Министерство сельского хозяйства Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Иркутский Государственный Аграрный Университет им. А.А. Ежевского

Кафедра информатики и математического моделирования

АСАЛХАНОВ П.Г., БЕНДИК Н.В.

Учебное пособие

«МЕТОДОЛОГИИ И ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ»

для студентов
направления подготовки 09.04.03 «Прикладная информатика»

Иркутск, 2020

УДК 004.414.2:004.4’22

Печатается по решению научно-методического совета ФГБОУ ВО Иркутского ГАУ (протокол № от _____2020 г.).

Рецензенты:

Асалханов, П.Г. Методологии и технологии проектирования информационных систем / Учебное пособие для студентов направления «Прикладная информатика» // П.Г. Асалханов, Н.В. Бендик, – Иркутск: Изд-во Иркутский ГАУ, 2020. – 128 с. – ил.

Учебное пособие содержит материал по курсу "Методологии и технологии проектирования информационных систем" и представляет собой руководство по созданию информационных систем с использованием различных подходов, стандартов, методологий, нотаций и технологий. Пособие предназначено для студентов, изучающих основы системного анализа и проектирования информационных систем.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Все нужно проектировать сверху вниз, за исключением фундамента, с которого нужно начинать.
Алан Джей Перлис

Темпы развития и динамика современного мира находят свое отражение не только в различных сферах деятельности, применяемых технологиях, методах производства, но и предъявляют к ним определенные требования. Управление любой деятельностью, ее поддержка, сегодня неразрывно связаны с информационными системами (ИС), которые специалисты пытаются создавать с учетом возможных изменений в различных сферах.

В данном пособии проводится анализ имеющихся на сегодняшний день подходов проектирования ИС относительно трех аспектов, которые являются наиболее чувствительными к различного рода изменениям, и поэтому их сложнее всего учитывать, когда речь идет о создании ИС. Такими аспектами являются: предметная область, структура данных и возможность интеграции.

Все существующие подходы проектирования в своей основе имеют идеологию, базирующуюся на приоритетах и потребностях общества. В современном мире такими приоритетами являются: многократное, повторное использование знаний; создание средств всем миром; переход на более высокий уровень абстрагирования – переход от объектов и вещей к понятиям и характеристикам; использование инструментальных средств, с помощью которых сами пользователи должны создавать информационные системы; создание инструментальных средств самими пользователями, с помощью которых возможно создавать информационные системы; необходимость создавать инструментальные средства, с помощью которых требования пользователя преобразуются в готовую ИС. Совокупность этих приоритетов

и будем называть современной идеологией создания информационных систем [40].

Учебное пособие является компиляцией популярных материалов, раскрывающих практические аспекты применения CASE-средств визуального проектирования с использованием структурного и объектно-ориентированного подходов. В первой главе приводятся теоретические положения, связанные с нормативно-технической документацией на разработку и проектирование ИС, управление жизненным циклом ИС, внедрением и сопровождением ИС.

В второй главе учебного пособия рассматриваются методологии современного проектирования информационных систем, такие как: SADT, RAD, ARIS, BPMN, Scrum, DSDM, extreme Programming, FDD и UML.

Третья глава посвящена вопросам организации и технологии проектирования информационных систем. Проанализированы следующие виды проектирования: каноническое и типовое. Помимо этого, рассмотрены технологии Rational Unified Process и OpenUP.

1 ОСНОВНЫЕ ПОНЯТИЯ ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

1.1 Основные понятия и определения

Основные понятия в последние годы не претерпели сильных изменений, формулировки стали более точными и лаконичными, исключающими неоднозначность понятий. Наиболее полные определения представлены в Федеральных законах Российской Федерации и стандартах [44].

Информация – «сведения (сообщения, данные) независимо от формы их представления» [7].

Информационные технологии – «процессы, методы поиска, сбора, хранения, обработки, предоставления, распространения информации и способы осуществления таких процессов и методов» [7].

Информационная система – «совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств» [7].

Проектирование информационных систем – это упорядоченная совокупность методологий и средств создания или модернизации информационных систем [42].

Управление информационными системами – «применение методов управления процессами планирования, анализа, дизайна, создания, внедрения и эксплуатации информационной системы организации для достижения ее целей» [1].

Жизненный цикл информационных системы – «развитие рассматриваемой системы во времени, начиная от замысла и кончая списанием» [8].

Модель жизненного цикла – «структурная основа процессов и действий, относящиеся к жизненному циклу, которая служит в качестве общей ссылки для установления связей и взаимопонимания сторон» [8].

Архитектура информационных систем – это концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.

Бизнес-процесс – это цепочка взаимосвязанных действий, направленных на создание товарной продукции или услуги.

Регламент бизнес-процесса – это четко определенный порядок выполнения бизнес-процесса, определяющий состав и действия участников.

Модель данных – это система организации данных и управления ими.

Методология проектирования информационных систем – это совокупность принципов проектирования (моделирования), выраженная в определенной концепции.

Технология проектирования – это совокупность концептуальных методов и средств (методологий) проектирования ИС, а также методов и средств организации проектирования, то есть управления процессом создания или модернизации проекта информационной системы.

Средства моделирования – это программы описания и моделирования систем.

Типовое проектное решение (ТПР) – это многократно используемое проектное решение.

Нотации – это определенные способы представления элементов информационной системы.

Реинжиниринг бизнес-процессов – это фундаментальная реорганизация бизнес-процессов с целью повышения их эффективности.

Системный подход – процесс рассмотрения любой системы в качестве совокупности взаимосвязанных элементов.

Процессный подход – представление любой системы в качестве совокупности процессов.

Функциональный подход – предусматривает четкое закрепление за каждой структурной единицей набора функций.

Техническое задание – документ, используемый заказчиком в качестве средства для описания и определения задач, выполняемых при реализации договора [9].

1.2 Исторические аспекты развития технологий проектирования информационных систем

Середина прошлого столетия ознаменовалась началом активного развития информационных технологий. Прежде всего, военные ведомства и передовые предприятия многих стран понимают важность и ценность создания и развития информационных систем. С появлением вычислительной техники обработка больших объемов информации и автоматизация основных производственных процессов и органов управления на всех уровнях становятся наиважнейшей задачей для обеспечения военного превосходства наиболее развитых государств и конкурентных преимуществ коммерческих компаний. Разработчики национальных и крупномасштабных информационных систем стали первыми осознавать необходимость создания специальных средств проектирования и моделирования бизнес-процессов, позволяющими сделать их работу более эффективной и сократить не только сроки создания информационных систем, но минимизировать ошибки. Ошибки и неточности встречаются постоянно, чем раньше они диагностируется и локализуется, тем меньше стоимость переделки. Известно, что стоимость выявление и устранение ошибки на стадии проектирования в два раза обходится дороже, на стадии тестирования информационной системы в десять раз, а на стадии эксплуатации в сто раз, чем на стадии анализа бизнес-процессов и разработки технического задания [21].

При создании сложных информационных систем зачастую очень трудно понять требования персонала заказчика. Они могут быть сформулированы некорректно, а в процессе анализа конкретных бизнес-

процессов даже измениться. Поэтому появление методологий современного проектирования и моделирования информационных систем было насущной задачей, над которой работали специалисты разных стран.

Появление автоматизированных систем управления в шестидесятых годах прошлого столетия определялось получением начальных знаний и опыта их разработки и внедрения. Анализировались все успехи и неудачи создания первых АСУ, но бесспорным было сокращение времени обработки информации, производственных и управленческих затрат и как следствие персонала.

Опыт зарубежных компаний по разработке и внедрению корпоративных информационных систем свидетельствует о появлении программ, в первую очередь связанных с автоматизацией учетных функций бухгалтерий, отдела кадров и складов. И намного позже появляются другие автоматизированные системы управления производством, логистикой, взаимоотношениями с клиентами и поставщиками. На последнем этапе разрабатываются информационные системы управления всей компанией, позволяющие полностью перейти к электронному документообороту и автоматизировать все сферы деятельности фирмы. С появлением персональных компьютеров происходит децентрализация процессов управления, все чаще внедряются модули с распределенными системами обработки информации [23].

