередь для быстрого проектирования и сборки программного средства, а также его оперативного наращивания, изменения и уточнения. Языки 4GL осуществляют частичную автоматическую кодогенерацию программного средства.

Следует отметить, что в данной модели при разработке прототипов помимо языков 4GL могут эффективно применяться CASE-средства (см. разд. 6).

В данной модели ЖЦ определен фиксированный период проведения

прототипирования и произвольное количество итераций.

Проверка и оценка каждого прототипа осуществляется пользователем в реальной эксплуатационной среде.

Разработчик программного средства может *контролировать прототипирование* с помощью:

1) установления приоритетов требований к программному средству;

2) ужесточения ограничений временного интервала;

3) привлечения конечного пользователя.

Из описания данной модели видно, что при разработке прототипов фактически используется RAD-модель жизненного цикла, базирующаяся на использовании временного блока (см. подразд. 2.3).

Резюме

В рассмотренном варианте эволюционной модели анализ требований к системе и проектирование системной архитектуры выполняются один раз. Для разработки версий программного средства используется прототипирование. При этом применяется RAD-модель ЖЦ с фиксированным периодом проведения прототипирования.

2.5.5. Спиральная модель Боэма

Спиральные модели ЖЦ поддерживают эволюционную стратегию разработки ПС и систем.

Данные модели объединяют в себе преимущества других видов моделей. Кроме того, в них включен ряд вспомогательных и организационных процессов, предусмотрены анализ и управление рисками, возможности использования прототипирования и быстрой разработки систем и ПС на основе RAD-модели [27].

Базовая концепция спиральной модели заключается в следующем. Каждый цикл разработки (итерация) представляет собой набор операций, соответствующий шагам в каскадной модели. Шагам каскадной модели соответствует и последовательность витков спиральной модели.

Следует отметить, что большинство спиральных моделей было разработано ранее принятия международного стандарта *ISO/IEC 12207*:*1995*. В связи с этим необходимо выполнять адаптацию этих моделей с учетом положений действующих национальных аналогов данного стандарта и его новой редакции *ISO/IEC 12207*:*2008*.

Базовой в семействе спиральных моделей является классическая модель Боэма, разработанная в 1988 г. [27]. На рис. 2.18 изображен вариант классической спиральной модели Боэма, адаптированный с учетом структуры процесса разработки и основных положений стандарта *СТБ ИСО/МЭК 12207–2003*.

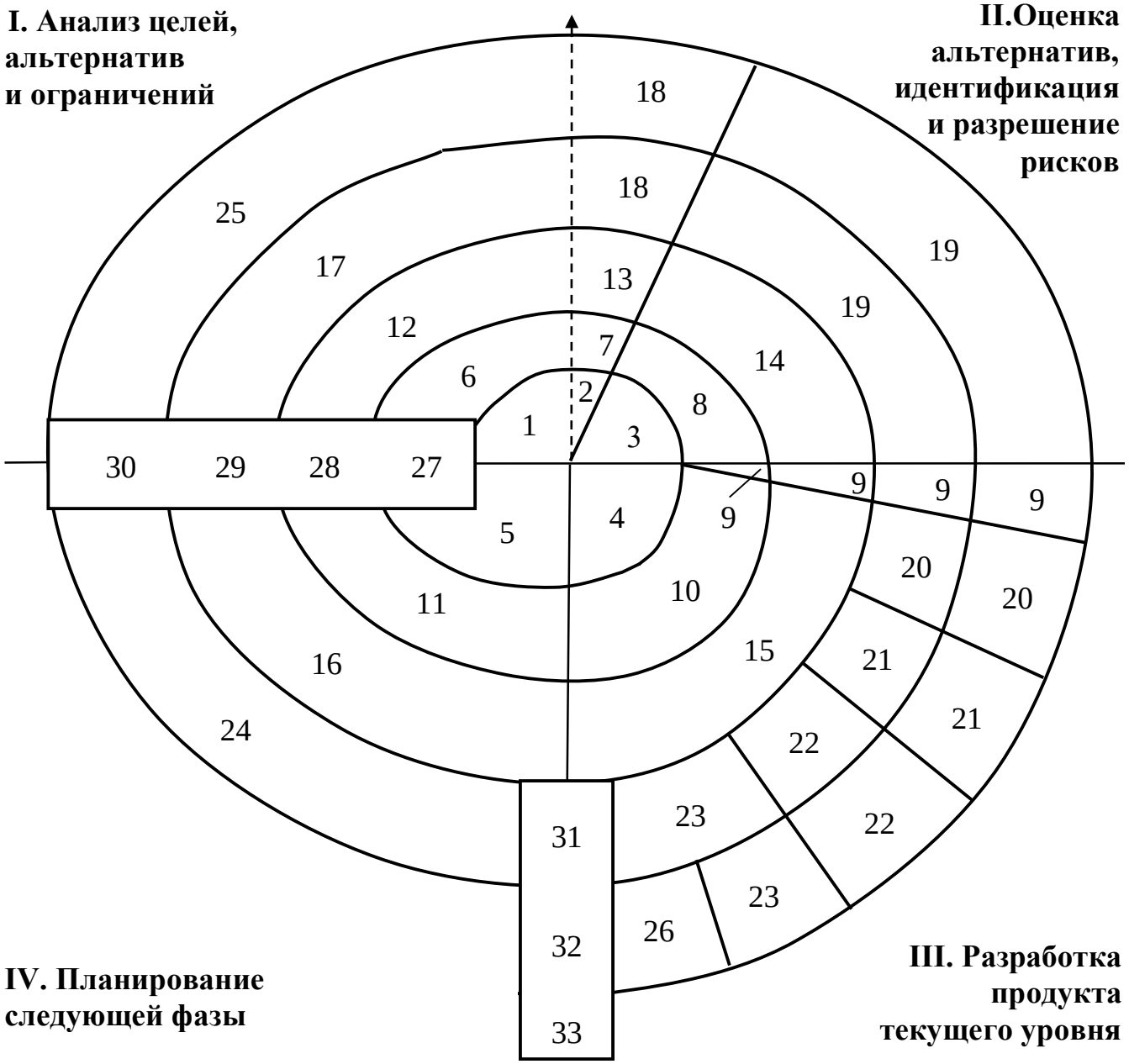

Рис. 2.18. Вариант спиральной модели Боэма,
адаптированный к положениям стандарта *СТБ ИСО/МЭК 12207–2003*

Для удобства описания модели этапы разработки сгруппированы в фазы, соответствующие виткам спирали. Каждая фаза может выполняться итерационно за один

или несколько циклов. На рис. 2.18 использованы следующие обозначения этапов разработки.

