Министерство образования и науки Российской Федерации
Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова

**И.В. Шкуропадский**

# Стандартизация, сертификация и управление качеством программного обеспечения

***Учебно-методическое пособие
по изучению курса
и выполнению лабораторных работ***

Новочеркасск
ЮРГПУ (НПИ)
2017

УДК 004.41 (075.8)
ББК 32.973-018
Ш 66

Рецензент – канд. техн. наук, доц. **С.Ф. Сафаров**

**Шкуропадский И.В.**

Ш66 Стандартизация, сертификация и управление качеством программного обеспечения: учебно-методическое пособие по изучению курса и выполнению лабораторных работ / И.В. Шкуропадский; Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова. – Новочеркасск: ЮРГПУ (НПИ), 2017. – 48 с.

Пособие содержит учебно-методические материалы по курсу «Стандартизация, сертификация и управление качеством программного обеспечения»: краткое изложение содержания лекционного курса, описание лабораторного практикума, методические указания к выполнению лабораторных работ, оформлению отчёта и подготовке к его защите, контрольные вопросы, библиографический список использованной и рекомендуемой литературы.

Пособие предназначено для использования в учебном процессе при подготовке бакалавров по направлению 38.03.05 – «Бизнес-информатика».

УДК 004.41 (075.8)
ББК 32.973-018

# СОДЕРЖАНИЕ

*ЧАСТЬ I. КРАТКИЙ КОНСПЕКТ ЛЕКЦИЙ*

# 1. СТАНДАРТИЗАЦИЯ, СЕРТИФИКАЦИЯ И УПРАВЛЕНИЕ КАЧЕСТВОМ В ПРОГРАММНЫХ ПРОЕКТАХ

## 1.1. Стандарт. Задачи стандартизации

Основным законом, определяющим место, цели и задачи стандартизации в России является закон РФ «О техническом регулировании». В законе дано следующее определение стандарта.

«*Стандарт* – это документ, в котором в целях добровольного многократного использования устанавливаются характеристики продукции, правила осуществления и характеристики процессов производства, эксплуатации, хранения, перевозки, реализации и утилизации, выполнения работ или оказания услуг».

Таким образом, стандарт – это специальным образом оформленный договор между всеми участниками определенного вида деятельности, например, производства программного обеспечения. Стандарты позволяют зафиксировать договоренности между участниками, обеспечить совместимость результатов, многократно использовать удачные технические решения. Применение стандартов позволяет использовать заимствованные узлы, детали, программные модули и информационные ресурсы, помогает планировать и управлять ходом разработок.

В законе «О техническом регулировании» перечислены следующие задачи стандартизации:

1. **Обеспечение единства технических решений.** Программные продукты, обрабатывающие или использующие результаты измерений, должны подчиняться стандартам Государственной системы обеспечения единства измерений. При разработке программного обеспечения (ПО) и информационных систем (ИС) разрабатываются терминологические справочники, позво-

ляющие системе и всем ее пользователям одинаково правильно интерпретировать и обрабатывать информацию.

2. **Обеспечение совместимости.** Информационная и программная совместимость – наиболее важный для разработчиков программных продуктов вид совместимости. Большинство стандартов в области разработки программного обеспечения призваны обеспечить информационную и программную совместимость новых и ранее созданных программных продуктов.

3. **Методики выполнения измерений.** В программировании специфическими измерениями являются тесты, подтверждающие работоспособность программ, а также испытания, определяющие значения характеристик программ (например, время реакции на запрос). Разработка и использование методик тестирования и испытаний также является задачей стандартизации.

4. **Использование типовых технических решений.** Удачные технические решения могут быть использованы многократно. Это сокращает затраты на проектирование, а сами изделия становятся более дешевыми и надежными. Разработка универсальных программных модулей и их использование в различных программных продуктах является примером типовых технических решений.

Методы управления, заключающиеся в создании и применении стандартов, называются нормативными методами управления.

## 1.2. Стандарты процесса разработки программного обеспечения

Разработка, производство и использование достаточно сложных объектов требует, чтобы все участники этого процесса договорились по техническим вопросам и придерживались этих договоренностей в своей деятельности. К таким сложным объектам относится, например, программное обеспечение информационных систем.

Договоренности, достигнутые сотрудниками одного предприятия, оформляются в виде стандарта организации (СТО). Требования СТО после утверждения их руководителем организации

становятся обязательными для всех сотрудников данного предприятия.

Для обеспечения взаимодействия предприятий также требуются стандарты. Совокупность требований, отражающих специфику одной отрасли, оформляется в виде отраслевых стандартов (ОСТов). Требования, специфичные для данного региона, оформляются в виде региональных стандартов. Наконец, требования, касающиеся всех граждан и предприятий страны, оформляются в виде государственных стандартов (ГОСТов).

Одним из механизмов присоединения страны к мировому сообществу является признание этой страной международных стандартов. В этом случае в стране выпускается национальный стандарт, полностью идентичный международному стандарту.

Процесс разработки ПО невозможно стандартизировать или систематизировать таким образом, чтобы любая группа разработчиков могла использовать его автоматически, для любого программного продукта и с любых условиях разработки. Каждая организация должна разработать свою собственную модель процесса или приспособить некоторый настраиваемый шаблон процесса под свои нужды. Тем не менее, процесс разработки опирается на модель жизненного цикла ПО, которая может быть стандартизирована и рекомендована в качестве основы для организации работы над программным проектом.

*Жизненный цикл* (ЖЦ) программного обеспечения – непрерывный процесс, который начинается с момента принятия решения о необходимости создания системы и заканчивается в момент её полного изъятия из эксплуатации.

Процесс разработки ПО может в точности следовать положениям одной из основных моделей ЖЦ, использовать вариант одной из моделей или опираться на несколько моделей одновременно, преследуя цель использовать достоинства и компенсировать недостатки каждой из моделей.

Согласно стандарту ГОСТ Р ИСО/МЭК 12207–2010 жизненный цикл программного обеспечения основывается на трёх видах процессов:

1) основные (разработка, эксплуатация и др.);

2) вспомогательные (верификация, оценка, управление конфигурацией, документирование и др.);
3) организационные (создание инфраструктуры и др.).

С другой стороны, в жизненном цикле программного обеспечения выделяют ряд ключевых этапов. Согласно стандарту ГОСТ Р ИСО/МЭК 15288–2005 рассматривают следующие этапы жизненного цикла:

1. *Замысел*. – Замысел новой системы. Оценка реализуемости, предварительное планирование.
2. *Разработка*. – Проект системы, или прототип системы, или конечный программный продукт.
3. *Производство*. – Выпуск продукции. Предоставление продукции пользователям.
4. *Эксплуатация*. – Обеспечение декларированных характеристик системы в процессе её эксплуатации.
5. *Сопровождение*. – Техническое обслуживание и сопровождение, обеспечивающее непрерывное функционирование системы.
6. *Снятие с эксплуатации*. – Прекращение использования системы.

Кроме процессов жизненного цикла могут быть стандартизированы и модели ЖЦ в целом. Так, например, стандарт ГОСТ 34.601–90 определяет структуру процесса разработки автоматизированных ИС, которая по своей структуре и последовательности этапов соответствует каскадной модели ЖЦ ПО.

## 1.3. Сертификация

*Сертификация* – это подтверждение соответствия объекта сертификации предъявленным к нему требованиям.

В системе сертификации можно выделить ряд следующих ключевых элементов.

1. Объект сертификации. – Объект, свойства которого подтверждаются.
2. Цель сертификации. – Определяет, зачем проводится сертификация, как будут использованы ее результаты
3. Требования сертификации. – Устанавливается перечень свойств объекта сертификации, наличие которых под-

твердается в процессе сертификации. Требования могут быть определены законодательно или предъявлены заказчиком.

4. Орган сертификации. – Государственное учреждение или частная фирма, проводящие сертификацию.
5. Схема сертификации. – Указываются правила проведения сертификации, включающие перечень подтверждаемых требований, методику их контроля и подтверждения, а также вид и юридический статус выдаваемого документа.
6. Сертификат. – Документ, выдаваемый органом сертификации, подтверждающий соответствие объекта сертификации предъявленным к нему требованиям.

В России действует национальная система сертификации – ГОСТ Р, образованная на базе Госстандарта России. Она получила признание в большинстве стран мира и включает, в частности, сертификацию программных продуктов. Для получения сертификата системы ГОСТ Р нужно обратиться или в один из аттестованных Госстандартом России органов по сертификации, или непосредственно в Госстандарт, где порекомендуют такой орган. Можно также обратиться в одну из аккредитованных Госстандартом испытательных лабораторий. В отличие от органа по сертификации лаборатория не имеет права самостоятельно выдавать сертификат соответствия, но может проводить испытания, которые затем проверяются и утверждаются органом, связанным с данной лабораторией соответствующими договорами. Каждый орган или лаборатория имеет свою область аккредитации, определяющую номенклатуру программных продуктов, которые они имеют право сертифицировать.