На следующем этапе, в семидесятых годах прошлого столетия пришло понимание, что информация – стратегический ресурс любой компании, который необходимо грамотно использовать. В тоже время главными потребителями информации являются руководители. Идеи использования распределенных систем не находят пока применения из-за отсутствия компактной вычислительной техники, которая появится позднее и перевернет весь мир. В компаниях и организациях создаются информационные отделы и службы, вычислительные центры и лаборатории.

Восьмидесятые годы характеризуются появлением специализированных методологий проектирования информационных систем и CASE-средств. На их основе разрабатываются первые программные средства, а персональные компьютеры позволяют приступить к созданию децентрализованных информационных систем. Различные персональные компьютеры объединяются в локальную сеть. Этот период характеризуется интеграций информационных систем и появлением различных концепций управления ими на единой методологической основе.

Девяностые годы стали триумфом персональных компьютеров. Невысокая стоимость и компактные размеры сделали их чрезвычайно популярными и общедоступными для индивидуализации использования при решении управленческих задач. Разрабатываются корпоративные информационные системы, реализующие принципы распределенной обработки данных. Становится возможным автоматизация всех отделов и служб компаний, а не только бухгалтерии. Появляются системы электронного документооборота, в том числе для предприятий с развитой филиальной сетью в разных городах и регионах. Сокращаются сроки обработки данных, производственных, складских и прочих управленческих отчетов.

Появление и развитие методологий моделирования и проектирования информационных систем не было простым процессом. На всех этапах этого пути были талантливые, энергичные, необычайно трудолюбивые люди, которые вкладывали свои знания, силы, опыт, а порой и денежные средства для успешной реализации информационных проектов. Вот некоторые из них:

В СССР основателем и теоретиком автоматизированных систем управления был выдающийся ученый академик В.М. Глушков. Под его руководством в 1963-1964 годах в Институте кибернетики Академии наук СССР были начаты работы по созданию автоматизированных систем сбора, учета, обработки данных, оперативно-календарного планирования производства на базе отечественной вычислительной техники. При этом

ставилась задача разработки универсальной АСУ, пригодной для внедрения на многих предприятиях страны. Опытная эксплуатация и апробация проходили на самых современных предприятиях государства, таких как Львовский телевизионный завод «Электрон» и Кунцевский радиозавод. Некоторые решения были признаны и за рубежом, так была предложена общая алгоритмическая схема последовательного анализа вариантов, включавшая в себя вычислительные методы динамического программирования (В. С. Михалевича и Н. З. Шора). В.В. Шкурба развил эту схему вместе с методами имитационного моделирования для решения задач упорядочения, в частности в теории расписаний и календарного планировании.

В 1970-1980-х годах информационные системы интегрировались в комплексные АСУ, решающие задачи автоматизированного проектирования новых изделий, технологической подготовки производства и автоматизации организационного управления предприятием. Комплексные АСУ были разработаны и внедрены на Ульяновском авиационном заводе и других предприятиях оборонного комплекса под руководством А. А.Морозова, В. И. Скурихина. Комплексные АСУ создавались научно-исследовательскими институтами такими как, Всесоюзного объединения “Союзсистемпром” Минприбора СССР: ЦНИИТУ, г. Минск; ГНИПИ ВТ, г. Казань; НИИУМС, г. Пермь и др.

В США Дуглас Т. Росс (SoftTech, Inc) разработал язык АПТ для работы станков с ЧПУ, который в средине 60-х положил начало разработкам графических языков моделирования. А в 1969 году им же предложена методология SADT (IDEF0) для моделирования информационных систем средней сложности, в рамках программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой департаментом Военно-Воздушных Сил США в рамках большого аэрокосмического проекта.

К концу двадцатого века было разработано несколько десятков методов моделирования сложных систем. Они все были разные по функциональным

возможностям, но во многом имели схожие подходы к анализу и описанию предметной области. Возникла острая необходимость объединения удачных решений в одну методику, которая устраивала большую часть разработчиков информационных систем. В результате этих процессов был разработан язык UML.

У многих ведущих компаний, таких как Rational Software, Oracle Corporation, IBM, Microsoft, HewlettPackard, i-Logix, Texas Instruments и Unisys была четкая уверенность в необходимости создания подобного языка программирования. С этой целью был создан консорциум UML Partners, рабочую группу по семантике UML возглавил Крис Кобрин. Ведущую роль в создании унифицированного языка моделирования (UML) сыграли Гради Буч, Айвар Джекобсон и Джеймс Рамбо, работающие в компании Rational Software. Ими разработаны следующие методы объектного моделирования сложных информационных систем [2,3,4]:

1. Метод объектного моделирования программного обеспечения сложных информационных систем (метод Буча).
2. Метод анализа требований к бизнес- приложениям (метод Джекобсона).
3. Метод анализа обработки данных в сложных информационных системах (метод Рамбо).

Предварительная версия 0.8 унифицированного метода программирования была выпущена в октябре 1995 года. Первая версия UML 0.9 вышла в июне 1996 году и получила мощную поддержку Группы OMG, занимающейся разработкой единых стандартов в сфере web-технологий, включающую в себя несколько сотен компаний, работающих в области IT-технологий и производстве компьютерной техники. Это позволило выпустить в первые года сразу несколько версий. Так, в 1997 году появляется сразу две версии UML 1.0 и UML 1.1. В 1998 году разработчиками представляется версия UML 1.2, в 1999 году версия UML 1.3, в 2001 году

выходит версия UML 1.4, а в 2003 году версия UML 1.5. Эта версия и принимается в качестве международного стандарта ISO/IEC 19501-2005.

Сейчас наиболее популярна версия UML 2.4.1, вышедшая в 2011 году, которая тоже оформлена виде международных стандартов ISO/IEC 19505-1 и 19505- 2. Для нее разработаны инструментальные средства поддержки и визуального программирования, осуществляющих прямую генерацию кода из моделей UML, в основном посредством языков программирования C++ и Java. Среди них программы Rational Rose и Visual Paradigm for UML [29].

В настоящее время методологии и средства моделирования бизнес-процессов, процессно- ориентированных методов анализа и проектирования информационных систем широко представлены как в России, так и в большинстве стран мира.

1.3 Процессы и модели жизненного цикла информационных систем

В соответствии с ГОСТ Р ИСО/МЭК 12207- 99 процессы жизненного цикла включают себя работы, которые могут выполняться в жизненном цикле программных средств, распределены по пяти основным, восьми вспомогательным и четырем организационным процессам [18].

Основные процессы жизненного цикла. Основные процессы жизненного цикла состоят из пяти процессов, которые реализуются под управлением основных сторон, вовлеченных в жизненный цикл программных средств. Под основной стороной понимают одну из тех организаций, которые инициируют или выполняют разработку, эксплуатацию или сопровождение программных продуктов. Основными сторонами являются заказчик, поставщик, разработчик, оператор и персонал сопровождения программных продуктов. Основными процессами являются:

1. **Процесс заказа**. Определяет работы заказчика, то есть организации, которая приобретает систему, программный продукт или программную услугу.

2. **Процесс поставки**. Определяет работы поставщика, то есть организации, которая поставляет систему, программный продукт или программную услугу заказчику.

3. **Процесс разработки**. Определяет работы разработчика, то есть организации, которая проектирует и разрабатывает программный продукт.

4. **Процесс эксплуатации**. Определяет работы оператора, то есть организации, которая обеспечивает эксплуатационное обслуживание вычислительной системы в заданных условиях в интересах пользователей.

5. **Процесс сопровождения**. Определяет работы персонала сопровождения, то есть организации, которая предоставляет услуги по сопровождению программного продукта, состоящие в контролируемом изменении программного продукта с целью сохранения его исходного состояния и функциональных возможностей. Данный процесс охватывает перенос и снятие с эксплуатации программного продукта.