A. ***Фаза разработки концепции*** (соответствует первому витку спирали).

В состав данной фазы входят следующие этапы:

1 – определение потребности;

2 – анализ рисков фазы разработки концепции;

3 – концептуальное прототипирование;

4 – разработка концепции требований к системе/программному продукту (концепции эксплуатации);

5 – планирование проекта и процесса разработки.

B. ***Фаза анализа требований*** (соответствует второму витку спирали).

В состав данной фазы входят следующие этапы:

6 – анализ целей, альтернатив и ограничений, связанных с системой/программным продуктом;

7 – анализ рисков фазы анализа требований;

8 – прототипирование требований (называемое демонстрационным прототипированием);

9 – оценка характеристик системы/программного продукта;

10 – разработка требований к системе/программному продукту и их аттестация;

11 – планирование перехода на фазу проектирования системы/программного продукта.

C. ***Фаза проектирования системы/программного продукта*** (соответствует третьему витку спирали).

В состав данной фазы входят следующие этапы:

12 – анализ целей, альтернатив и ограничений, связанных с текущим циклом проектирования;

13 – анализ рисков фазы проектирования;

14 – прототипирование проектирования системы/программного продукта (оценочное прототипирование);

9 – оценка характеристик системы/ программного продукта;

15 – проектирование системной/программной архитектуры, верификация и аттестация результатов проектирования;

16 – планирование перехода на фазу реализации.

D. ***Фаза реализации (технического проектирования, программирования и сборки)***. Соответствует четвертому витку спирали.

В состав данной фазы входят следующие этапы:

17 – анализ возможности реализации в текущем цикле целей, альтернатив и ограничений, связанных с техническим проектированием, программированием и сборкой системы/продукта;

18 – анализ рисков фазы реализации;

19 – прототипирование реализации (операционное прототипирование);

9 – оценка характеристик системы/продукта;

20 – техническое проектирование программного средства;

21 – программирование и тестирование программного средства;

22 – сборка и квалификационные испытания программного средства;

23 – сборка и квалификационные испытания системы;

24 – планирование перехода на фазу расширения функциональных возможностей.

E. ***Фаза сопровождения и расширения функциональных возможностей*** (соответствует пятому витку спирали).

В состав данной фазы входят следующие этапы:

25 – анализ целей, альтернатив и ограничений, связанных с сопровождением и расширением функциональных возможностей;

18 – анализ рисков реализации;

19 – прототипирование реализации (операционное прототипирование);

9 – оценка характеристик системы/программного продукта;

20 – техническое проектирование программного средства;

21 – программирование и тестирование программного средства;

22 – сборка и квалификационные испытания программного средства;

23 – сборка и квалификационные испытания системы;

26 – приемочные испытания.

Выполнение этапов 9, 18 – 23 данной фазы аналогично соответствующим этапам фазы реализации.

Результаты каждого цикла разработки подвергаются соответствующим ***оценочным действиям*** (см. рис. 2.18, левый прямоугольник):

27 – оценка концепции;

28 – оценка требований;

29 – оценка проектирования;

30 – оценка версии системы/продукта.

В нижний прямоугольник модели сведены действия, связанные с ***поставкой результатов*** текущего уровня разработки:

31 – поставка первой пригодной версии;

32 – поставка очередной пригодной версии;

33 – аудит конфигурации версии.

Модель Боэма поделена на *четыре квадранта*. В каждый квадрант модели входят основные и вспомогательные действия по разработке продукта или системы.

В квадранте I – анализ целей, альтернативных вариантов и ограничений – определяются рабочие характеристики, выполняемые функции, стабильность (возможность внесения изменений), аппаратно/программный интерфейс продукта разработки данной фазы или цикла. Формулируются требования к данному продукту.

Определяются альтернативные способы реализации системы или программного продукта (разработка, повторное использование компонент, покупка, договор подряда и т.п.). Определяются ограничения, налагаемые на применение альтернативных вариантов (затраты, график выполнения, интерфейс, ограничения среды и др.).

Определяются риски, связанные с недостатком опыта в данной предметной области, применением новых технологий, жесткими графиками, недостаточно хорошо организованными процессами.

В квадранте II – оценка альтернативных вариантов, идентификация и разрешение рисков – выполняется оценка альтернативных вариантов, рассмотренных в предыдущем квадранте; оценка возможных вариантов сокращения или устранения рисков. Выполняется прототипирование как основа для работ следующего квадранта.

В квадрант III – разработка продукта текущего уровня – включаются действия по непосредственной разработке системы или программного продукта: разработка требований, проектирование системы и ее программных компонентов, разработка и тестирование исходных текстов программ, сборка, тестирование и квалификационные испытания продукта или системы и т.п.

В квадранте IV – планирование следующей фазы – выполняются действия, связанные с решением о переходе на цикл следующей фазы разработки или выполнении еще одного цикла текущей фазы разработки, в частности, оценка заказчиком (пользователем) результатов текущего цикла, уточнение требований, разработка или коррекция планов проекта и следующего цикла, управление конфигурацией.

Работа над проектом в соответствии со спиральной моделью начинается с определения заказчиком потребности в разработке системы или программного продукта (этап 1 квадранта I в центре спирали, см. рис. 2.18).

Первая создаваемая версия системы основывается на предварительных требованиях заказчика (пользователя). Затем начинается планирование следующего цикла, учитывающее требования и пожелания заказчика (пользователя), сформулированные им по результатам работ соответствующего уровня квадранта III. Каждая последующая версия более точно реализует требования заказчика (пользователя). Степень вносимых изменений от одной версии системы (программного продукта) к следующей уменьшается с каждой новой версией. В результате получается окончательный вариант системы/программного продукта.

Для каждого цикла модели анализируются требования, альтернативные варианты и ограничения; определяются, сокращаются или устраняются риски; разрабатывается версия продукта или системы этого цикла спирали и подтверждается ее правильность; планируется следующий цикл и выбираются методы его осуществления. В конце каждого цикла осуществляется оценка его результатов, по итогам которой выполняется либо переход к следующей фазе, либо в случае необходимости выполнение еще одного цикла данной фазы.