Одним из примеров такой организации является Российский научно-технический центр информации по стандартизации, метрологии и оценке соответствия («Стандартинформ – Сертификат»). Организация аккредитована в качестве Органа по добровольной сертификации программных средств и информационных продуктов вычислительной техники в Системе сертификации ГОСТ Р Федерального агентства по техническому регулированию и метрологии.

# 2. УПРАВЛЕНИЕ КАЧЕСТВОМ В КОНТЕКСТЕ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 2.1. Жизненный цикл программной системы

Практически все современные программные продукты являются сложными системами. Под *системой* в данном случае понимается совокупность взаимодействующих компонентов и взаимосвязей между ними, направленная на достижение определенной цели.

*Программной системой* (ПС) будем считать комплекс связанных между собой программ (программный комплекс), снабженный необходимой документацией, ориентированный на сбор, хранение, поиск, обработку, передачу и отображение информации в некоторой предметной области.

Организация процесса разработки ПС требует комплексного подхода к анализу и учёту многих факторов, влияющих на качество создаваемого продукта. Реализация проекта требует поиска компромисса между имеющимися ресурсами, выделяемым на разработку временем и требуемыми функциональными возможностями. Поэтому процесс разработки системы с самого начала должен быть организован таким образом, чтобы достичь такого компромисса и привести разработку к выпуску качественного программного продукта и вводу системы в эксплуатацию.

Существующие методы анализа, разработки и использования сложных систем основаны на применении понятия *жизненного цикла* (ЖЦ) системы. Такой подход является эффективным при анализе любых сложных объектов и реализации больших проектов.

Жизненный цикл представляет собой некоторую последовательность этапов и совокупность протекающих в течение ЖЦ процессов. Для каждого этапа определяются: состав и последовательность выполняемых работ, получаемые результаты, методы и

средства, необходимые для выполнения работ, роли и ответственность участников и т.д. Такое формальное описание ЖЦ позволяет спланировать и организовать процесс коллективной разработки, внедрения и эксплуатации информационной системы и обеспечить управление этим процессом.

## 2.2. Стандарты этапов и процессов жизненного цикла программных систем

В настоящее время принято несколько международных и национальных стандартов, регламентирующих этапы и процессы ЖЦ ПС. Перечислим наиболее важные из них.

1. ГОСТ 34.601–90. Информационная технология. Автоматизированные системы. Стадии создания.
2. ГОСТ 19.102–77. Единая система программной документации. Стадии разработки.
3. ГОСТ Р ИСО/МЭК 12207–99. Информационная технология. Процессы жизненного цикла программных средств.
4. ГОСТ Р ИСО/МЭК 15288–2005. Системная инженерия – процессы жизненного цикла систем.
5. ISO/IEC 15504 Information Technology – Software Process Assessment. Международный стандарт, определяющий порядок оценки процессов жизненного цикла ПС.
6. ISO/IEC 9000–3–2002. Системная и программная инженерия – руководство по применению стандарта ISO 9001–2000 к программному обеспечению.

Общей особенностью перечисленных стандартов является то, что они разработаны государственными структурами. Эти стандарты рассчитаны на тесное взаимодействие заказчика и разработчика и лучше всего адаптированы к случаю, когда заказчиком разработки ПС является, например, крупная государственная структура или промышленная корпорация.

Перечень работ, которые необходимо выполнить в ходе жизненного цикла ПС, регламентируется в современных стандартах с помощью понятия процесса.

*Процесс* – это совокупность взаимосвязанных действий, преобразующих некоторые входные объекты или данные в выходные.

## 2.3. Стандарт ГОСТ Р ИСО/МЭК 15288–2005

Наиболее полный перечень процессов и составляющих их действий приведен в ГОСТ Р ИСО/МЭК 15288–2005. Согласно этому стандарту все процессы разбиты на четыре группы: процессы соглашения, процессы предприятия, процессы проекта и технические процессы. В отношении каждого из процессов в стандарте определены цели, результаты реализации и действия, входящие в процесс. Всего в стандарте специфицировано 25 процессов, 123 результата реализации и 208 видов работ.

Данный стандарт явился результатом обобщения опыта разработчиков сложных технических систем в оборонно-промышленных комплексах и крупных корпорациях развитых стран и отражает современный взгляд на ЖЦ ПС. В нем сделана попытка найти компромисс – с одной стороны, сделать процесс разработки ПС как можно более управляемым и предсказуемым, а с другой стороны, сделать его достаточно гибким и способным учитывать изменение требований к программной системе в процессе её разработки. С этой целью стандартом предусматривается возможность использования процессов ЖЦ: однократно, многократно, рекурсивно, последовательно и параллельно.

Работы, входящие в процесс, не привязываются жестко к конкретным этапам жизненного цикла и в зависимости от характера разработки и решаемых задач могут инициироваться на любом этапе ЖЦ. Часть процессов и работ при адаптации данного стандарта к конкретному проекту можно исключить из ЖЦ. Стандарт согласован с требованиями стандартов серии ИСО 9000 и группы стандартов ИСО/МЭК 15504, касающихся оценки зрелости процессов проектирования.

## 2.4. Стандарт ГОСТ Р ИСО/МЭК 12207–99

Особенностью стандарта ГОСТ Р ИСО/МЭК 12207–99, который также посвящен ЖЦ ПС, является то, что он не содержит подробного описания этапов работ. Все работы по созданию и сопровождению ИС в указанном стандарте рассматриваются с точки зрения процессов. Перечень процессов ЖЦ, приведенный в данном стандарте, незначительно отличается от перечня, данного в стандарте ГОСТ 15288. Всего в данном стандарте специфици-

ровано 17 процессов, в составе которых выделено 74 вида работ, которые, в свою очередь, разделены на 232 решаемые задачи.

В стандарте достаточно определенно и однозначно изложено, что должны делать, что рекомендуется делать основным участникам ЖЦ – заказчику, поставщику, разработчику. В то же время не указывается, какими методами должны производиться те или иные работы. Для определения методов выполнения работ необходимо пользоваться стандартами и руководствами более низкого уровня. В стандарте предусмотрена возможность его адаптации к условиям конкретного программного проекта. При этом так же, как и в стандарте ГОСТ 15288, часть работ и процессов можно исключить из ЖЦ, используя некоторое подмножество всего множества процессов и работ, приведенных в стандарте. Такая возможность делает данный стандарт достаточно гибким.

Особенность стандарта состоит в том, что заказчику отводится весьма активная роль. Предполагается, что заказчик способен сам сформулировать требования к ПС, которые оформляются в виде заявочных предложений в процессе приобретения. Разработчик подключается к анализу заявочных предложений на этапе поставки, после заключения договора с заказчиком. При этом явно не прописан традиционный для российских организаций этап составления технического задания.

В этом стандарте менее подробно, по сравнению с ГОСТ 15288, описаны некоторые организационные процессы. В частности, явно не упомянуты процессы управления рисками, инвестициями, ресурсами. Во время принятия данного стандарта в России не был еще введен термин *валидация*, поэтому в данном стандарте вместо него использован термин *аттестация*.

Стандарт ГОСТ 12207 не во всем совместим с ГОСТ 15288. Есть различия в терминологии, что затрудняет их совместное использование. В связи с этим разработчику приходится самостоятельно составлять профиль проекта на основе имеющихся стандартов, исходя из специфики решаемых задач, требований заказчика, тенденций на рынке программных продуктов и своих предпочтений.

# 3. СТАНДАРТНЫЕ МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 3.1. Модель жизненного цикла

Понятие *жизненного цикла* находит наиболее важное применение при организации производственного процесса разработки ПС. Принятые международные и национальные стандарты, регламентирующие ЖЦ ПС, разработаны таким образом, чтобы охватить все множество существующих ПС и все известные методологии программирования. Для более детального описания и регламентации процесса разработки ПС, а также определения последовательности выполнения работ служат модели ЖЦ ПС.

*Модель жизненного цикла* программной системы – структура, описывающая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного обеспечения в течение всей жизни ПС, от определения требований до завершения её использования.

В дальнейшем, рассматривая различные модели ЖЦ, для простоты будем использовать следующий набор этапов:

1. Анализ (разработка требований).
2. Проектирование (создание проекта).
3. Реализация (программирование).
4. Тестирование (исправление ошибок).
5. Внедрение (ввод в эксплуатацию).