Вспомогательные процессы жизненного цикла [17]. Вспомогательные процессы жизненного цикла состоят из восьми процессов. Вспомогательный процесс является целенаправленной составной частью другого процесса, обеспечивающей успешную реализацию и качество выполнения программного проекта. Вспомогательный процесс, при необходимости, инициируется и используется другим процессом. Вспомогательными процессами являются:

1. **Процесс документирования**. Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.

2. **Процесс управления конфигурацией.** Определяет работы по управлению конфигурацией.

3. **Процесс обеспечения качества**. Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных планов. Совместные анализы, аудиторские проверки,

верификация и аттестация могут использоваться в качестве методов обеспечения качества.

4. **Процесс верификации**. Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.

5. **Процесс аттестации**. Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.

6. **Процесс совместного анализа**. Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.

7. **Процесс аудита**. Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).

8. **Процесс решения проблемы**. Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

Организационные процессы жизненного цикла [18]. Организационные процессы жизненного цикла состоят из четырех процессов. Они применяются в какой-либо организации для создания и реализации основной структуры, охватывающей взаимосвязанные процессы жизненного цикла и соответствующий персонал, а также для постоянного совершенствования данной структуры и процессов. Эти процессы, как правило, являются типовыми, независимо от области реализации конкретных проектов и договоров; однако уроки, извлеченные из таких проектов и

договоров, способствуют совершенствованию организационных вопросов. Организационными процессами являются:

1. **Процесс управления**. Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.

2. **Процесс создания инфраструктуры**. Определяет основные работы по созданию основной структуры процесса жизненного цикла.

3. **Процесс усовершенствования**. Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.

4. **Процесс обучения**. Определяет работы по соответствующему обучению персонала.

Данный ГОСТ Р ИСО/МЭК 12207-99 охватывает жизненный цикл программных средств от концепции замыслов через определение и объединение процессов для заказа и поставки программных продуктов и услуг. Кроме того, данная структура предназначена для контроля и модернизации данных процессов.

Процессы, определенные в настоящем стандарте, образуют множество общего назначения. Конкретная организация, в зависимости от своих целей, может выбрать соответствующее подмножество процессов для выполнения своих конкретных задач. Поэтому настоящий стандарт следует адаптировать для конкретной организации, проекта или приложения. Настоящий стандарт предназначен для использования как в случае отдельно поставляемых программных средств, так и для программных средств, встраиваемых или интегрируемых в общую систему [17].

Модели жизненного цикла информационной системы. Модель жизненного цикла информационной системы может включать:

1) стадии;

2) основные результаты выполнения работ на каждой стадии;

3) ключевые события.

Под стадией понимается определенный этап процесса разработки информационной системы.

Жизненный цикл информационной системы характеризуется периодом времени от идеи создания информационной системы и заканчивая моментом вывода ее из эксплуатации и включает в себя следующие стадии.

1. Предпроектное обследование.
2. Проектирование.
3. Создание информационной системы.
4. Ввод в эксплуатацию.
5. Эксплуатация информационной системы.
6. Вывод из эксплуатации.

Процессы жизненного цикла информационных систем представлены на рисунке 1.

Процессы жизненного цикла, определенны стандартом ГОСТ Р ИСО/МЭК 15288-2005, могут применяться любой организацией при приобретении и использовании или создании и поставке системы. Они распространяются на любой уровень системной иерархии и на любую стадию жизненного цикла.

Процессы жизненного цикла основываются на принципах модульности (максимальная слаженность функций процесса и минимальная связь между процессами) и собственности (процесс связывается с ответственностью). Функции, которые осуществляются данными процессами, определяются в зависимости от конкретных целей, результатов и набора действий, составляющих данный процесс. Процессы, описанные в настоящем стандарте, не препятствуют и не исключают использование дополнительных процессов, которые организация посчитает необходимыми [20].

Управление процессами жизненного цикла ИС. Цель управления процессами жизненного цикла системы заключается в гарантировании

доступности эффективных процессов жизненного цикла для использования организацией. Данный процесс обеспечивает процессы жизненного цикла системы, которые согласованы с целями и политикой организации, определены, адаптированы и поддержаны соответствующим образом для учета особенностей отдельных проектов и способны реализовываться с помощью эффективных проверенных методов и инструментальных средств.

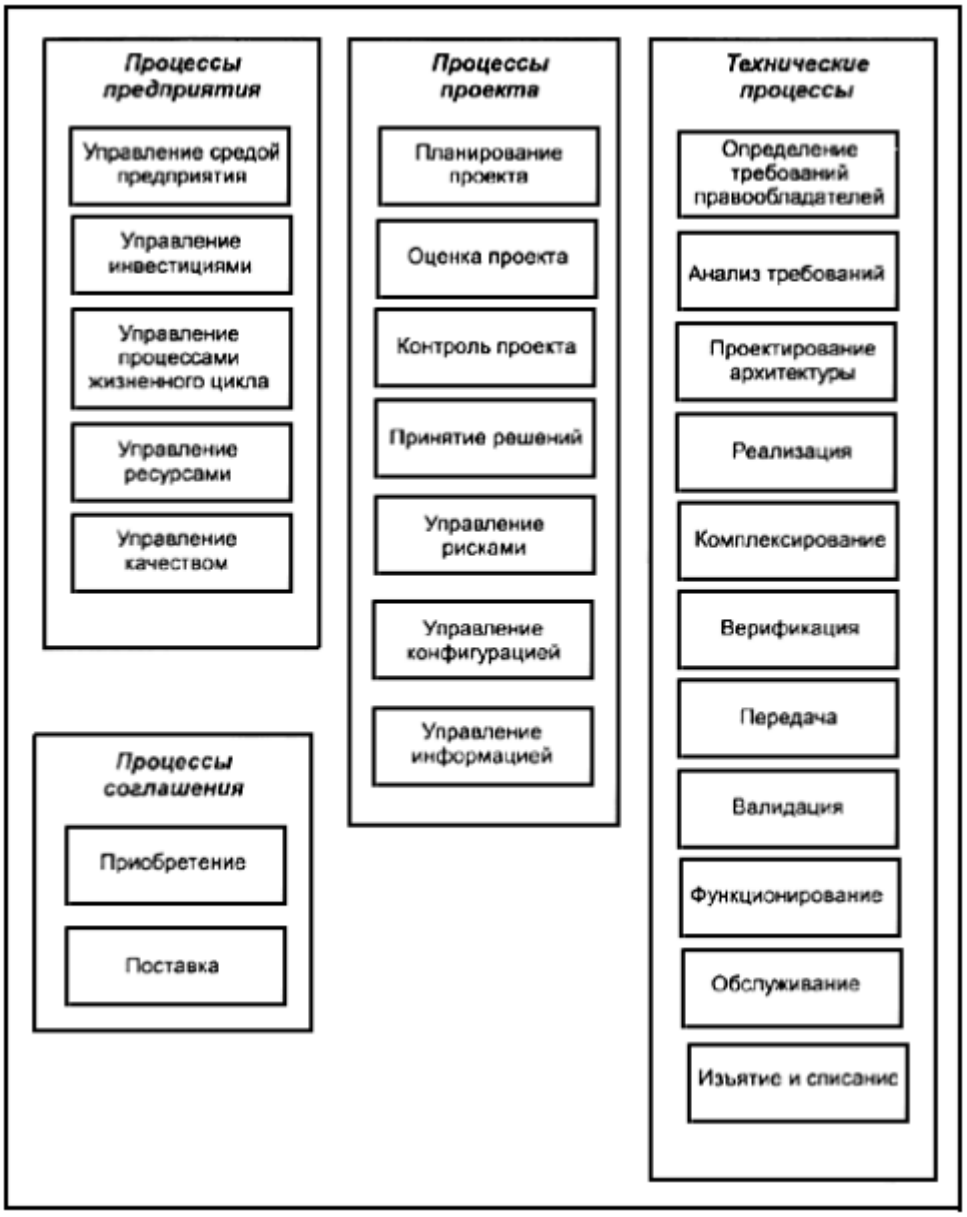


Рисунок 1 - Процессы жизненного цикла информационной системы

В результате эффективного управления процессами жизненного цикла системы:

a) определяются процессы жизненного цикла системы, которые будут использоваться организацией;

b) определяется политика применения процессов жизненного цикла системы;

c) определяется политика адаптации процессов жизненного цикла системы для удовлетворения потребностей отдельных проектов;

d) определяются критерии оценки результатов применения процессов жизненного цикла системы;

e) предпринимаются действия по совершенствованию способов определения и применения процессов жизненного цикла системы.