Количество итераций модели зависит от сложности проекта. Работы каждой итерации должны бать адаптированы под конкретный проект.

Следует обратить внимание на то, что программирование в спиральной модели выполняется значительно позже, чем в других моделях. Это позволяет минимизировать

риски посредством последовательных уточнений требований, выдвигаемых заказчиком (пользователем). На каждой итерации рассматривается один или несколько главных факторов риска, начиная с фактора наивысшего риска. *Типичные риски* включают в себя неправильно истолкованные требования, неправильно спроектированную архитектуру, потенциальные проблемы эксплуатации продукта или системы, проблемы в технологии и т.д.

При использовании спиральной модели при выполнении соответствующего ей проекта проявляются следующие ее *достоинства*:

1) сокращение и заблаговременное определение непреодолимых рисков, благодаря наличию действий по их анализу;

2) усовершенствование управления процессом разработки, затратами, соблюдением графика и кадровым обеспечением, что достигается путем выполнения анализа в конце каждой итерации.

При использовании спиральной модели применительно к неподходящему ей проекту проявляются следующие ее *недостатки*:

1) усложненность структуры модели, что приводит к сложности ее использования разработчиками, администраторами проекта и заказчиками;

2) необходимость высокопрофессиональных знаний для оценки рисков;

3) высокая стоимость модели за счет стоимости и дополнительных временных затрат на планирование, определение целей, выполнение анализа рисков и прототипирование при прохождении каждого цикла спирали; неоправданно высокая стоимость модели для проектов, имеющих низкую степень риска или небольшие размеры.

Применение спиральной модели Боэма целесообразно при разработке проектов в организации, обладающей навыками адаптации модели с учетом его сложности и критичности.

Резюме

Спиральная модель Боэма является одной из самых сложных моделей ЖЦ. Данная модель состоит из фаз разработки концепции, анализа требований, проектирования системы/продукта, реализации (технического проектирования, программирования и сборки), сопровождения и расширения функциональных возможностей. Модель поделена на четыре квадранта. В каждый квадрант входят основные и вспомогательные

действия по разработке продукта или системы. Особое внимание в модели уделяется анализу и путям разрешения рисков.

2.5.6. Упрощенные варианты спиральной модели

Как было отмечено, основным недостатком модели Боэма является сложность. С учетом этого разработан ряд упрощенных спиральных моделей ЖЦ. Ниже приведены некоторые из данных моделей.

Модель Института качества SQI

Институтом качества SQI предлагается к использованию один из простейших вариантов спиральной модели. Данный вариант представлен на рис. 2.19. В этой модели жизненный цикл проекта разделен на четыре квадранта: «Планирование», «Анализ рисков», «Разработка», «Оценивание заказчиком». В пределах квадрантов выделяются только основные действия различного уровня.

Как и в других спиральных моделях, разработка начинается в центре модели с анализа начальных требований. Анализируются риски, связанные с требованиями, ищутся пути их устранения. На основе результатов разрешения рисков принимается решение о продолжении разработки. Создается и оценивается заказчиком (пользователем) начальный прототип. По результатам оценки уточняются требования к разрабатываемому продукту (системе).

В каждом цикле разработки (витке спирали) выполняются аналогичные действия. При этом в каждом последующем витке спирали разрабатывается более полная версия продукта (системы) до получения полнофункционального продукта с характеристиками, удовлетворяющими заказчика (пользователя).

Таким образом, в данной модели устранена чрезмерная детализация процесса. Необходимый уровень детализации предусматривается при выполнении работ этапов планирования конкретного проекта.