К настоящему времени наибольшее распространение получили следующие основные модели ЖЦ:

1) каскадная (водопадная) модель и её варианты;

2) инкрементная модель;

3) спиральная модель.

Модель ЖЦ является методологической основой для организации процесса разработки реальной ПС.

## 3.2. Каскадная модель

*Каскадная* или *водопадная* модель ЖЦ является классической моделью однократного прохода, которая описывает линейную последовательность этапов создания программной системы (рис. 3.1).

Каскадная модель ЖЦ предусматривает выполнение стадий жизненного цикла в строго определённом порядке. Переход на следующую стадию осуществляется только после полного завершения работ на предыдущей стадии. Данная модель детально описана в ГОСТ 34.601–90.

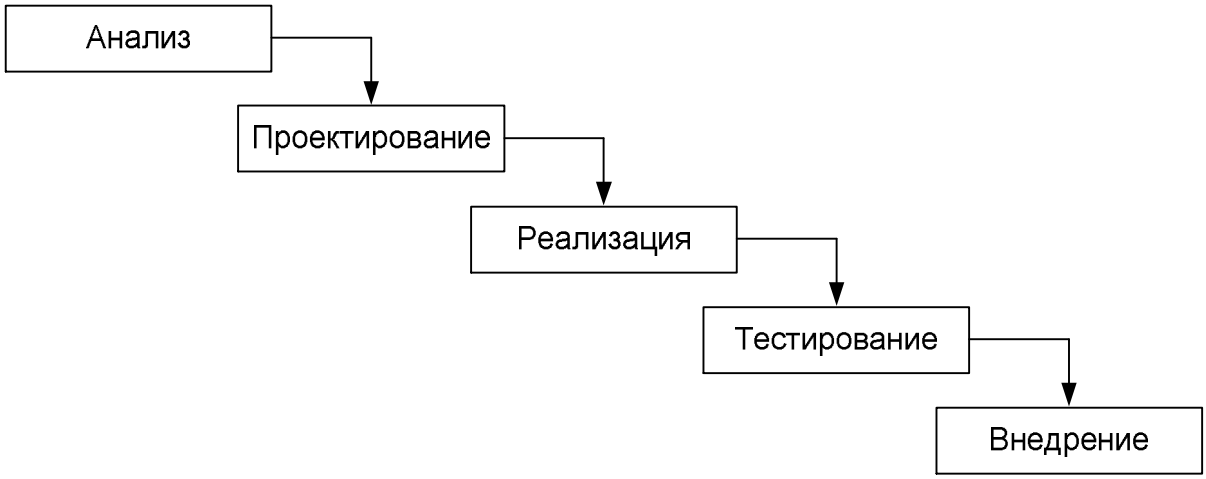


**Рис. 3.1. Каскадная (водопадная) модель жизненного цикла ИС**

Достоинством каскадной модели является явное описание всех этапов работы и определение последовательности их реализации. Это позволяет планировать сроки завершения работ и соответствующие затраты.

Недостатком каскадной модели является то, что реальный процесс создания ПС в действительности практически никогда не укладывается в жёсткую каскадную схему. Постоянно возникает потребность в возврате к предыдущим этапам для уточнения требований и исходных данных.

*Каскадная модель с промежуточным контролем* является модификацией каскадной модели ЖЦ, которая по окончании текущего этапа предусматривает возможность перехода на предыдущий этап для уточнения требований. Межэтапные корректировки позволяют учитывать и сглаживать ошибки результатов

выполнения предыдущих этапов. Этот подход частично снимает недостатки классической каскадной модели.

Общим недостатком всех каскадных моделей ЖЦ является то, что для них требования к ПС зафиксированы в виде формальной спецификации и не могут быть изменены в процессе создания системы. Таким образом, заказчик зачастую получает систему, не соответствующую его ожиданиям.

## 3.3. Инкрементная модель

*Инкрементная модель* ЖЦ отличается от классической каскадной тем, что в ней существует сразу несколько комплектов требований к системе (спецификаций) с разной степенью полноты. Вся разработка делится на заданное количество шагов (итераций, инкрементов). В процессе разработки под каждый набор требований создаётся своя версия программной системы. Таким образом, результатом разработки является не одна, а несколько версий ПС, создаваемых последовательно друг за другом.

При использовании инкрементной модели ЖЦ обычно особо выделяют базовый набор требований к ПС, который определяет функциональные возможности первой версии системы – её прототипа.

*Прототип* – версия ПС, предназначенная для демонстрации заказчику некоторых ключевых свойств будущего продукта. Создание прототипа позволяет вовлечь заказчика в разработку программной системы в самом начале работы.

Результатом выполнения последнего шага является окончательная версия ПС, готовая к вводу в эксплуатацию.

Главным достоинством инкрементной модели ЖЦ является то, что такой жизненный цикл позволяет заказчику контролировать процесс разработки системы, начиная с её самой ранней версии – прототипа.

Недостатком инкрементной модели является то, что, как и для классической каскадной модели ЖЦ, перед началом разработки необходимо сформулировать полный набор требований к программной системе для каждой версии, включая прототип и промежуточные версии.

## 3.4. Спиральная модель

Одной из наиболее эффективных подходов к разработке сложных ПС является использование эволюционной стратегии разработки. В этом случае система строится в виде последовательности версий, причём в начале процесса определены не все требования. В процессе разработки требования уточняются, и система непрерывно дорабатывается.

*Спиральная модель* ЖЦ относится к эволюционным моделям (рис. 3.2). Каждый виток раскручивающейся спирали соответствует разработке одной (начальной, промежуточной или окончательной) версии ПС и представляет собой полный цикл разработки, начиная с анализа и заканчивая внедрением.

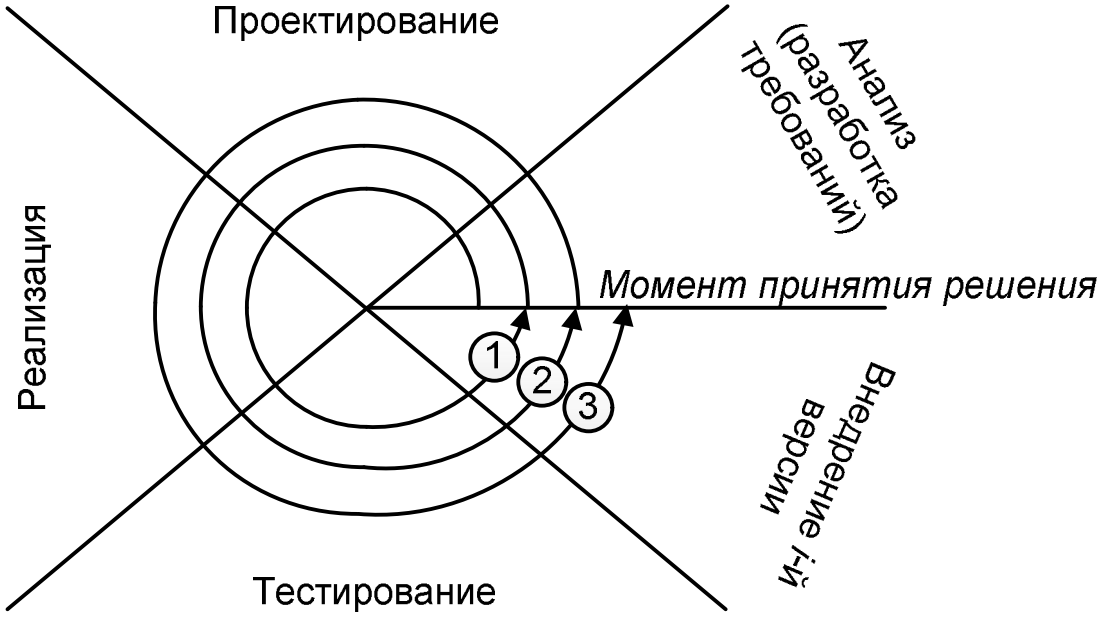


**Рис. 3.2. Спиральная модель («1», «2», «3» – номера версий)**

Спиральная модель отличается от инкрементной модели тем, что первый этап каждой итерации (анализ и разработка требований) выполняется только после завершения предыдущей итерации и выпуска очередной версии системы. Причём этот анализ проводится с учётом полученных результатов и только после согласования этих результатов с заказчиком. Таким образом, нет необходимости заранее выполнять анализ и формулировать требования для всех итераций.

Другим важным отличием спиральной модели ЖЦ является то, что количество требуемых итераций заранее неизвестно. Перед началом каждой итерации выполняется анализ полученных результатов и принимается решение о продолжении или прекращении разработки системы. Если цель достигнута, и разработанная система полностью удовлетворяет потребностям заказчика, то разработка прекращается. Если же возникает необходимость доработки ПС, то процесс разработки переходит на следующий виток спирали.