При реализации процессов управления процессами жизненного цикла системы организация должна осуществлять следующие действия в соответствии с принятой политикой и процедурами:

a) устанавливать стандартные наборы процессов жизненного цикла систем для соответствующих стадий жизненного цикла системы;

b) определять приемлемые политику и процедуры адаптации и требования к их утверждению;

c) определять методы и инструментальные средства, которые поддерживают выполнение процессов жизненного цикла системы;

d) по возможности устанавливать показатели, которые позволяют определять характеристики выполненных стандартных процессов;

e) контролировать выполнение процесса, сохранять и анализировать показатели процесса и определять тенденции по отношению к критериям предприятия;

f) определять возможности для усовершенствования стандартных процессов жизненного цикла систем;

g) совершенствовать имеющиеся процессы, методы и инструментальные средства, используя найденные возможности.

Каждая система имеет свой жизненный цикл. Жизненный цикл может быть описан с использованием абстрактной функциональной модели,

представляющей концептуализацию потребности в системе, ее реализации, применения, развития и ликвидации [19].

Система развивается на протяжении жизненного цикла в результате действий, осуществляемых и управляемых людьми, работающими в организациях и использующими определенные процессы в своей деятельности. Детали модели жизненного цикла выражаются в терминах этих процессов, их результатов, взаимосвязи и возникновения. Настоящий стандарт определяет множество процессов, названных процессами жизненного цикла, при помощи которых может быть смоделирован жизненный цикл системы.

Жизненные циклы различаются по свойствам, целям, использованию системы, а также по преобладающим условиям. Тем не менее, несмотря на очевидное множество различий в жизненных циклах систем, существует базовый набор стадий жизненного цикла, составляющих полный жизненный цикл любой системы. Каждая стадия имеет определенную цель и вклад в полный жизненный цикл и рассматривается при планировании и выполнении жизненного цикла системы [16].

Стадии представляют собой основные периоды жизненного цикла, связанные с системой и относящиеся к состоянию описания системы или непосредственно к системе. Стадии отображают значимый прогресс и достижение запланированных этапов развития системы на протяжении всего жизненного цикла и дают начало важнейшим решениям относительно своих входов и выходов. Эти решения используются организациями для учета неопределенностей и рисков, непосредственно связанных с затратами, сроками и функциональностью при создании или применении системы. Таким образом, стадии обеспечивают организации структурой работ, в рамках которых управление предприятием обладает высокой способностью для обзора и контроля проекта и технических процессов.

Организации проходят стадии жизненного цикла различными способами, устраняя противоречия между стратегией осуществления бизнеса

и стратегией уменьшения рисков. Параллельное прохождение стадий или их прохождение в различном порядке может привести к формам жизненного цикла с совершенно разными характеристиками.

Часто в качестве альтернативных вариантов используются последовательная, инкрементная или эволюционная формы жизненного цикла; в отдельных случаях могут быть разработаны комбинации этих форм. Выбор и разработка организацией конкретных форм жизненного цикла зависят от ряда факторов, включая бизнес-контекст, природу и сложность системы, стабильность требований, технологические возможности, потребность в различных системных возможностях во времени и наличие бюджетных средств и ресурсов [14].

Аналогично тому, как все системные элементы осуществляют вклад в систему как в единое целое, так и каждая стадия жизненного цикла должна учитываться на любой другой ее стадии. Следовательно, участвующие стороны должны координировать свои действия и кооперироваться друг с другом на протяжении всего жизненного цикла. Синергия стадий жизненного цикла и сторон, вкладывающих средства в реализацию функциональностей на этих стадиях, является необходимой для успешного осуществления проектных мероприятий.

Тесная связь и, по возможности, единение проектных команд, различных функций и организаций, ответственных за другие стадии жизненного цикла, приводят к логичности и согласованности жизненного цикла [13]. Набольшее распространение получили следующие модели жизненного цикла информационных систем: каскадная (классическая или водопадная), итерационная и спиральная.

Каскадная (классическая, водопадная) модель жизненного цикла информационной системы. Модель была предложена в 1970 году Уинстоном Ройсом. Переход на следующий этап осуществляется после полного окончания работ по предыдущему этапу, при этом оформляется полный комплект рабочей документации. Все этапы выполняются в строгой

последовательности с утвержденными сроками и четкими затратами. Это основные достоинства каскадной модели ЖЦ ИС, которая применялась в условиях полной определенности решаемых задач и совершенно не приемлема, когда и разработчики, и заказчики не имеют четкого видения всех особенностей проектируемой ИС. Кроме того, невозможно идти дальше, пока не сдан предыдущий этап, а после сдачи нельзя возвращаться к нему для устранения обнаруженных недочетов, что серьезно затрудняет работы по совершенствованию и доработке, создаваемой ИС. Эта модель нравиться и заказчикам, и разработчикам по причине жесткой дисциплины финансирования этапов только после их предъявления. Но полностью отсутствует гибкость в работе над созданием ИС.

Каскадная модель представлена на рисунке 2. На практике, все же приходится возвращаться к предыдущим этапам и в этом случае, в последнее время наиболее востребованной стала итерационная модель ЖЦ ИС.

Рисунок 2 – Каскадная (водопадная, классическая) модель ЖЦ ИС

Итерационная модель жизненного цикла ИС. Поэтапная модель с промежуточным контролем — итерационная модель разработки

информационной системы. Каждый этап имеет обратные связи в процессе корректировки и создает условия для корректировки ранее созданных этапов. При этом трудоемкость работ и временные затраты существенно сокращаются по сравнению с водопадной моделью жизненного цикла. Итерационная модель ЖЦ информационной системы представлена на рисунке 3.

Рисунок 3 – Итерационная модель ЖЦ ИС

Создание информационной системы – это организованный процесс построения и последовательного преобразования согласованных моделей на всех этапах жизненного цикла. При этом все разработанные модели находятся в репозитории проекта и доступны всем разработчикам, что позволяет эффективно вести одновременную работу над проектированием и созданием информационной системы.

Спиральная модель жизненного цикла информационной системы. Спиральная модель предложена Барри Боэм в 1988 году и определяет, в основном стартовые этапы жизненного цикла информационной системы. При этом обосновывается и проверяется возможность реализации спроектированных технических решений. На каждом витке создается

прототип проектируемой информационной системы, который на следующих витках спирали ЖЦ ИС совершенствуется, дополняется и доводится до полного внедрения. При этом не обязательно дожидаться окончания каждого этапа, данная модель позволяет переходить на следующие витки спирали и решать проблемы или недоделки на следующем уровне, что делает работу над проектом более эффективной, гибкой и завершить в более сжатые сроки.

Новый виток спирали соответствует поэтапной модели создания фрагмента информационной системы. При использовании спиральной модели ЖЦ:

- происходит ориентация на модернизацию информационной системы;
- осуществляется аккумулирование всех решений в процессе проектирования и создания моделей и прототипов информационной системы;
- проводится анализ издержек и всех рисков в процессе проектирования ИС.

Спиральный процесс состоит из следующей повторяющейся последовательности:

1. Определение требований.
2. Анализ.
3. Проектирование.
4. Реализация и тестирование.
5. Интеграция.
6. Внедрение.

Этот многократный цикл, завершающийся созданием новой версии информационной системы представлен на рисунке 4.