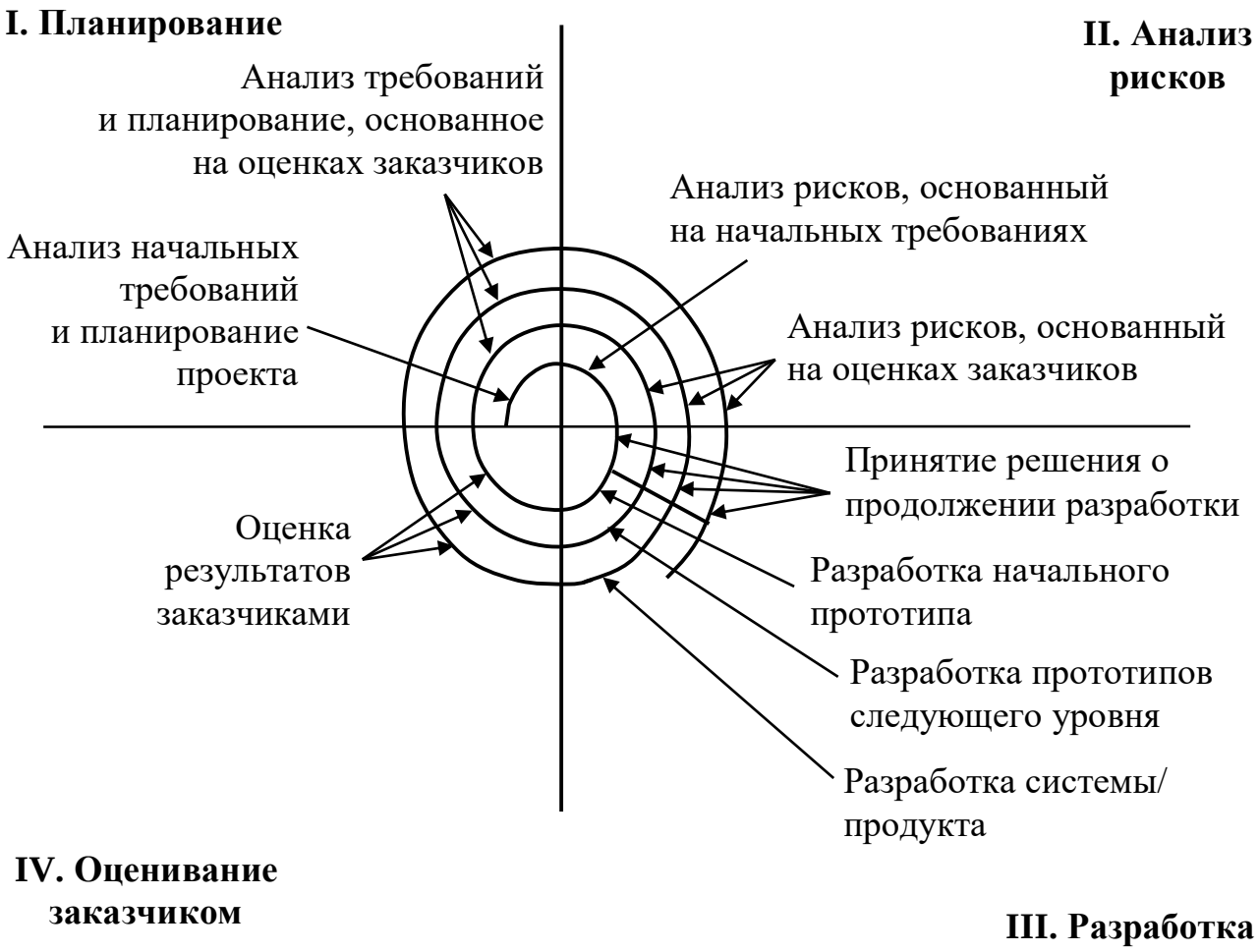

Рис. 2.19. Вариант спиральной модели Института качества SQI

Модель Института Управления проектами PMI

Институтом Управления проектами PMI (Project Management Institute, США) предложен к использованию свой вариант упрощенной спиральной модели. На рис. 2.20 данный вариант представлен в адаптированной к положениям стандарта *СТБ ИСО/МЭК 12207–2003* форме, учитывающей структуру процесса разработки (см. подразд. 1.2).

В рассматриваемом варианте модели процесс разработки проекта разделен на четыре квадранта: «Анализ требований», «Проектирование», «Конструирование», «Оценивание заказчиком».

Модель базируется на применении четырех итерационных циклов:

- цикл доказательства концепции (внутренний виток спирали);
- цикл первой версии (второй виток спирали);

- цикл очередной версии (третий виток спирали);
- цикл конечной версии (внешний виток спирали).

Рис. 2.20. Модель Института Управления проектами PMI,
адаптированная к требованиям стандарта СТБ ИСО/МЭК 12207–2003

В цикле доказательства концепции выполняются следующие этапы работ (см. рис. 2.20):

1 – анализ начальных требований (требований заказчика);

2 – концептуальное проектирование;

3 – конструирование концептуального прототипа;

4 – анализ рисков.

В цикле первой версии выполняются следующие этапы работ:

5 – анализ требований к системе;

6 – проектирование системы;

7 – конструирование первой версии;

8 – квалификационные испытания и оценка.

В цикле очередной версии выполняются следующие этапы работ:

9 – анализ требований к ПС системы;

10 – проектирование ПС;

11 – конструирование очередной версии,

а также этап 8 квалификационных испытаний и оценки.

В цикле конечной версии выполняются следующие этапы работ:

12 – анализ требований к программным модулям;

13 – проектирование программных модулей;

14 – конструирование конечной версии,

а также этап 8 квалификационных испытаний и оценки.

На рис. 2.20 также обозначены:

15 – ввод в действие и обеспечение приемки;

16 – эксплуатация и сопровождение.

Как и в других спиральных моделях действия, выполняемые в каждом витке спирали, осуществляются итерационно за один или несколько циклов разработки.

В квадранте «Анализ требований» выполняются действия, связанные с разработкой требований к результатам очередного цикла модели.

В квадранте «Проектирование» выполняются концептуальное проектирование, проектирование системы, проектирование ПС, программных компонентов и программных модулей.

В квадранте «Конструирование» выполняются кодирование, тестирование и сборка компонентов текущей версии программного средства (или системы) различного уровня.

В квадранте «Оценивание заказчиком» осуществляется оценка рисков проекта (в начале ЖЦ), квалификационные испытания промежуточных версий системы или программного продукта и оценка их результатов заказчиком с целью определения необходимости в переходе к следующему циклу разработки или повторению

предыдущего цикла, квалификационные испытания конечного программного средства или системы в целом.

Модель «win-win»

На рис. 2.21 представлен модифицированный вариант спиральной модели под названием «win-win» (взаимный выигрыш), разработанный Боэмом в 1998 г. [27]. Целью данной модели является обеспечение таких условий выполнения проекта, при которых все его стороны оказываются в выигрыше. С учетом этого модель уделяет повышенное внимание участникам проекта (пользователям, заказчикам, разработчикам, тестировщикам и т.д.). Модель основана на постоянном согласовании всех работ ЖЦ разработки.

Каждый цикл в модели разделен на *шесть этапов*:

I – планирование работ уровня (цикла), обновление всего плана разработки и определение участников работ планируемого уровня;

II – определение условий, необходимых для успешного выполнения работ участниками уровня;

III – согласование условий успешного выполнения работ; анализ целей, ограничений и альтернативных вариантов уровня;

IV – оценка альтернативных вариантов уровня (в отношении как продукта, так и процесса), устранение или сокращение рисков;

V – разработка продукта текущего уровня;

VI – аттестация продукта и процесса текущего уровня; анализ и утверждение результатов уровня.

В каждом цикле разработки, исходя из запланированных работ данного цикла, определяется соответствующий состав исполнителей и оптимальные условия, необходимые для их работы. С учетом существующих ограничений анализируются и оцениваются возможные альтернативные варианты реализации цикла, оцениваются и разрешаются риски, связанные с каждым вариантом.

Исходя из результатов разрешения рисков выполняется разработка продукта текущего уровня, его аттестация и анализ заказчиком (пользователем). По результатам анализа пользователь готовит предложения по уточнению требований к продукту

следующего уровня (цикла).

Рис. 2.21. Спиральная модель «win-win»

На основе данных предложений на следующем цикле выполняется разработка обновленных требований, реализуемых на данном уровне, с учетом чего планируются работы этого уровня.

К *достоинствам спиральной модели «win-win»* по отношению к другим спиральным моделям можно отнести:

1) ускорение разработки продуктов проекта благодаря содействию, оказываемому участникам проекта;

2) уменьшение стоимости продуктов проекта благодаря уменьшению объема переделок и ускорению разработки;

3) более высокий уровень удовлетворенности со стороны участников проекта;

4) как результат – более высокое качество разработанных продуктов.

Спиральная модель Консорциума по вопросам разработки программного обеспечения

Рис. 2.22 иллюстрирует структуру одного цикла спиральной модели, созданной Консорциумом по вопросам разработки программного обеспечения (Software Productivity Consortium) [27].