Достоинством спиральной модели ЖЦ является то, что до реализации доводится обоснованный окончательный вариант ПС, который удовлетворяет действительным требованиям заказчика. Таким образом, снижаются риски, связанные с неправильным пониманием потребностей заказчика или неправильной реализацией требований к системе.

Другим достоинством спиральной модели жизненного цикла является ускорение разработки ПС, обусловленное более активным привлечением заказчика к формированию требований на основе анализа работы промежуточных версий.

Главный недостаток спиральной модели – сложность планирования работ и оценки затрат, сроков и рисков выполнения проекта. Основной проблемой является определение момента перехода на следующую итерацию. Для её решения вводятся ограничения на длительность этапов и итераций по времени.

# 4. УПРАВЛЕНИЕ КАЧЕСТВОМ В МЕТОДОЛОГИИ MICROSOFT SOLUTIONS FRAMEWORK

## 4.1. Модели и дисциплины *MSF*

Методология разработки ПО *Microsoft Solutions Framework* (*MSF*) разработана компанией *Microsoft* на основе своего практического опыта работы над программными проектами. Методология описывает подходы и принципы управления людьми и рабочими процессами для организации процесса разработки программного обеспечения.

Методология *MSF* представляет собой согласованный набор концепций, моделей и правил, определяющих процесс разработки программного обеспечения. Компонентами методологии являются:

1. Базовые принципы *MSF*. – Выражают основные идеи, ценности и стандарты, применимые ко всем элементам методики.
2. Модели *MSF*. – Описывают подходы к организации проектных групп и процессов работы.
3. Дисциплины *MSF*. – Предметные области, которые используют специфический набор методов, терминов и подходов.
4. Практические методики (практики) *MSF*. Например: анализ результатов после контрольной точки, определение и контроль факторов риска и т.д.
5. Рекомендации *MSF*. – Не обязательные, но рекомендуемые практики и руководства, связанные с применением моделей и дисциплин *MSF*.

Пакет руководств по *MSF* структурно состоит из пяти документов, каждый из которых описывает определённую *модель* или *дисциплину*:

1. Модель процессов.

2. Модель проектной группы.
3. Дисциплина управления проектами.
4. Дисциплина управления рисками.
5. Дисциплина управления подготовкой.

Рассмотрим общие характеристики и назначение указанных моделей и дисциплин методологии *MSF*.

**1. Модель процессов** определяет общую методологию разработки и внедрения программного продукта и описывает последовательность действий, осуществляемых в ходе реализации проекта. Процесс *MSF* ориентирован на «вехи» – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Благодаря своей гибкости и отсутствию жестко навязываемых процедур, модель процессов *MSF* может быть применена при реализации весьма широкого круга программных проектов.

**2. Модель проектной группы** определяет шесть ролевых кластеров, их области компетенции, зоны ответственности, цели и задачи. Кластер может быть представлен одним или несколькими сотрудниками, в зависимости от размера проекта, его сложности и профессиональных навыков, требуемых для реализации всех областей компетенции кластера.

**3. Дисциплина управления проектами** определяет принципы и методы организации деятельности проектной группы для достижения целей проектов в рамках согласованных параметров качества, бюджета, сроков и прочих ограничений.

**4. Дисциплина управления рисками** описывает принципы и предлагает рекомендации для выявления и анализа рисков реализации проекта, планирования и реализации стратегий по их профилактике и смягчению возможных последствий, отслеживания состояния рисков, а также извлечения уроков из обретенного опыта.

**5. Дисциплина управления подготовкой** посвящена управлению знаниями, профессиональными умениями и способностями, необходимыми для планирования, создания и сопровождения программных продуктов. Дисциплина управления подготовкой дает рекомендации по применению превентивного подхо-

да к управлению знаниями на протяжении всего жизненного цикла программного продукта.

## 4.2. Модель процессов *MSF*

Модель процессов *MSF* сочетает в себе свойства двух основных моделей жизненного цикла программного обеспечения: каскадной и спиральной.

От спиральной модели жизненного цикла методология *MSF* использует подход, основанный на итеративной разработке. Весь ЖЦ проекта протекает в виде последовательности итераций, каждая из которых заканчивается выпуском версии. Методология *MSF* рекомендует вкладывать в первую версию продукта только базовую функциональность и затем наращивать ее в следующих версиях.

Как и в обычной спиральной модели жизненного цикла в модели процессов *MSF* пересмотр функциональности, планов, спецификаций и требований не прекращается до конца проекта и производится после каждой итерации. Такой подход позволяет планировать последующие итерации, учитывая опыт предыдущих, обеспечивая гибкость и устойчивость к изменению требований заказчика.

Элементы каскадной модели жизненного цикла реализуются в модели процессов *MSF* в виде системы *вех* и *фаз* (рис. 4.1).

*Вехи* – это контрольные точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. В отличие от этапа или стадии, которые описывают характер и объём работ, вехи определяют цели разработки. Методология *MSF* предусматривает следующие ключевые вехи:

1. Концепция проекта утверждена.
2. Планы проекта утверждены.
3. Разработка завершена.
4. Готовность решения утверждена.
5. Внедрение завершено.

Кроме этого может существовать большое количество промежуточных вех, которые показывают достижение в ходе проекта

определенного прогресса и расчленяют большие сегменты работы на меньшие, обозримые участки.

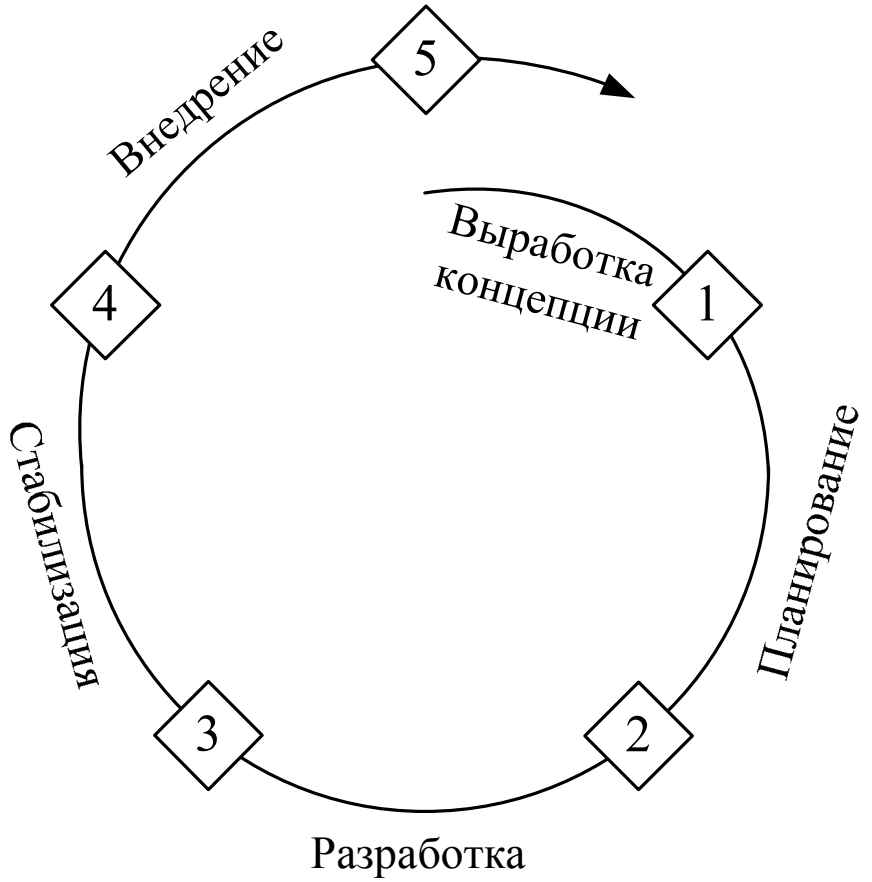


**Рис. 4.1. Модель процессов *MSF* («1»…«5» – вехи)**

*Фазы* – это этапы (стадии) между вехами. Модель процессов *MSF* включает следующие основные фазы процесса разработки:

1. Выработка концепции.
2. Планирование.
3. Разработка.
4. Стабилизация.
5. Внедрение.

Фазы в модели процессов *MSF* по их последовательности, характеру и задачам соответствуют этапам в стандартных каскадной и спиральной моделях ЖЦ. Главной особенностью модели процессов *MSF* является наличие вех – контрольных точек, позволяющих качественно и количественно оценить результат выполнения работ. Таким образом, процесс разработки программного продукта становится более формализованным и управляемым.