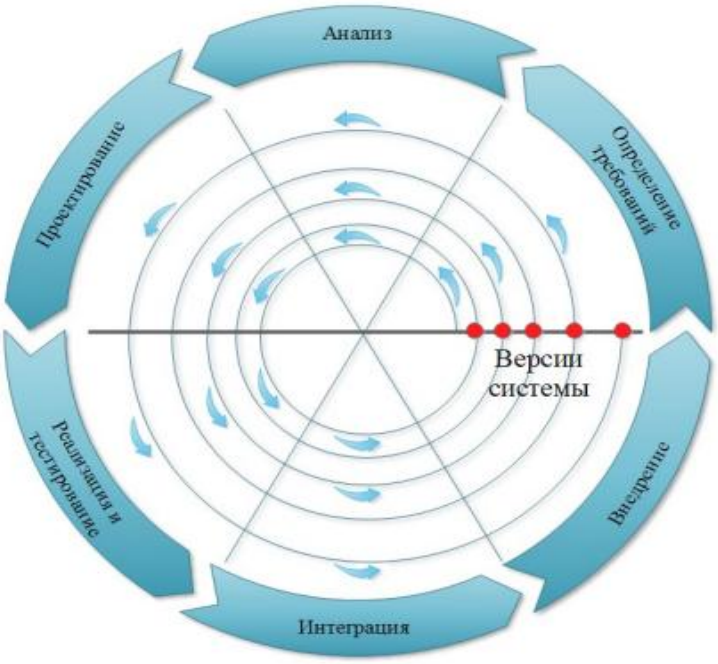


Рисунок 4 – Спиральная модель ЖЦ ИС

Для применения спиральной модели ЖЦ ИС может быть несколько причин, это необходимость минимизации рисков и возможность представления заказчику прототип или эскизную версию проекта для конкретизации пожеланий и учета их в следующих циклах. А также в случае если разрабатываемая информационная система достаточно сложна и существует реальная необходимость создавать промежуточные версии продукта, не откладывая эту работу на финишные этапы, как это предписывает водопадная модель.

Основная задача спиральной модели жизненного цикла информационной системы заключается в том, чтобы на каждой итерации создавать очередную версию системы, используя разработанный прототип предыдущих этапов. Такая модель позволяет более гибко работать с заказчиком, постоянно учитывать его замечания и предложения, совершенствовать проектируемую систему в процессе каждого нового витка спирали.

2 МЕТОДОЛОГИИ СОВРЕМЕННОГО ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

При разработке ИС применяется множество различных методологий и процессов.

Под методологией понимают набор некоторых принципов, методик и практик, применяя которые, можно достигать определенных целей и результатов.

Процессы, в отличие от методологий, представляют собой формальное описание всех бизнес-процессов разработки ИС, следуя которым, можно достичь конечную цель — получение готового программного продукта.

В настоящее время наибольшее распространение в практике все больше находят так называемые гибкие методологии разработки (англ. Agile software development, далее — agile-методы). Гибкие методологии — это семейство подходов к разработке программного обеспечения, которые ориентированы:

— на применение итеративной разработки;

— динамическое формирование требований;

— обеспечение реализации требований.

Основная цель большинства гибких методологий — обеспечить минимизацию рисков. Это достигается организацией разработки в виде серии коротких циклов — итераций. Задача каждой итерации — представить пользователю версию программного продукта. Поэтому каждая итерация в результате представляет собой программный проект. В процессе итерации решаются все задачи, которые необходимы для создания в проекте уточняющих дополнений, такие как: планирование; анализ требований; проектирование; программирование; тестирование; документирование.

Существует много методик, относящихся к классу гибких методологий разработки. В рамках настоящего параграфа приведем список самых известных из них.

2.1 Методология функционального моделирования работ SADT

Методология SADT (Structured Analysis and Design Technique - методология структурного анализа и проектирования), разработанная Дугласом Т. Россом в 1969-1973 годах базируется на структурном анализе систем и графическом представление организации в виде системы функций, которые имеют три класса структурных моделей:

1. Функциональная модель.
2. Информационная модель.
3. Динамическая модель.

Процесс моделирования по методологии SADT состоит из следующих этапов:

1. Сбор информации и анализ информации о предметной области.
2. Документирование полученной информации.
3. Моделирование.
4. Корректура модели в процессе итеративного рецензирования.

Основным достоинством этой методологии являются простота и наглядность. В качестве недостатка – невозможность описать реакцию описываемого процесса на изменяющиеся внешние факторы. Для этих целей служат другие методологии.

SADT реализуется в следующих нотациях.

1. IDEF0 (Icam Definition) - функциональные модели и соответствующие диаграммы. SADT-модель, представляющая систему в виде иерархии взаимосвязанных функций, которые выполняет система, называется *функциональной моделью*. Функциональная модель показывает, какие функции выполняет исследуемая система, как эти функции связаны между собой и как они упорядочены по степени важности или по порядку исполнения. Каждая функция, представленная в модели, может быть детализирована с любой степенью подробности, то есть разложена на составляющие ее функции, каждая их которых также может быть разложена на составляющие и т.д., пока не будет достигнута необходимая степень точности ответа на вопросы, поставленные относительно системы.

Функциональная модель строится с помощью графического языка диаграмм. Каждая функция в модели может быть детально описана в виде отдельной диаграммы.

Как разновидность SADT-моделирования функциональное моделирование обозначилось под названием стандарт IDEF0.

Методология IDEF0 нашла широкое признание и применение, в первую очередь, благодаря простой графической нотации, используемой для построения модели. Главными компонентами модели являются диаграммы. На них отображаются функции системы в виде прямоугольников, а также связи между ними и внешней средой посредством стрелок. Использование всего лишь двух графических примитивов (прямоугольник и стрелка) позволяют быстро объяснить правила и принципы построения диаграмм IDEF0 людям, незнакомым с данной методологией. Это достоинство позволяет подключить и активизировать деятельность заказчика по

описанию бизнес-процессов с использованием формального и наглядного графического языка.

Модель может содержать *4 типа диаграмм*:

- контекстную диаграмму;
- диаграммы декомпозиции;
- диаграммы дерева узлов;
- диаграммы только для экспозиции (for exposition only, FEO).

Контекстная диаграмма (диаграмма верхнего уровня), являясь вершиной древовидной структуры диаграмм, показывает назначение системы (основную функцию) и ее взаимодействие с внешней средой. В каждой модели может быть только одна контекстная диаграмма. После описания основной функции выполняется функциональная декомпозиция, то есть определяются функции, из которых состоит основная.

Далее функции делятся на подфункции и так до достижения требуемого уровня детализации исследуемой системы. Диаграммы, которые описывают каждый такой фрагмент системы, называются *диаграммами декомпозиции*. После каждого сеанса декомпозиции проводятся сеансы экспертизы – эксперты предметной области указывают на соответствие реальных процессов созданным диаграммам. Найденные несоответствия устраняются, после чего приступают к дальнейшей детализации процессов.

Диаграмма дерева узлов показывает иерархическую зависимость функций (работ), но не связи между ними. Их может быть несколько, поскольку дерево можно построить на произвольную глубину и с произвольного узла.

Диаграммы для экспозиции строятся для иллюстрации отдельных фрагментов модели с целью отображения альтернативной точки зрения на происходящие в системе процессы (например, с точки зрения руководства организации).

На следующем рисунке показаны основные элементы графической нотации IDEF0 [25, 35].

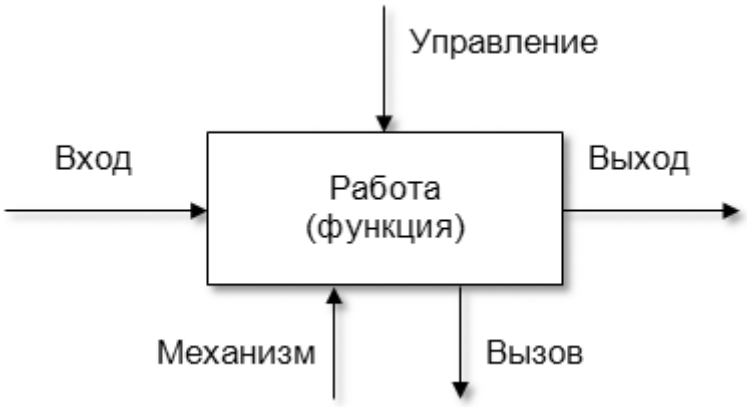


Рисунок 5 - Элементы графической нотации IDEF0

Прямоугольник представляет собой *работу (процесс, деятельность, функцию или задачу)*, которая имеет фиксированную цель и приводит к некоторому конечному результату. Имя работы должно выражать действие (например, «Изготовление детали», «Формирование ведомости ЦДЛ № 3»).