В данной модели каждый цикл разделен на *пять секторов*:

I – Определение проекта;

II – Анализ рисков;

III – Разработка плана проекта;

IV – Разработка продукта текущего уровня;

V – Управление и планирование.

Результаты выполнения каждого этапа на рис. 2.22 изображены в прямоугольниках. Содержание работ каждого этапа указано в соответствующем секторе спирали.

На данном рисунке приняты следующие обозначения этапов проекта:

1 – Определение планируемого цикла проекта, в том числе:

- определение количества и состава участников работ;
- анализ целей;
- анализ альтернатив;
- анализ ограничений;

2 – Определение рисков;

3 – Оценка рисков;

4 – Планирование разрешения (сокращения и устранения) рисков;

5 – Оценка разрешения рисков;

6 – Оценка альтернатив с учетом результатов оценки разрешения рисков;

7 – Планирование и разработка графиков проекта;

8 – Разработка и верификация продукта;

9 – Мониторинг (надзор) и оценка продукта;

10 – Анализ необходимости изменений версии;

11 – Оценка изменений;

12 – Обновление плана проекта для следующего цикла (витка спирали).

Рис. 2.22. Спиральная модель Консорциума по вопросам разработки программного обеспечения

В каждом цикле проекта на основе целей цикла анализируются альтернативные варианты реализации продукта данного уровня и ограничения, связанные с каждым

вариантом. Оцениваются риски, связанные с альтернативами.

С учетом этого выбирается конкретный альтернативный вариант и создается план разработки. Разрабатывается текущая версия продукта. По результатам ее оценки определяется необходимость изменений версии, уточняются требования к продукту следующего цикла, обновляется план следующего цикла.

Таким образом, в рассматриваемой модели из пяти секторов витка спирали непосредственной разработке продукта посвящен один сектор. Остальные направлены на усовершенствование управления и планирования проекта, оценку продукта, своевременное выявление, ранжирование и разрешение рисков. Это приводит к улучшению условий выполнения проекта, сокращению потерь, связанных с внесением изменений в продукт и как следствие – улучшению качества работ проекта и конечного продукта.

Компонентно-ориентированная спиральная модель

На рис. 2.23 представлен вариант спиральной модели, называемый компонентно-ориентированной моделью [23]. В этой модели основное внимание уделяется процессу разработки. Модель ориентирована на повторное использование существующих программных компонентов.

Программные компоненты, созданные в предыдущих проектах или предыдущих циклах текущего проекта, хранятся в специальной библиотеке. В каждом цикле текущего проекта, исходя из требований, идентифицируются и ищутся в библиотеке кандидаты в компоненты. Они используются в проекте повторно. Если таких кандидатов не выявлено, разрабатываются и включаются в библиотеку новые компоненты.

К *достоинствам* компонентно-ориентированной спиральной модели относятся сокращение длительности и стоимости разработки конечного продукта за счет повторного использования компонентов.

Очевидно, что данную модель можно совместить с другими видами спиральных моделей.

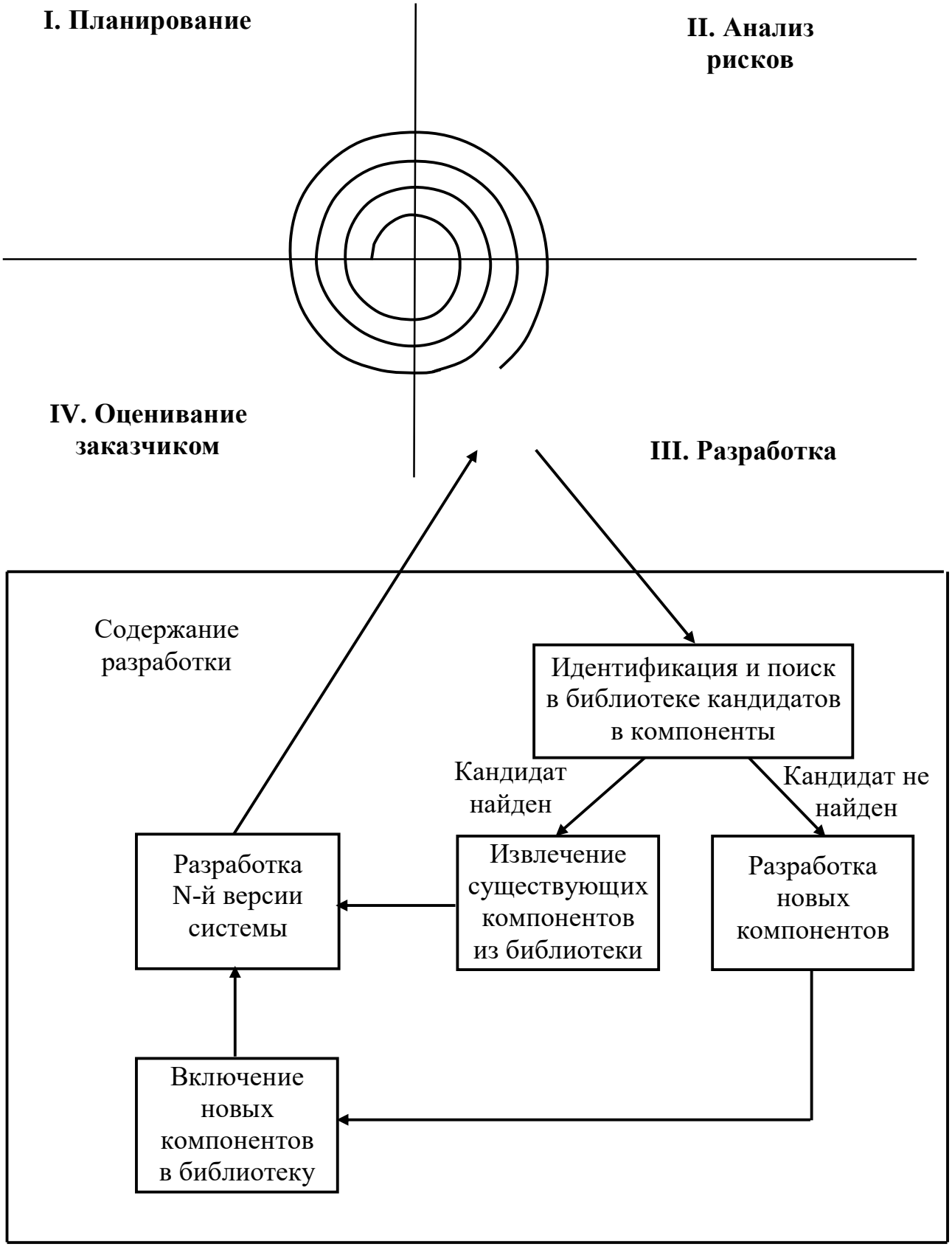

Рис. 2.23. Вариант компонентно-ориентированной спиральной модели

Достоинства упрощенных спиральных моделей

Обобщая написанное выше об упрощенных спиральных моделях, можно выделить общие для них *достоинства*:

1) существенное упрощение по сравнению с базовой спиральной моделью Боэма делает данные модели интуитивно более понятными заказчикам, разработчикам и администраторам проекта, а также уменьшает стоимость проектов;

2) повышенное внимание во всех упрощенных спиральных моделях уделяется действиям, непосредственно не связанным с разработкой (анализ рисков, планирование, работа с персоналом и т.п.). Непосредственная разработка в большинстве моделей занимает не более одного-двух секторов модели. Это обеспечивает более высокое качество процессов и работ проекта и продукта в целом, упрощает прогнозирование сроков и стоимости разработки, повышает удовлетворенность заказчика результатами работ.

Резюме

Упрощенные варианты спиральной модели созданы для снижения сложности базовой спиральной модели Боэма за счет устранения излишней детализации модели. В данных моделях процесс разделен на 4 – 6 секторов, из которых только один-два посвящены непосредственно разработке продукта. Остальные сектора направлены на анализ рисков, планирование, оценки результатов заказчиками, обеспечение условий успешного выполнения работ – то есть действия, влияющие как на качество процессов ЖЦ ПС и систем, так и на качество продуктов разработки.

РАЗДЕЛ 3. ВЫБОР МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ДЛЯ КОНКРЕТНОГО ПРОЕКТА

3.1. Классификация проектов по разработке программных средств и систем

Существуют различные схемы классификации проектов, связанных с разработкой ПС и систем.

В ***ГОСТ Р ИСО/МЭК ТО 12182–2002 – Информационная технология. Классификация программных средств*** – приведена схема классификации ПС по 16 видам, каждый из которых подразделяется на классы. Данная классификация имеет общий характер и в целом не может использоваться для обоснования выбора модели ЖЦ ПС и систем.

Институтом качества программного обеспечения SQI (Software Quality Institute, США) специально для выбора модели ЖЦ предложена схема классификации проектов по разработке ПС и систем [27]. Основу данной классификации составляют *четыре категории критериев*. По каждому критерию проекты подразделяются на *два альтернативных класса*.

Критерии классификации проектов, предложенные Институтом SQI, объединены в следующие категории.

1. Характеристики требований к проекту

Критерии данной категории классифицируют проекты в зависимости от свойств требований пользователя (заказчика) к разрабатываемой системе или программному средству. Например, известны ли требования к началу проекта, сложны ли они, будут ли они изменяться.

2. Характеристики команды разработчиков

Чтобы иметь возможность пользоваться критериями данной категории классификации проектов, состав команды разработчиков необходимо сформировать до

выбора модели ЖЦ. Характеристики команды разработчиков играют важную роль при выборе модели ЖЦ, поскольку разработчики несут основную ответственность за успешную реализацию проекта. В первую очередь следует учитывать квалификацию разработчиков, их знакомство с предметной областью, инструментальными средствами разработки и т.п.

3. Характеристики пользователей (заказчиков)

Чтобы иметь возможность пользоваться критериями данной категории классификации проектов, до выбора модели ЖЦ необходимо определить возможную степень участия пользователей (заказчиков) в процессе разработки и их взаимосвязь с командой разработчиков на протяжении проекта.

Это важно, поскольку отдельные модели ЖЦ требуют усиленного участия пользователей в процессе разработки. Выбор таких моделей при отсутствии реальной возможности пользователей участвовать в проекте приведет к существенному снижению качества результатов разработки.

4. Характеристики типов проектов и рисков

В некоторых моделях в достаточно высокой степени предусмотрено управление рисками. В других моделях управление рисками вообще не предусматривается. Поэтому при выборе модели ЖЦ следует учитывать реальные риски проекта, критичность и сложность продуктов разработки.

Критерии данной категории отражают различные виды рисков, в том числе связанные со сложностью проекта, достаточностью ресурсов для его исполнения, учитывают график проекта и т.д. С учетом этого обеспечивается выбор модели, минимизирующей выявленные риски.

Примеры критериев каждой из категорий приведены в подразд. 3.2.

Следует отметить, что данная классификация проектов, направленная на обоснованный выбор модели ЖЦ, применима для *достаточно масштабных проектов* по разработке ПС и систем. Использование данной классификации и основанной на ней процедуры выбора модели ЖЦ (см. подразд. 3.2) для небольших проектов может привести к чрезмерному увеличению графика проекта, дополнительным затратам и дополнительным рискам.

Таким образом, руководитель проекта должен чётко представлять себе масштаб проекта и необходимость его классификации для выбора модели ЖЦ в соответствии с критериями, предложенными Институтом SQI.

Резюме

Институтом качества программного обеспечения SQI (США) специально для выбора модели ЖЦ разработана схема классификации проектов по разработке ПС и систем. Основу данной классификации составляют четыре категории критериев: характеристики требований к проекту, характеристики команды разработчиков, характеристики пользователей (заказчиков), характеристики типов проектов и рисков. По каждому из критериев проекты подразделяются на два альтернативных класса. Данная классификация предназначена для достаточно крупных проектов.

3.2. Процедура выбора модели жизненного цикла программных средств и систем

Для выбора подходящей к условиям конкретного проекта модели ЖЦ ПС и систем Институтом качества программного обеспечения SQI рекомендуется использовать специальную процедуру [27]. Данная процедура базируется на применении четырех таблиц вопросов. Примеры вопросов приведены в табл. 3.1 – 3.4.