# 5. УПРАВЛЕНИЕ КАЧЕСТВОМ В МЕТОДОЛОГИИ RATIONAL UNIFIED PROCESS

## 5.1. Принципы и особенности *RUP*

Методология разработки ПО *Rational Unified Process* (*RUP*) разработана компанией *Rational Software*. Методология описывает, как эффективно применять коммерчески обоснованные и практически опробованные подходы к разработке программных продуктов. Методология *RUP* включает в себя комплекс руководств, примеров, шаблонов и наставлений по использованию инструментальных средств для выполнения всех возможных работ по созданию и сопровождению программного продукта.

В основе методологии *RUP* лежат шесть основных принципов:

1) компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта;
2) работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам;
3) ранняя идентификация и непрерывное устранение возможных рисков;
4) концентрация на выполнении требований заказчиков к исполняемой программе;
5) ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки;
6) постоянное обеспечение качества на всех этапах разработки проекта.

Методология *RUP* основана на трёх ключевых идеях:

1. Весь ход работ направляется итоговыми целями проекта, выраженными в виде *вариантов использования* – сценариев взаимодействия результирующей программной системы с пользователями или другими системами, при выполнении которых пользователи получают значимые для них результаты и услуги.

Разработка начинается с выделения вариантов использования и на каждом шаге контролируется степенью приближения к их реализации.

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

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

Особенностью методологии *RUP* является то, что вместо написания большого количества текстовых документов в течение всего процесса разработки создаются, корректируются и активно используются графические модели, описывающие различные стороны разрабатываемого программного продукта. Эти графические модели формируют общую базу знаний, доступную каждому члену группы разработчиков. В качестве языка моделирования в общей базе знаний используется унифицированный язык моделирования *UML*.

Степень формализации процесса разработки может меняться в зависимости от потребностей проекта. Можно по окончании каждого этапа и каждой итерации создавать все требуемые документы и достигнуть максимального уровня формализации, а можно создавать только необходимые для работы документы. За счет возможности реализации такого подхода к формализации процессов методология *RUP* является достаточно гибкой и широко популярной.

## 5.2. Модель процесса разработки RUP

С точки зрения организации процесса разработки методология *RUP* использует итеративную модель жизненного цикла. Процесс разработки состоит из четырёх фаз, каждая из которых включает в себя одну или несколько итераций (рис. 3.1). Количество итераций, как и в других процессах разработки и моделях ЖЦ, определяется сложностью создаваемой системы.

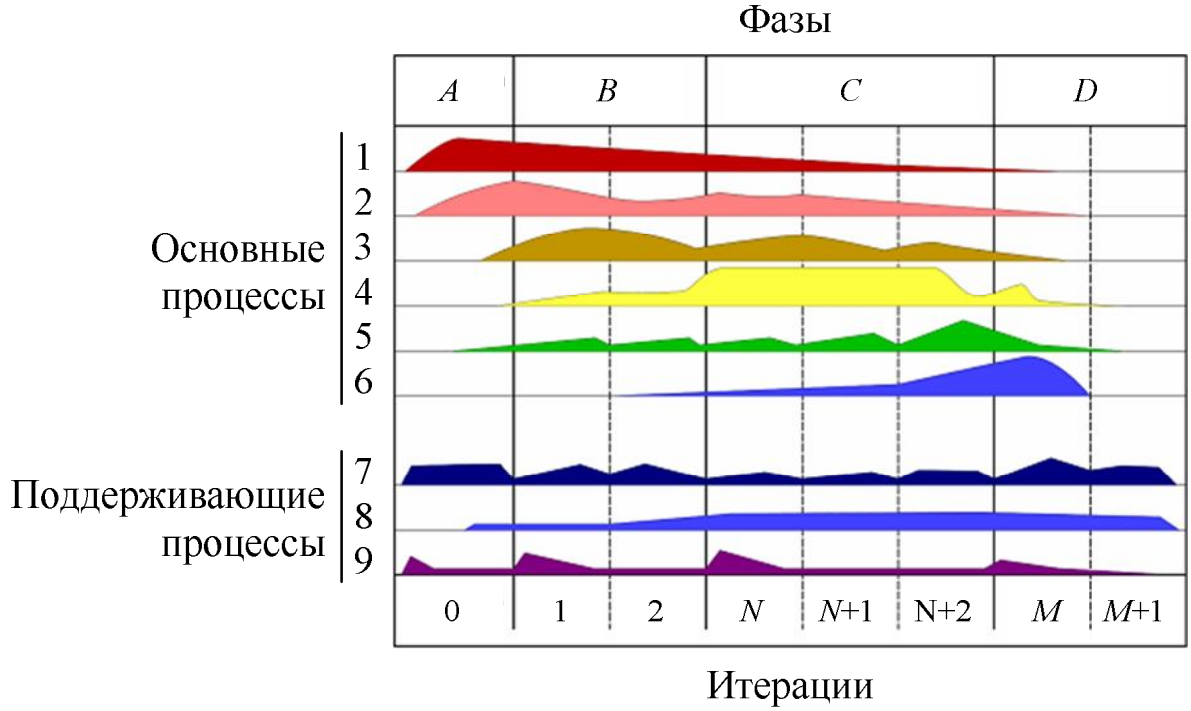


**Рис. 5.1. Модель процесса разработки *RUP***

В отличие от спиральной модели ЖЦ, методология *RUP* использует фазы для группировки отдельных итераций с точки зрения достигаемых результатов:

1. Начальная фаза (рис. 5.1, «*A*»): общее описание системы (основные требования, характеристики и ограничения), план проекта.
2. Фаза уточнения (рис. 5.1, «*B*»): функциональные требования, архитектура системы (модель предметной области, технологическая платформа), проект системы, прототип системы.
3. Фаза конструирования (рис. 5.1, «*C*»): продукт, готовый к внедрению (программное обеспечение, эксплуатационная, техническая и пользовательская документация).

4. Фаза внедрения (рис. 5.1, «*D*»): окончательная версия системы, введённая в эксплуатацию.

Особенностью методологии *RUP* является то, что фазы не имеют жестких ограничений на вид выполняемых работ. Например, фаза уточнения включает в себя не только работы по проектированию системы, но и работы, связанные с программированием, тестированием. Также фаза конструирования не исключает продолжения работ, связанных с построением бизнес-моделей и уточнением требований к системе. В связи с этим важное место в модели процесса *RUP* занимает понятие *дисциплины*.

*Дисциплина RUP* соответствует понятию технологического процесса и представляет собой последовательность действий, приводящую к получению значимого результата.

В рамках *RUP* определены шесть основных дисциплин (технологических процессов):

1) бизнес-моделирование;
2) управление требованиями;
3) анализ и проектирование;
4) реализация;
5) тестирование;
6) развертывание;

и три вспомогательных (поддерживающих):

7) управление конфигурацией и изменениями;
8) управление проектом;
9) создание инфраструктуры.

Главным достоинством методологии *RUP* является удачное сочетание универсального и гибкого подхода к описанию процесса разработки (фазы, итерации, дисциплины) и формализованного подхода к формированию базы знаний (использование графических моделей, построенных с помощью языка *UML*).

# 6. УПРАВЛЕНИЕ КАЧЕСТВОМ В МЕТОДОЛОГИИ RAPID APPLICATION DEVELOPMENT

## 6.1. Гибкие методологии разработки

Гибкие методологии разработки программного обеспечения (*Agile software development*) – это группа методологий, которые ориентированы на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп. К классу гибких методологий разработки относятся, например, экстремальное программирование (*XP*), *Scrum*, *FDD* и др.

Большинство гибких методологий сводят процесс разработки к серии коротких циклов – итераций. Подразумевается, что версия программного продукта готова к выпуску и внедрению в конце каждой итерации, какой бы короткой она не была. Высокая частота выпуска и внедрения готовых версий позволяет разработчикам и заказчику своевременно обнаруживать проблемы постановки и реализации требований и исправлять их уже в следующей версии продукта.

Другим достоинством гибких методологий *Agile* является то, что они делают упор на непосредственное и частое общение всех задействованных в разработке лиц, включая заказчика или его полномочного представителя, определяющего требования к продукту. Это значительно ускоряет процесс выработки и согласования проектных решений.

В то же время характерные особенности гибких методологий на практике могут проявить себя как недостатки, связанные с планированием работ и управлением проектом.

Гибкий подход к выработке и реализации требований к системе не предполагает построения планов разработки на длительный срок. Это обусловлено тем, что заказчик в начале очередной

итерации может сформулировать требования, противоречащие архитектуре текущей версии созданного и поставляемого продукта. В худшем случае это может привести к неоправданно большим затратам времени и средств на переработку архитектуры системы и повторную реализацию её компонентов.