Взаимодействие работ между собой и внешним миром описывается в виде стрелок. В IDEF0 различают *5 видов стрелок*:

- *вход* (англ. input) – материал или информация, которые используются и преобразуются работой для получения результата (выхода). Вход отвечает на вопрос «Что подлежит обработке?». В качестве входа может быть как материальный объект (сырье, деталь, экзаменационный билет), так и не имеющий четких физических контуров (запрос к БД, вопрос преподавателя). Допускается, что работа может не иметь ни одной стрелки входа. Стрелки входа всегда рисуются входящими в левую грань работы;

- *управление* (англ. control) – управляющие, регламентирующие и нормативные данные, которыми руководствуется работа. Управление отвечает на вопрос «В соответствии с чем выполняется работа?». Управление влияет на работу, но не преобразуется ей, т.е. выступает в качестве ограничения. В качестве управления могут быть правила, стандарты, нормативы, расценки, устные указания. Стрелки управления рисуются входящими в верхнюю грань работы. Если при построении диаграммы

возникает вопрос, как правильно нарисовать стрелку сверху или слева, то рекомендуется ее рисовать как вход (стрелка слева);

- *выход* (англ. output) – материал или информация, которые представляют результат выполнения работы. Выход отвечает на вопрос «Что является результатом работы?». В качестве выхода может быть как материальный объект (деталь, автомобиль, платежные документы, ведомость), так и нематериальный (выборка данных из БД, ответ на вопрос, устное указание). Стрелки выхода рисуются исходящими из правой грани работы;

- *механизм* (англ. mechanism) – ресурсы, которые выполняют работу. Механизм отвечает на вопрос «Кто выполняет работу или посредством чего?». В качестве механизма могут быть персонал предприятия, студент, станок, оборудование, программа. Стрелки механизма рисуются входящими в нижнюю грань работы;

- *вызов* (англ. call) – стрелка указывает, что некоторая часть работы выполняется за пределами рассматриваемого блока. Стрелки выхода рисуются исходящими из нижней грани работы.

2. DFD (Data Flow Diagrams) - диаграммы потоков данных, моделируют движение информации в системе. При построении функциональной модели системы альтернативой нотации IDEF0 является нотация *диаграмм потоков данных* (Data Flow Diagrams, DFD). В отличие от IDEF0, предназначенной для проектирования систем вообще, DFD предназначена для проектирования информационных систем. Ориентированность этой методологии на проектирование автоматизированных систем делает ее удобным и более выгодным инструментом при построении функциональной модели TO-BE.

Как и в IDEF0, основу нотации DFD составляет графический язык описания процессов. Авторами одной из первых графических нотаций DFD (1979 г.) стали Эд Йордан (Yourdon) и Том де Марко (DeMarko).

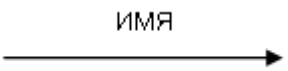
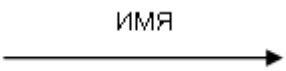
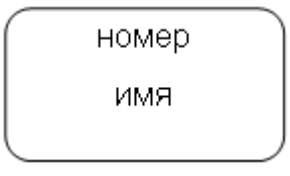
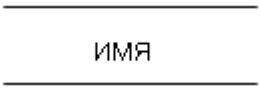
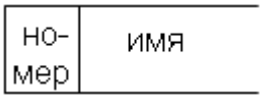
В настоящее время наиболее распространенной является нотация Гейна-Сарсона (Gane-Sarson).

Модель системы в нотации DFD представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе. Модель системы содержит контекстную диаграмму и диаграммы декомпозиции. Принципы построения функциональной модели с помощью DFD аналогичны принципам методологии IDEF0. Вначале строится контекстная диаграмма, где отображаются связи системы с внешним окружением. В дальнейшем выполняется декомпозиция основных процессов и подсистем с построением иерархии диаграмм.

Согласно DFD источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации [24, 27].

При построении диаграмм различают элементы графической нотации, представленные в табл. 1.

Таблица 1 - Элементы графической нотации DFD

Наименование	Нотация Йордана	Нотация Гейна-Сарсона
Поток данных		
Процесс (система, подсистема)		
Накопитель данных		

Внешняя сущность		

Поток данных определяет информацию (материальный объект), передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т. д.

Каждый поток данных имеет имя, отражающее его содержание. Направление стрелки показывает направление потока данных. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться назад в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним – двунаправленным.

На диаграммах IDEF0 потоки данных соответствуют входам и выходам, но в отличие от IDEF0 стрелки потоков на DFD могут отображаться входящими и выходящими из любой грани внешней сущности, процесса или накопителя данных [37].

Процесс (в IDEF0 – функция, работа) представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом.

Каждый процесс должен иметь имя в виде предложения с глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например:

- «Ввести сведения о клиентах»;
- «Рассчитать допускаемую скорость»;
- «Сформировать ведомость допускаемых скоростей»

Номер процесса служит для его идентификации и ставится с учетом декомпозиции. В отличие от IDEF0 вложенность процессов обозначается через точку (например, в IDEF0 – «236», в DFD – «2.3.6»).

Преобразование информации может показываться как с точки зрения процессов, так и с точки зрения *систем* и *подсистем*. Если вместо имени процесса «Рассчитать допускаемую скорость» написать «Подсистема расчета допускаемых скоростей», тогда этот блок на диаграмме стоит рассматривать, как подсистему.

Накопитель (хранилище) данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, области в оперативной памяти, файла на магнитном носителе и т.д.

Накопителю обязательно должно даваться уникальное имя и номер в пределах всей модели (всего набора диаграмм). Имя накопителя выбирается из соображения наибольшей информативности для разработчика. Например, если в качестве накопителей выступают таблицы проектируемой базы данных, тогда в качестве имен накопителей рекомендуется использовать имена таблиц. Таким образом, накопитель данных может представлять собой всю базу данных целиком, совокупность таблиц или отдельную таблицу. Такое представление накопителей в дальнейшем облегчит построение информационной модели системы.

Внешняя сущность (терминатор) представляет собой материальный объект или физическое лицо, выступающие как источник или приемник информации (например, заказчики, персонал, программа, склад, инструкция). Внешние сущности на DFD по смыслу соответствуют управлению и механизмам, отображаемым на контекстной диаграмме IDEF0.

Определение некоторого объекта, субъекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ проектируемой информационной системы. В связи с этим внешние сущности, как правило, отображаются только на контекстной диаграмме DFD. В процессе анализа и проектирования некоторые внешние сущности могут быть перенесены на диаграммы декомпозиции, если это необходимо, или, наоборот, часть процессов (подсистем) может быть представлена как внешняя сущность.

3. ***IDEF1X или ERD*** (Entity-Relationship Diagrams) - диаграммы "сущность-связь". SADT-модель, которая ориентирована на объекты, входящие в исследуемую систему, их свойства и связи между ними, называется моделью данных. Обычно, это не что иное, как реляционная модель данных исследуемой системы, которая состоит из сущностей, описываемых наборов атрибутов, и связей между ними. Типы связей определяют характер сущностей. Модель данных может быть положена в основу информационной модели исследуемой системы, создаваемой с помощью различных реляционных СУБД [36].

CASE-средства, в частности ERwin, поддерживающие эту методологию, позволяют строить логическую, независимую от СУБД, модель данных для общего представления системы и входящих в нее объектов и физическую модель данных, которая может быть трансформирована в любую реляционную СУБД и описана на языке описания данных этой СУБД. CASE средство ERwin, например, поддерживает более 20 СУБД, на языке описания данных которых может быть сгенерирована физическая модель данных.