Таблица 3.1

Выбор модели жизненного цикла на основе характеристик требований

№ кри-те-рия	Критерии категории требований	Каскадная	V-образная	RAD	Инкремент-ная	Быстрого прототипиро-вания	Эволюцион-ная
1.	Являются ли требования к проекту легко определимыми и реализуемыми?	Да	Да	Да	Нет	Нет	Нет

№ кри-те-рия	Критерии категории требований	Каскадная	V-образная	RAD	Инкремент-ная	Быстрого прототипиро-вания	Эволюцион-ная
2.	Могут ли требования быть сформулированы в начале ЖЦ?	Да	Да	Да	Да	Нет	Нет
3.	Часто ли будут изменяться требования на протяжении ЖЦ?	Нет	Нет	Нет	Нет	Да	Да
4.	Нужно ли демонстрировать требования с целью их определения?	Нет	Нет	Да	Нет	Да	Да
5.	Требуется ли проверка концепции программного средства или системы?	Нет	Нет	Да	Нет	Да	Да
6.	Будут ли требования изменяться или уточняться с ростом сложности системы (программного	Нет	Нет	Нет	Да	Да	Да

№ кри-те-рия	Критерии категории требований	Каскадная	V-образная	RAD	Инкремент-ная	Быстрого прототипиро-вания	Эволюцион-ная
	средства) в ЖЦ?						
7.	Нужно ли реализовать основные требования на ранних этапах разработки?	Нет	Нет	Да	Да	Да	Да

Таблица 3.2

Выбор модели жизненного цикла на основе характеристик команды разработчиков

№ кри-те-рия	Критерии категории команды разработчиков проекта	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
1.	Являются ли проблемы предметной области проекта новыми для большинства разработчиков?	Нет	Нет	Нет	Нет	Да	Да
2.	Являются ли инструментальные средства, используемые в	Да	Да	Нет	Нет	Нет	Да

№ кри-те-рия	Критерии категории команды разработчиков проекта	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
	проекте, новыми для большинства разработчиков?						
3.	Изменяются ли роли участников проекта на протяжении ЖЦ?	Нет	Нет	Нет	Да	Да	Да
4.	Является ли структура процесса разработки более значимой для разработчиков, чем гибкость?	Да	Да	Нет	Да	Нет	Нет
5.	Важна ли легкость распределения человеческих ресурсов проекта?	Да	Да	Да	Да	Нет	Нет
6.	Приемлет ли команда разработчиков оценки, проверки,	Да	Да	Нет	Да	Да	Да

№ кри-те-рия	Критерии категории команды разработчиков проекта	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
	стадии разработки?						

Таблица 3.3

Выбор модели жизненного цикла на основе характеристик коллектива пользователей

№ кри-те-рия	Критерии категории коллектива пользователей	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
1.	Будет ли присутствие пользователей ограничено в ЖЦ разработки?	Да	Да	Нет	Да	Нет	Да
2.	Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки?	Нет	Нет	Нет	Да	Да	Да
3.	Будут ли пользователи	Нет	Нет	Да	Нет	Да	Нет

№ кри-те-рия	Критерии категории коллектива пользователей	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
	вовлечены во все фазы ЖЦ разработки?						
4.	Будет ли заказчик отслеживать ход выполнения проекта?	Нет	Нет	Нет	Нет	Да	Да

Таблица 3.4

Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

№ кри-те-рия	Критерии категории типов проекта и рисков	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
1.	Разрабатывается ли в проекте продукт нового для организации направления?	Нет	Нет	Нет	Да	Да	Да
2.	Будет ли проект являться расширением существующей системы?	Да	Да	Да	Да	Нет	Нет
3.	Будет ли проект крупно- или	Нет	Нет	Нет	Да	Да	Да

№ кри-те-рия	Критерии категории типов проекта и рисков	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
	среднемасштабным?						
4.	Ожидается ли длительная эксплуатация продукта?	Да	Да	Нет	Да	Нет	Да
5.	Необходим ли высокий уровень надежности продукта проекта?	Нет	Да	Нет	Да	Нет	Да
6.	Предполагается ли эволюция продукта проекта в течение ЖЦ?	Нет	Нет	Нет	Да	Да	Да
7.	Велика ли вероятность изменения системы (продукта) на этапе сопровождения?	Нет	Нет	Нет	Да	Да	Да
8.	Является ли график сжатым?	Нет	Нет	Да	Да	Да	Да
9.	Предполагается ли повторное использование компонентов?	Нет	Нет	Да	Да	Да	Да

№ кри-те-рия	Критерии категории типов проекта и рисков	Каскадная	V-образная	RAD	Инкре-ментная	Быстрого прототипи-рования	Эволюци-онная
10.	Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)?	Нет	Нет	Нет	Нет	Да	Да

Каждая из табл. 3.1 – 3.4 представляет одну из категорий классификации проектов (см. подразд. 3.1). Каждый из вопросов (строка в таблице) предназначен для классификации анализируемого проекта по определенному критерию категории. Столбцы данных таблиц соответствуют обобщенным моделям ЖЦ, фактически представляющим стратегии разработки ПС. При этом под RAD-моделью подразумевается независимая RAD-модель, не встроенная в другие модели жизненного цикла (см. пп. 2.3.1, 2.3.2).

Рассматриваемая процедура состоит из следующей последовательности шагов.

1-й шаг. Проанализировать отличительные черты проекта по критериям категорий, представленным в виде вопросов.

2-й шаг. Ответить на вопросы по анализируемому проекту, отметив слова «да» или «нет» в соответствующих строках табл. 3.1 – 3.4. Если слов «да» или «нет» в строке несколько, необходимо отметить все из них (все «да» или все «нет»).

В качестве примера в табл. 3.1 выделены варианты ответов для проекта разработки сложного и критичного программного средства, требования к которому заранее не известны и будут уточняться по ходу разработки.

3-й шаг. Расположить по степени важности категории (таблицы) и/или критерии, относящиеся к каждой категории (вопросы внутри таблиц), относительно проекта, для которого выбирается модель ЖЦ.

4-й шаг. Выбрать из моделей (см. табл. 3.1 – 3.4) ту модель, которая соответствует столбцу с наибольшим количеством отмеченных ответов с учетом их степени важности

(с наибольшим количеством отмеченных ответов в верхней части приоритетных таблиц). Выбранная модель ЖЦ является наиболее приемлемой для анализируемого проекта.

Для рассмотренного примера на основе результатов заполнения табл. 3.1 наиболее подходящими являются модели быстрого прототипирования и эволюционные модели. Уточнение выбора модели следует производить по результатам анализа табл. 3.1 – 3.4.

Резюме

Процедура выбора модели ЖЦ ПС и систем Института SQI базируется на применении четырех таблиц вопросов, соответствующих предложенной данным институтом классификации проектов. По каждому из вопросов необходимо выделить ответы, соответствующие конкретному проекту, и выбрать модель с наибольшим количеством отмеченных ответов.