Кроме того, считается, что организация процесса разработки с опорой на гибкие методологии мотивирует разработчиков решать все поступающие задачи наиболее простым и быстрым способом, часто не обращая внимания на качество решения задачи. Это относится как к потребительским качествам продукта (функциональность, производительность, надёжность, удобство использования), так и к качеству программного кода (форматирование кода, наличие комментариев, логическая непротиворечивость, следование принципам структурного и объектно-ориентированного программирования). Накопление и откладывание на неопределённый срок большого количества таких проблем может привести к тому, что в какой-то момент их совместное решение при выпуске очередной версии окажется крайне трудоёмким и дорогостоящим.

Таким образом, гибкие методологии *Agile* представляют собой оригинальный подход к организации процесса разработки программного обеспечения, успешность применения которого на практике сильно зависит от многих факторов и условий. Из приведённого выше описания достоинств и недостатков этого подхода видно, что гибкие методологии успешно применяются, если и разработчик, и заказчик технически и организационно компетентны, доверяют друг другу, ответственно и профессионально подходят к разработке программного продукта. В противном случае риски использования гибких методологий являются слишком высокими, и для организации процесса разработки следует выбрать другой, более формализованный подход.

## 6.2. *CASE*-технологии и *CASE*-средства

Первым шагом в проектировании ИС является получение формального описания предметной области, построение полных и непротиворечивых функциональных и информационных моделей системы. Это логически сложная, трудоемкая и длительная

по времени работа, требующая высокой квалификации участвующих в ней специалистов. Следует также учитывать, что в процессе создания и функционирования ИС потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем. Указанные сложности способствовали появлению специальных технологий (*CASE*-технологий) и программных средств (*CASE*-средств), призванных повысить эффективность разработки программного обеспечения и создания информационных систем.

Термин *CASE* (*Computer Aided Software/System Engineering*) используется в настоящее время в весьма широком смысле. Первоначальное значение термина *CASE*, ограниченное вопросами автоматизации разработки только программного обеспечения, в настоящее время обрело новый смысл, охватывающий процесс разработки сложных информационных систем в целом. Широко используются два тесно связанных друг с другом понятия: *CASE*-технология и *CASE*-средства.

*CASE-технология* представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных информационных систем, которая поддерживается комплексом взаимосвязанных программных средств автоматизации.

*CASE-средства* – это программные средства, поддерживающие процессы создания и сопровождения информационных систем, включая анализ предметной области, формулировку требований, проектирование прикладного программного обеспечения и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление, управление проектом и т.д.

Современные *CASE*-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО. Наиболее трудоемкими этапами создания ИС являются этапы анализа требований к системе и её проектирования, в процессе которых *CASE*-средства обеспечивают качество принимаемых технических решений и подготовку проектной доку-

ментации. При этом большую роль играют методы визуального представления информации. Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую программную систему, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

## 6.3. Методология быстрой разработки приложений *RAD*

Быстрая разработка приложений *RAD* (*Rapid Application Development*) является одной из современных методологий разработки программного обеспечения. Как и другие методологии (*MSF*, *RUP* и др.) *RAD* описывает итеративный подход к организации процесса разработки ПО и соответствующую модель жизненного цикла. Методологию *RAD* также часто связывают с технологией *визуального программирования* и применением современных *интегрированных сред разработки* программного обеспечения.

Одним из основных принципов методологии *RAD* является необходимость применения *CASE*-средств, обеспечивающих целостность проекта на всех этапах его реализации. Все модели и прототипы должны быть получены с применением тех *CASE*-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой опасности. Также *CASE*-средства применяются для автоматической генерации программного кода при помощи автоматических генераторов, получающих информацию непосредственно из репозитория проекта.

Методология *RAD* предполагает, что разработка ПО осуществляется небольшой командой разработчиков за несколько месяцев путём использования инкрементного прототипирования с применением инструментальных средств визуального моделирования и разработки. Для выработки требований и оценки результатов каждой итерации требуется активное привлечение заказчи-

ка к участию в работе на протяжении всего времени разработки программного продукта. Это позволяет в результате обеспечить полное выполнение функциональных и нефункциональных требований заказчика, а также получение качественной документации, обеспечивающей удобство эксплуатации и сопровождения системы.

Методология *RAD* основывается на визуализации процесса создания программного кода приложений и поддерживается инструментальным программным обеспечением, которое предоставляет разработчикам средства *визуального программирования.* Применение средств визуального программирования позволяет значительного ускорить процесс разработки приложений, а также уменьшить трудоёмкость работы по модификации уже готовой программы, внесению в неё необходимых дополнений или изменений.

Методология *RAD* не сводится только к визуальной генерации пользовательского интерфейса. Возможности этой технологии гораздо шире набора процедур, включающих помещение управляющих элементов на экранные формы с последующей установкой их свойств. Эффективность визуального программирования определяется не столько наличием визуальных компонентов, сколько их взаимосвязью и взаимодействием с традиционными средствами разработки приложений. Средства визуального программирования являются неотъемлемой частью современных интегрированных сред разработки программного обеспечения.

Интегрированная среда разработки (ИСР) является средством, с помощью которого выполняются проектирование, программирование, тестирование и отладка прикладных программ. Примерами современных ИСР, поддерживающих методологию *RAD* и технологию визуального программирования, являются *Microsoft Visual Studio*, *Embarcadero RAD Studio*, *IntelliJ IDEA*, *MonoDevelop* и др.

*ЧАСТЬ II. ЛАБОРАТОРНЫЙ ПРАКТИКУМ*

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Описание лабораторного практикума включает в себя учебно-методические материалы к выполнению трёх лабораторных работ по всем темам рабочей программы дисциплины «Стандартизация, сертификация и управление качеством программного обеспечения».

Работа выполняется как во время аудиторных занятий, так и в виде самостоятельной внеаудиторной работы. Выполнение каждой лабораторной работы состоит из трёх этапов:

1. Подготовка и получение допуска к работе.
2. Получение индивидуального задания и выполнение основной части работы.
3. Оформление и защита отчёта о проделанной работе.

В начале каждой лабораторной работы выполняется повторение теоретического материала и проверка готовности к выполнению работы с помощью контрольных вопросов. После получения допуска к выполнению работы выдаётся индивидуальный вариант задания для самостоятельной работы. На заключительном этапе оформляется отчёт о проделанной работе с описанием полученных результатов и выполняется процедура защиты отчёта.

Процедура защиты отчёта заключается в проверке:

1) правильности структуры и оформления отчёта;
2) корректности полученных результатов;
3) способности дать объяснение и необходимое обоснование полученным результатам.

Отчет должен включать в себя:

1. Титульный лист.
2. Задание на лабораторную работу.
3. Содержание отчёта.
4. Описание результатов по каждой части задания.
5. Приложение.

# Лабораторная работа № 1
# ЭТАПЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Цели и задачи лабораторной работы

Целями выполнения лабораторной работы являются:

1. Закрепление знаний о видах и назначении информационных систем. Изучение области применения и функциональных возможностей современных ИС.
2. Приобретение навыков составления документа-обоснования для внедрения ИС.
3. Закрепление имеющихся знаний о моделях жизненного цикла ИС и способах их применения для разработки программного обеспечения.
4. Приобретение навыков анализа требований, условий и ограничений проекта создания ИС и оценки трудоёмкости его реализации.
5. Приобретение навыков составления планов разработки ИС на основе разных моделей жизненного цикла.

В процессе выполнения лабораторной работы решаются следующие задачи:

1. Разрабатывается пример возможного применения одной из информационных систем заданного вида в деятельности некоторого объекта автоматизации (предприятия, организации).
2. Составляется документ-обоснование на внедрение информационной системы.
3. Выполняется анализ постановки задачи. Готовятся исходные данные для планирования. Формулируются ограничения и условия разработки.
4. Разрабатываются прототипы документов: «Техническое задание», «Технический проект», «План тестирования», «План ввода в эксплуатацию».

5. Составляется календарный план разработки информационной системы.

## Контрольные вопросы для допуска к работе

1. Информационные системы.
2. Виды информационных систем, их назначение и состав.
3. Технологии разработки информационных систем.
4. Проектирование информационных систем.
5. Жизненный цикл информационных систем.
6. Этапы жизненного цикла: анализ, проектирование, программирование, тестирование, эксплуатация.
7. Стандартные модели жизненного цикла.
8. Каскадная модель жизненного цикла.
9. Преимущества и недостатки каскадной модели жизненного цикла.
10. Каскадная модель с промежуточным контролем.
11. V-образная каскадная модель.
12. Итеративная модель жизненного цикла.
13. Спиральная модель жизненного цикла.

## Порядок выполнения работы

Вариант индивидуального задания определяет информационную систему, для создания которой необходимо составить план разработки на основе каскадной и спиральной моделей жизненного цикла.