Результатом использования данной нотации является концептуальная модель. *Цель концептуального проектирования* – создание концептуальной схемы данных на основе представлений о предметной области каждого отдельного типа пользователей. *Концептуальная схема* представляет собой описание основных сущностей (таблиц) и связей между ними без учета

принятой модели БД и синтаксиса целевой СУБД. Часто на такой схеме отображаются только имена сущностей (таблиц) без указания их атрибутов. *Представление пользователя* включает в себя данные, необходимые конкретному пользователю для принятия решений или выполнения некоторого задания.

Ниже рассмотрим последовательность шагов при концептуальном проектировании [5, 6].

1. Выделение сущностей. Первый шаг в построении концептуальной схемы данных состоит в определении основных объектов (сущностей), которые могут интересовать пользователя и, следовательно, должны храниться в БД. При наличии функциональной модели IDEF0 прообразами таких объектов являются входы, управления и выходы. Еще лучше для этих целей использовать DFD. Прообразами объектов в этом случае будут накопители данных. Как было отмечено выше, накопитель данных является совокупностью таблиц (набором объектов) или непосредственно таблицей (объектом). Для более детального определения набора основных объектов необходимо также проанализировать потоки данных и весь методический материал, требуемый для решения задачи. В ходе анализа и проектирования информационной модели наборы объектов должны быть детализированы. Возможные трудности в определении объектов связаны с использованием постановщиками задачи:

- примеров и аналогий при описании объектов (например, вместо обобщающего понятия «работник» они могут упоминать его функции или занимаемую должность: «руководитель», «ответственный», «контролер», «заместитель»);

- синонимов (например, «допускаемая скорость» и «установленная скорость», «разработка» и «проект», «барьерное место» и «ограничение скорости»);

- омонимов (например, «программа» может обозначать компьютерную программу, план предстоящей работы или программу телепередач).

Далеко не всегда очевидно то, чем является определенный объект – сущностью, связью или атрибутом. Например, как следует классифицировать «семейный брак»? На практике это понятие можно вполне обоснованно отнести к любой из упомянутых категорий. Анализ является субъективным процессом, поэтому различные разработчики могут создавать разные, но вполне допустимые интерпретации одного и того же факта. Выбор варианта в значительной степени зависит от здравого смысла и опыта проектировщика.

Каждая сущность должна обладать некоторыми свойствами:

- должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация;

- обладать одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;

- обладать одним или несколькими атрибутами (первичным ключом), которые однозначно идентифицируют каждый экземпляр сущности, то есть делают уникальной каждую строку таблицы;

- может обладать любым количеством связей с другими сущностями.

В графической нотации IDEF1X для отображения сущности используются обозначения, изображенные на следующем рисунке.

а) б)

независимая зависимая

Рисунок 6 – Сущности

Сущность в методологии IDEF1X является *независимой (сильной, родительской, доминантной, владельцем)*, если сущность не зависит от существования другой сущности (другими словами, каждый экземпляр сущности может быть однозначно идентифицирован без определения его

связей с другими сущностями, или уникальность экземпляра определяется только собственными атрибутами). Сущность называется *зависимой (слабой, дочерней, подчиненной)*, если ее существование зависит от существования других сущностей. Терминология «родительская» – «дочерняя» и «владелец» – «подчиненный» также может использоваться в отношении двух зависимых сущностей, если экземпляры одной из них (дочерней, подчиненной) могут быть однозначно определены с использованием экземпляров другой (родительской, владельца), несмотря на то, что вторая сущность в свою очередь зависит от третьей сущности.

2. Определение атрибутов. Как правило, атрибуты указываются только для сущностей. Если у связи имеются атрибуты, то это указывает на тот факт, что связь является сущностью. Самый простой способ определения атрибутов – после идентификации сущности или связи, задать себе вопрос «Какую информацию требуется хранить о …?». Существенно помочь в определении атрибутов могут различные бумажные и электронные формы и документы, используемые в организации при решении задачи. Это могут быть формы, содержащие как исходную информацию (например, «Ведомость возвышений наружного рельса в кривых»), так и результаты обработки данных (например, «Форма № 1»).

Выявленные атрибуты могут быть следующих видов:

- простой (атомарный, неделимый) – состоит из одного компонента с независимым существованием (например, «должность работника», «зарплата», «норма непогашенного ускорения», «радиус кривой» и т.д.);

- составной (псевдоатомарный) – состоит из нескольких компонентов (например, «ФИО», «адрес», и т. д.). Степень атомарности атрибутов, закладываемая в модель, определяется разработчиком. Если от системы не требуется выборки всех клиентов с фамилией Иванов или проживающих на улице Комсомольской, то составные атрибуты можно не разбивать на атомарные;

- однозначный – содержит только одно значение для одного экземпляра сущности (например, у кривой в плане может быть только одно значение радиуса, угла поворота, возвышения наружного рельса и т.д.);

- многозначный – содержит несколько значений (например, у одного отделения компании может быть несколько контактных телефонов);

- производный (вычисляемый) – значение атрибута может быть определено по значениям других атрибутов (например, «возраст» может быть определен по «дате рождения» и текущей дате, установленной на компьютере);

- ключевой – служит для уникальной идентификации экземпляра сущности (входит в состав первичного ключа), быстрого поиска экземпляров сущности или задания связи между экземплярами родительской и дочерней сущностей;

- неключевой (описательный);

- обязательный – при вводе нового экземпляра в сущность или редактировании обязательно указывается допустимое значение атрибута, т.е. после указанных действий оно не может быть неопределенным (NOT NULL). Атрибуты, входящие в первичный ключ сущности, являются обязательными.

После определения атрибутов задаются их *домены (области допустимых значений)*, например:

- наименование участка – набор из букв русского алфавита длиной не более 60 символов;

- поворот кривой – допустимые значения «Л» (влево) и «П» (вправо);

- радиус кривой – положительное число не более 4 цифр.

Задание доменов определяет набор допустимых значений для атрибута (нескольких атрибутов), а также тип, размер и формат атрибута (атрибутов).

На основании выделенного множества атрибутов для сущности определяется набор ключей. *Ключ* – один или несколько атрибутов сущности, служащих для однозначной идентификации ее экземпляров, их быстрого поиска или задания связи между экземплярами родительской и дочерней

сущностей. Ключи, используемые для однозначной идентификации экземпляров, подразделяются на следующие типы:

- *суперключ (superkey)* – атрибут или множество атрибутов, которое единственным образом идентифицирует экземпляр сущности. Суперключ может содержать «лишние» атрибуты, которые необязательны для уникальной идентификации экземпляра. При правильном проектировании структуры БД суперключом в каждой сущности (таблице) будет являться полный набор ее атрибутов;

- *потенциальный ключ (potential key)* – суперключ, который не содержит подмножества, также являющегося суперключом данной сущности, т. е. суперключ, содержащий минимально необходимый набор атрибутов, единственным образом идентифицирующих экземпляр сущности. Сущность может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то он называется составным ключом. Среди всего множества потенциальных ключей для однозначной идентификации экземпляров выбирают один, так называемый первичный ключ, используемый в дальнейшем для установления связей с другими сущностями;

- *первичный ключ (primary key)* – потенциальный ключ, который выбран для уникальной идентификации экземпляров внутри сущности;

- *альтернативные ключи (alternative key)* – потенциальные ключи, которые не выбраны в качестве первичного ключа.

Рассмотрим пример. Пусть имеется таблица, содержащая сведения о студенте, со следующими столбцами:

- фамилия;
- имя;
- отчество;
- дата рождения;
- место рождения;
- номер группы;

- ИНН;

- номер пенсионного страхового свидетельства (НПСС);

- номер паспорта;

- дата выдачи паспорта;

- организация, выдавшая паспорт.

Для каждого экземпляра (записи) в качестве суперключа может быть выбран весь набор атрибутов. Потенциальными ключами (уникальными идентификаторами) могут быть:

- ИНН;

- номер пенсионного страхового свидетельства;

- номер паспорта.