3.3. Адаптация модели жизненного цикла разработки программных средств и систем к условиям конкретного проекта

Как отмечалось в подразд. 1.2, основным стандартом в области ЖЦ ПС и систем в Республике Беларусь является стандарт *СТБ ИСО/МЭК 12207–2003*.

В соответствии с данным стандартом выбор модели ЖЦ должен осуществляться при выполнении первой работы процесса разработки (подготовка процесса, см. подразд. 1.2). В соответствии с положениями стандарта, если модель ЖЦ ПС не определена в договоре, то разработчик должен определить или выбрать модель, соответствующую области реализации, величине и сложности проекта.

Выбор подходящей модели ЖЦ – это *первая стадия* применения модели в конкретном проекте. На этой стадии может использоваться процедура выбора модели ЖЦ ПС и систем Института SQI, рассмотренная в подразд. 3.2.

Вторая стадия заключается в адаптации выбранной модели к потребностям данного проекта, к процессу разработки, принятому в данной организации, и к требованиям действующих стандартов.

С учетом этого должны быть выбраны и структурированы в модель ЖЦ ПС работы и задачи процесса разработки из стандарта *СТБ ИСО/МЭК 12207–2003*. Данные работы и

задачи могут пересекаться, взаимодействовать, выполняться итерационно или рекурсивно.

В соответствии с положениями стандартов *СТБ ИСО/МЭК 12207–2003* и *ГОСТ Р ИСО/МЭК ТО 15271–2002* вопросы структурирования в выбранной модели работ и задач процесса разработки должны решаться с учетом следующих ***характеристик проекта***, определенных в данных стандартах:

1) *организационные подходы*, принятые в организации (например, связанные с защитой, безопасностью, конфиденциальностью, управлением рисками, использованием независимого органа по верификации и аттестации, использованием конкретного языка программирования, обеспечением техническими ресурсами); действия, связанные с особенностями реализации данных подходов, должны быть структурированы в модель ЖЦ;

2) *политика заказа* (например типы договора, необходимость использования процесса поставки, использование услуг сторонних разработчиков или субподрядчиков, которых необходимо контролировать);

3) *политика сопровождения программных средств* (например ожидаемые период сопровождения и периодичность внесения изменений, критичность применения, квалификация персонала сопровождения, необходимая для сопровождения среда);

4) *вовлеченные стороны* (например заказчик, поставщик, разработчик, субподрядчик, посредники по верификации и аттестации, персонал сопровождения; численность сторон); большое количество сторон вызывает необходимость структурирования в модель ЖЦ работ, связанных с усиленным административным контролем;

5) *работы жизненного цикла системы* (например подготовка проекта заказчиком, разработка и сопровождение поставщиком); в модель ЖЦ должны быть структурированы работы соответствующих процессов;

6) *характеристики системного уровня* (например количество подсистем и объектов конфигурации, межсистемные и внутрисистемные интерфейсы, интерфейсы пользователя, оценка временных ограничений, наличие реализованных техническими средствами программ); в модели ЖЦ должны быть учтены работы процесса разработки, относящиеся к системному уровню;

7) *характеристики программного уровня* (например количество программных

объектов, типы документов, характеристики качества ПС [12], типы и объемы программных продуктов); выделяются следующие ***типы программных продуктов***:

- новая разработка; при адаптации модели ЖЦ должны учитываться все требования к процессу разработки;
- использование готового программного продукта; при выборе и адаптации модели ЖЦ следует учесть, что на ее первом этапе должна быть выполнена оценка функциональных характеристик, документации, применимости, возможность поддержки готового продукта; процесс разработки может не понадобиться;
- модификация готового программного продукта; при выборе и адаптации модели ЖЦ следует учесть, что на ее первом этапе должна быть выполнена оценка функциональных характеристик, документации, применимости, возможность поддержки готового продукта; процесс разработки реализуется с учетом критичности продукта и величины изменений;
- программный или программно-аппаратный продукт, встроенный или подключенный к системе; в модели ЖЦ необходимо учитывать работы процесса разработки, связанные с системой (работы 2, 3, 10, 11, см. подразд. 1.2);
- отдельно поставляемый программный продукт; не требуется учитывать работы процесса разработки, связанные с системой;
- непоставляемый программный продукт; требования стандарта *СТБ ИСО/МЭК 12207–2003* можно не учитывать;

8) *объем проекта* (в больших проектах, в которые вовлечены десятки или сотни лиц, необходим тщательный административный надзор и контроль с применением процессов совместного анализа, аудита, верификации, аттестации, обеспечения качества; данные процессы следует учесть в модели ЖЦ; для малых проектов такие методы контроля могут быть излишними);

9) *критичность проекта* (значительная зависимость работы системы от правильного функционирования ПС и своевременности выдачи результатов); для таких продуктов характерны повышенные требования к их качеству, поэтому необходим более тщательный надзор и контроль хода выполнения проекта; в модели жизненного цикла следует учитывать процессы верификации, аттестации, обеспечения качества;

10) *технические риски* (например, создание уникального или сложного программного средства, которое трудно сопровождать и использовать, неправильное, неточное или неполное определение требований); в таких случаях в модель ЖЦ следует структурировать действия по разрешению рисков или использовать модели, в которых анализ и разрешение рисков являются их составными частями (например, спиральные модели, см. пп. 2.5.5, 2.5.6);

11) *другие характеристики* (например, усиленный административный контроль за критичными или большими программными продуктами, требующий применения постоянных оценок).

В стандарте *ГОСТ Р ИСО/МЭК ТО 15271–2002* приведен пример модели ЖЦ из со структурированными в нее работами процесса разработки. Данная модель рассмотрена в п. 2.3.4 (см. рис. 2.11).

Резюме

Стандарт *СТБ ИСО/МЭК 12207–2003* определяет, что выбор модели ЖЦ должен осуществляться при выполнении первой работы процесса разработки (подготовка процесса) программного средства или системы. Выбранная модель должна быть адаптирована к потребностям данного проекта, к процессу разработки, принятому в данной организации, и к требованиям действующих стандартов. На адаптацию модели влияют характеристики проекта, определенные в стандартах *СТБ ИСО/МЭК 12207–2003* и *ГОСТ Р ИСО/МЭК ТО 15271–2002*.