В процессе выполнения лабораторной работы необходимо:

1. Разработать пример возможного применения одной из информационных систем в деятельности некоторого объекта автоматизации (предприятия или организации). Вид деятельности объекта автоматизации выбирается самостоятельно.
2. Составить документ-обоснование для внедрения информационной системы. Описать, чего позволит достичь внедрение информационной системы с точки зрения повышения эффективности работы объекта автоматизации (организации, предприятия).

3. Подготовить исходные данные. Исходными данными для планирования являются:
   3.1. Общее описание некоторой ИС (назначение, область применения, решаемые задачи, технологические особенности реализации и внедрения).
   3.2. Ограничения и условия разработки (требования заказчика, возможности команды разработчиков, сроки разработки, бюджет проекта и т.д.).
4. Составить план разработки ИС с применением каскадного подхода:
   4.1. Составить эскизный план разработки ИС на основе каскадной модели ЖЦ.
   4.2. Для этапа «Анализ требований» составить документ «Техническое задание» с подробным описанием функциональных требований к ИС.
   4.3. Для этапа «Проектирование» составить документ «Технический проект» с описанием проектных решений (архитектура системы, логическая структура базы данных, решения по реализации пользовательского интерфейса и т.д.).
   4.4. Для этапа «Тестирование» составить документ «План тестирования» с описанием методики тестирования и контрольных тестов.
   4.5. Для этапа «Внедрение» составить документ «План ввода ИС в эксплуатацию».
   4.6. Уточнить параметры календарного плана разработки ИС, учитывая ограничения и условия разработки.
   4.7. Объединить календарный план разработки и составленные документы в единый отчёт «Разработка ИС на основе каскадной модели ЖЦ».
5. Составить план разработки ИС с применением итеративного подхода:
   5.1. Разделить весь процесс создания и внедрения ИС на несколько итераций.
   5.2. На основе имеющихся документов (см. пункты 4.2 – 4.5) для каждой итерации составить отдельный комплект документов.

5.3. Составить календарный план итеративной разработки ИС.
5.4. Объединить план итеративной разработки и составленные документы в единый отчёт «Разработка ИС на основе спиральной модели ЖЦ».

# Варианты индивидуальных заданий

## 1. Составление документа-обоснования для внедрения информационной системы:

1. Корпоративные информационные системы (КИС).
2. Системы автоматизации бизнес-процессов (САБП).
3. Геоинформационные системы (ГИС).
4. Системы электронного документооборота (СЭДО).
5. Системы управления корпоративным контентом.
6. Системы планирования ресурсов предприятия.
7. Системы управления взаимоотношениями с клиентами.
8. Системы управления веб-контентом.
9. Интеллектуальные информационные системы.
10. Системы поддержки принятия решений.
11. Информационно-управляющие системы.
12. Информационно-вычислительные системы.
13. Информационно-справочные системы.
14. Обучающие системы.
15. Поисковые системы.
16. Системы автоматизированного проектирования (САПР).

## 2. Составление описания жизненного цикла и плана разработки информационной системы:

1. ИС «Телефонный справочник» (поисковая система).
2. ИС «Библиотека» (информационно-справочная система, поисковая система).
3. ИС «Издательство» (СЭДО, САБП).
4. ИС «Поликлиника» (СЭДО, информационно-справочная система).
5. ИС «Школа» (обучающая система, информационно-справочная система).

6. ИС «Ателье» (САБП).
7. ИС «Склад» (САБП).
8. ИС «Торговля» (САБП, СЭДО).
9. ИС «Автосалон» (САБП, СЭДО).
10. ИС «Продажа подержанных автомобилей» (информационно-справочная система, поисковая система).
11. ИС «Автосервис» (САБП).
12. ИС «Пассажирское автопредприятие» (САБП, СЭДО).
13. ИС «Диспетчерская служба такси» (ГИС, СЭДО).
14. ИС «Агентство по продаже авиабилетов» (информационно-справочная система, поисковая система).
15. ИС «Туристическое агентство» (информационно-справочная система, поисковая система).
16. ИС «Гостиница» (информационно-справочная система, СЭДО).

# Лабораторная работа № 2
# ПРОЦЕСС РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. МЕТОДОЛОГИИ РАЗРАБОТКИ

## Цели и задачи лабораторной работы

Целями выполнения лабораторной работы являются:

1. Закрепление имеющихся знаний о проектах разработки ПО, методах управления программными проектами, стандартах процесса разработки и жизненного цикла ПО.
2. Закрепление имеющихся знаний о современных методологиях разработки программного обеспечения.
3. Приобретение навыков анализа требований, условий и ограничений проекта создания ИС и оценки трудоёмкости его реализации.
4. Приобретение навыков составления планов разработки ИС на основе положений и рекомендаций различных методологий разработки ПО.

В процессе выполнения лабораторной работы решаются следующие задачи:

1. Выполняется анализ постановки задачи. Готовятся исходные данные для планирования. Формулируются ограничения и условия разработки ИС.
2. На основе требований к ИС определяются характеристики программного проекта. Оценивается сложность, масштаб и реализуемость проекта.
3. Разрабатывается документ «Техническое задание», описывающий требования к ИС.
4. Составляется план итеративной разработки ИС на основе положений и рекомендаций методологии *MSF*.
5. Составляется план итеративной разработки ИС на основе положений и рекомендаций методологии *RUP*.

## Контрольные вопросы для допуска к работе

1. Методология *MSF*. Модели и дисциплины *MSF*.
2. Модель процесса *MSF*. Итеративная разработка.
3. Структура модели жизненного цикла *MSF*. Вехи и фазы.
4. Методология *RUP*.
5. Модель процесса разработки *RUP*. Фазы и итерации.
6. Дисциплины *RUP*.

## Порядок выполнения работы

Вариант индивидуального задания определяет ИС, для создания которой необходимо составить план разработки на основе положений и рекомендаций двух методологий разработки программного обеспечения: *MSF* и *RUP*.

В процессе выполнения лабораторной работы необходимо:

1. Подготовить исходные данные для планирования, взяв за основу результаты, полученные при выполнении лабораторной работы № 1:
   1.1. Общее описание некоторой ИС.
   1.2. Ограничения и условия разработки.
2. Составить документ «Техническое задание» с подробным описанием концептуальных и функциональных требований к ИС.
3. Составить план разработки ИС с применением положений и рекомендаций методологии *Microsoft Solutions Framework*:
   3.1. Составить эскизный план разработки ИС на основе модели ЖЦ, описанной в модели процессов *MSF*.
   3.2. Определить примерное количество итераций, необходимое для разработки ИС.
   3.3. Рассматривая последовательно каждую итерацию, сформировать комплект проектной документации, состоящий из документов «План итерации № …» План каждой итерации должен включать в себя следующие разделы:
      3.3.1. для фазы «Выработка концепции» – постановку задачи на разработку соответствующей версии ИС;

3.3.2. для фазы «Планирование» – описание организационных и технических проектных решений по разработке ИС;
3.3.3. для фазы «Разработка» – характеристику ожидаемых результатов разработки очередной версии ИС;
3.3.4. для фазы «Стабилизация» – набор контрольных тестов для валидации и верификации программного обеспечения ИС;
3.3.5. для фазы «Внедрение» – описание мероприятий по переходу пользователей на новую версию ИС.

3.4. Объединить документы, составленные по отдельным итерациям, в единый отчёт «Планирование разработки ИС на основе методологии *MSF*».

4. Составить план разработки ИС с применением положений и рекомендаций методологии *Rational Unified Process*:

4.1. Составить эскизный план разработки ИС на основе модели ЖЦ, описанной в модели процессов *RUP*.
4.2. Определить примерное количество итераций, необходимое для разработки ИС. Распределить итерации по фазам процесса разработки (начальная фаза, фаза уточнения, фаза конструирования, фаза внедрения).
4.3. Рассматривая последовательно каждую фазу, сформировать комплект проектной документации, состоящий из документов «План фазы …» План каждой фазы должен включать в себя следующие разделы:
4.3.1. постановку задачи на разработку соответствующей версии ИС;
4.3.2. описание организационных и технических проектных решений по разработке ИС;
4.3.3. характеристику ожидаемых результатов разработки очередной версии ИС;
4.3.4. набор контрольных тестов для валидации и верификации программного обеспечения ИС;

4.3.5. описание мероприятий по переходу пользователей на новую версию ИС.

4.4. Объединить документы, составленные по отдельным фазам процесса разработки, в единый отчёт «Планирование разработки ИС на основе методологии *RUP*».

## Варианты индивидуальных заданий

В качестве списка вариантов индивидуальных заданий используется перечень информационных систем из лабораторной работы № 1.

# Лабораторная работа № 3
# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ АВТОМАТИЗАЦИИ ПРОЦЕССА РАЗРАБОТКИ. CASE-ТЕХНОЛОГИИ

## Цели и задачи лабораторной работы

Целями выполнения лабораторной работы являются:

1. Закрепление имеющихся знаний о *CASE*-технологиях, применяемых для автоматизации процесса разработки информационных систем.
2. Приобретение навыков выбора средств автоматизации процесса разработки ИС (*CASE*-средств) с учётом принятой модели жизненного цикла и используемой методологии разработки программного обеспечения.
3. Приобретение навыков применения *CASE*-технологии и *CASE*-средств для решения задач, возникающих в процессе создания информационных систем.
4. Закрепление имеющихся знаний о средствах разработки программного обеспечения информационных систем.
5. Приобретение навыков разработки клиентского программного обеспечения ИС с применением принципов методологии *RAD*.

В процессе выполнения лабораторной работы решаются следующие задачи:

1. Формулируются требования к функциональным возможностям *CASE*-средств, выбираемым для автоматизации процесса разработки заданной ИС.
2. Описывается реализация и порядок использования наиболее существенных компонентов *CASE*-технологии: репозитория, средств графического моделирования, технологий взаимодействия между разработчиками, средств

макетирования, прототипирования и автоматической генерации программного кода.

3. Разрабатывается документ, описывающий порядок применения *CASE*-технологии и *CASE*-средств для автоматизации процесса разработки заданной ИС.
4. Проектируется макет интерфейса и разрабатывается прототип клиентского приложения для заданной ИС.
5. Разрабатывается программный код клиентского приложения для реализации функциональных требований к ИС. Выполняется тестирование и отладка разработанного приложения.

## Контрольные вопросы для допуска к работе

1. Автоматизация процессов разработки ИС.
2. Средства автоматизации разработки программного обеспечения.
3. CASE-технология: назначение, состав и ключевые возможности.
4. CASE-средства: назначение и выполняемые функции.
5. Репозиторий. Роль репозитория в автоматизации процессов разработки ИС.
6. Подходы к автоматизации процессов разработки ИС.
7. Структурный подход (информационные, функциональные, структурные модели).
8. Объектно-ориентированный подход.
9. Методология быстрой разработки приложений RAD.
10. Интегрированные среды разработки ПО.
11. Технология визуального программирования.
12. Автоматическая генерация программного кода.
13. Применение RAD и визуального программирования для прототипирования клиентских приложений ИС.

## Порядок выполнения работы

Вариант индивидуального задания определяет информационную систему, процесс разработки которой необходимо автоматизировать с применением *CASE*-технологии и соответствующих программных средств.

В процессе выполнения лабораторной работы необходимо:

1. Сформулировать требования к CASE-технологии и функциональным возможностям CASE-средств, выбираемым для автоматизации процесса разработки ИС.
2. Описать структуру и содержание репозитория, используемого в качестве единой базы данных проекта. Указать способ физической реализации репозитория. Описать средства и методы доступа к объектам репозитория.
3. Описать возможности графического языка, используемого для построения различных моделей разрабатываемой ИС. Перечислить виды диаграмм и описать их назначение.
4. Описать используемые подходы к организации коллективной разработки ИС и управлению командой проекта. Перечислить поддерживаемые виды и способы взаимодействия между членами команды разработчиков.
5. Описать возможности CASE-средств для автоматической генерации программного кода. Описать возможности быстрого макетирования (разработки макетов экранных и печатных форм) и прототипирования (разработки прототипов будущей ИС).
6. Описать возможности современных интегрированных сред разработки программного обеспечения. Описать способы применения ИСР в качестве CASE-средств автоматизации процесса разработки ПО.
7. Разработать документ, описывающий порядок применения CASE-технологии и CASE-средств для автоматизации процесса разработки ИС на всех стадиях жизненного цикла.
8. Выполнить анализ требований к ИС. Составить перечень функциональных требований к клиентскому приложению. Сформулировать общие требования к пользовательскому интерфейсу.
9. Разработать проект пользовательского интерфейса приложения. С помощью интегрированной среды разработ-

ки создать макеты экранных форм с размещёнными на них элементами интерфейса.

10. Разработать прототип клиентского приложения, пользуясь средствами визуального программирования интегрированной среды разработки.
11. Реализовать необходимый функционал приложения добавлением программного кода для обработки системных событий и действий пользователя.
12. Выполнить тестирование общей работоспособности и отдельных функциональных возможностей разработанного приложения. Исправить возможные ошибки.
13. Выполнить верификацию функциональных возможностей разработанного приложения, сравнивая их с имеющимся перечнем функциональных требований.

## Варианты индивидуальных заданий

В качестве списка вариантов индивидуальных заданий используется перечень информационных систем из лабораторной работы № 1.

# КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Стандарт. Виды стандартов.
2. Стандартизация. Сертификация.
3. Управление качеством в программных проектах.
4. Программная система.
5. Стандарты процесса разработки ПО.
6. Жизненный цикл программной системы.
7. Процессы и этапы жизненного цикла.
8. Управление качеством ПС в контексте ЖЦ.
9. Стандарты этапов и процессов ЖЦ ПС.
10. Стандарт ГОСТ Р ИСО/МЭК 15288–2005.
11. Стандарт ГОСТ Р ИСО/МЭК 12207–99.
12. Модель жизненного цикла программной системы.
13. Каскадная модель ЖЦ.
14. Каскадная модель с промежуточным контролем.
15. Инкрементная модель ЖЦ.
16. Спиральная модель ЖЦ.
17. Современные методологии разработки ПО.
18. Методология Microsoft Solutions Framework.
19. Модели и дисциплины *MSF*.
20. Модель процессов *MSF*. Фазы. Вехи.
21. Управление качеством в методологии MSF.
22. Методология *Rational Unified Process*.
23. Процессы и дисциплины *RUP*.
24. Модель процесса разработки *RUP*. Фазы. Итерации.
25. Управление качеством в методологии *RUP*.
26. Гибкие методологии разработки ПО.
27. *CASE*-технологии. *CASE*-средства.
28. Методология *Rapid Application Development*.
29. Интегрированные среды разработки.
30. Визуальное программирование.
31. Управление качеством в методологии *RAD*.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Тебекин А. В. Управление качеством: учебник. – М.: Юрайт, 2013. – 365с.
2. Грекул В. И., Денищенко Г. Н., Коровкина Н. Л. Проектирование информационных систем: курс лекций: учеб. пособие для вузов. – М.: Интернет-ун-т информ. технологий, 2005. – 304 с.
3. Соловьев И. В., Майоров А. А. Проектирование информационных систем: Фундаментальный курс: учеб. пособие для вузов; Моск. гос. ун-т геодезии и картографии. – М.: Академический Проект, 2009. – 398 с.
4. Мещеряков С. В., Иванов В. М. Эффективные технологии создания информационных систем. – СПб.: Политехника, 2005. – 309 с.
5. Миронов М. Г. Управление качеством: учеб. пособие. – М.: Проспект, 2007. – 288 с.
6. Пономарев С. В., Мищенко С. В., Белобрагин В. Я. Управление качеством продукции. Введение в системы менеджмента качества: учеб. пособие для вузов. – М.: Стандарты и качество, 2004. – 248 с.
7. Шубенкова Е. В. Тотальное управление качеством: учеб. пособие для вузов; РЭА им. Г. В. Плеханова. – М.: Изд-во Экзамен, 2005. – 256 с.
8. Баранникова И. В., Ландер А. В. Метрология, стандартизация, сертификация в АСУ: учеб. пособие. – М.: МГГУ, 2008. – 91 с.
9. Сергеев А.Г., Латышев М.В., Терегеря В. В. Метрология, стандартизация, сертификация : учеб. пособие для вузов. – М.: Логос, 2003. – 536 с.

*Учебно-методическое издание*

**Шкуропадский** Иван Владимирович

# СТАНДАРТИЗАЦИЯ, СЕРТИФИКАЦИЯ И УПРАВЛЕНИЕ КАЧЕСТВОМ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ:

**учебно-методическое пособие по изучению курса и выполнению лабораторных работ**

---

Редактор Я.В. Максименко

Подписано в печать 15.06.2017.
Формат 60×84 $^{1}/_{16}$. Бумага офсетная. Печать цифровая.
Усл. печ. л. 2,79. Уч.- изд. л. 3,0. Тираж 50. Заказ № __-____.

---

Южно-Российский государственный политехнический университет
(НПИ) имени М.И. Платова

Отпечатано в ИД «Политехник»
346428, г. Новочеркасск, ул. Просвещения, 132
idp-npi@mail.ru