В качестве уникального идентификатора можно было бы выбрать совокупность атрибутов «Фамилия»+«Имя»+«Отчество», если вероятность учебы в вузе двух полных тезок была бы равна нулю.

Если в сущности нет ни одной комбинации атрибутов, подходящей на роль потенциального ключа, то в сущность добавляют отдельный атрибут – *суррогатный ключ (искусственный ключ, surrogate key)*. Как правило, тип такого атрибута выбирают символьный или числовой. В некоторых СУБД имеются встроенные средства генерации и поддержания значений суррогатных ключей (например, MS Access). Также стоит отметить, что некоторые разработчики вместо поиска потенциальных ключей и выбора из них первичного в каждую сущность добавляют искусственный атрибут, который в дальнейшем и используют в качестве первичного ключа.

Если потенциальных ключей несколько, то для выбора первичного ключа рекомендуется придерживаться следующих правил:

- количество атрибутов, входящих в ключ, должно быть минимальным (желательно, чтобы ключ был атомарным, то есть состоял из одного атрибута);

- размер ключа в байтах должен быть как можно короче;

- тип домена ключа – числовой. При выборе символьных атрибутов в ключ часто возникают проблемы с вводом ошибочных значений (путают регистр букв; добавляют лишние пробелы; используют буквы, пишущиеся на разных языках одинаково). В числовых атрибутах вероятность ошибки при вводе значения меньше;

- вероятность изменения значений ключа была наименьшей (например, «Номер пенсионного страхового свидетельства» более постоянный параметр, чем «ИНН» или «Номер паспорта»);

- с ключом проще всего работать пользователям (например, «Номер пенсионного страхового свидетельства» – это набор из 11 цифр, а «Номер паспорта» зависит от его вида: гражданина СССР, гражданина РФ или зарубежный).

Если некий атрибут (набор атрибутов) присутствует в нескольких сущностях, то его наличие обычно отражает наличие связи между экземплярами этих сущностей. В каждой связи одна сущность выступает как родительская, а другая – в роли дочерней. Это означает, что один экземпляр родительской сущности может быть связан с несколькими экземплярами дочерней. Для поддержки этих связей обе сущности должны содержать наборы атрибутов, по которым они связаны. В родительской сущности это первичный ключ. В дочерней сущности для моделирования связи должен присутствовать набор атрибутов, соответствующий первичному ключу родительской. Этот набор атрибутов в дочерней сущности принято называть *внешним ключом (foreign key)*.

В нотации IDEF1X атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой. Предварительная идентификация атрибутов на примере двух сущностей показана на следующем рисунке.

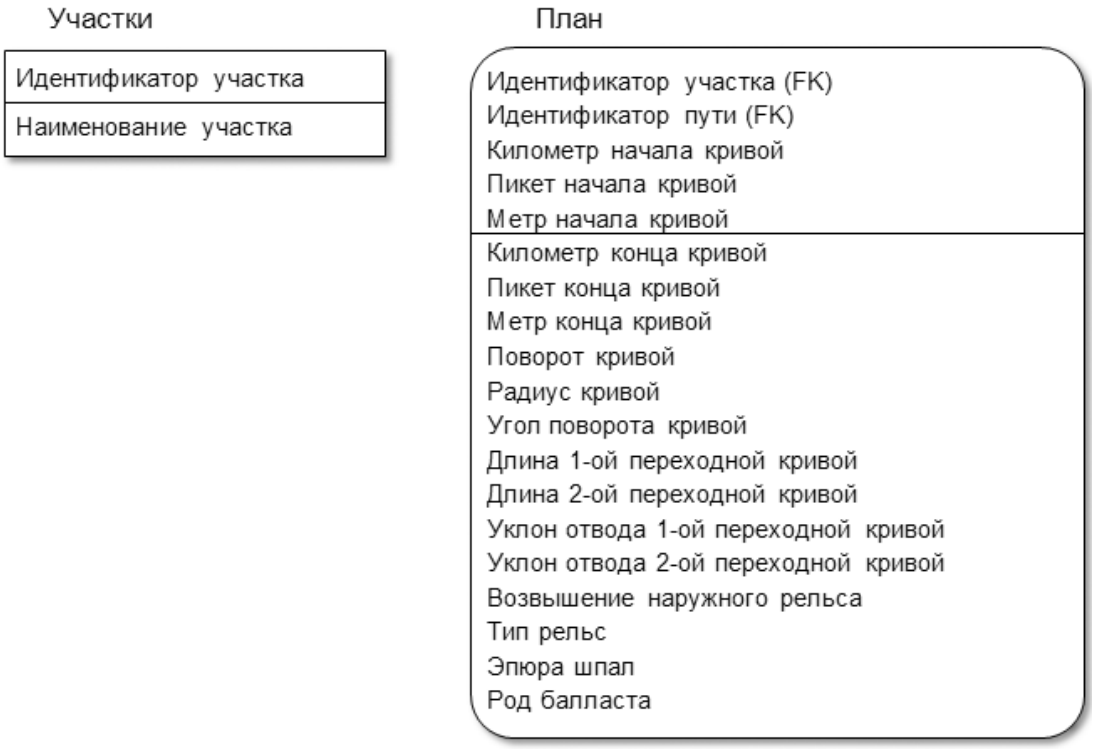


Рисунок 7 - Примеры сущностей

У независимой сущности «Участки» в качестве первичного ключа назначен суррогатный ключ, у зависимой сущности «План» – первичный ключ составной, состоящий из пяти атрибутов.

3. Определение связей. Наиболее характерными типами связей между сущностями являются:

- связи типа «часть–целое», определяемые обычно глаголами «состоит из», «включает» и т.п.;

- классифицирующие связи (например, «тип – подтип», «множество – элемент», «общее – частное» и т. п.);

- производственные связи (например, «начальник–подчиненный»);

- функциональные связи, определяемые обычно глаголами «производит», «влияет», «зависит от», «вычисляется по» и т. п.

Среди них выделяются только те связи, которые необходимы для удовлетворения требований к разработке БД.

Связь характеризуется следующим набором параметров:

- именем – указывается в виде глагола и определяет семантику (смысловую подоплеку) связи;

- кратностью (кардинальность, мощность): один-к-одному (1:1), один-ко-многим (1:N) и многие-ко-многим (N:M, N = M или N <> M). Кратность показывает, какое количество экземпляров одной сущности определяется экземпляром другой. Например, на одном участке (описывается строкой таблицы «Участки») может быть один, два и более путей (каждый путь описывается отдельной строкой в таблице «Пути»). В данном случае связь 1:N. Другой пример: один путь проходит через несколько раздельных пунктов и через один раздельный пункт может проходить несколько путей – связь N:M;

- типом: идентифицирующая (атрибуты одной сущности, называемые внешним ключом, входят в состав дочерней и служат для идентификации ее экземпляров, то есть входят в ее первичный ключ) и неидентифицирующая (внешний ключ имеется в дочерней сущности, но не входит в состав первичного ключа);

- обязательностью: обязательная (при вводе нового экземпляра в дочернюю сущность заполнение атрибутов внешнего ключа обязательно и введенные значения должны совпадать со значениями атрибутов первичного ключа какого-либо экземпляра родительской сущности) и необязательная (заполнение атрибутов внешнего ключа в экземпляре дочерней сущности необязательно или введенные значения не совпадают со значениями атрибутов первичного ключа ни одного экземпляра родительской сущности);

- степенью участия – количеством сущностей, участвующих в связи. В основном между сущностями существуют бинарные связи, т.е. ассоциации, связывающие две сущности (степень участия равна 2). В то же время по степени участия возможны следующие типы связей:

  - унарная (рекурсивная) – сущность может быть связана сама с собой. Например, в таблице «Работники» могут быть записи и по

подчиненным, и по их начальникам. Тогда возможна связь «начальник» – «подчиненный», определенная на одной таблице;

- тернарная – связывает три сущности. Например, «Студент» на «Сессии» получил «Оценку по дисциплине»;
- кватернарная и т.д.

В методологии IDEF1X степень участия может быть только унарной или бинарной. Связи большей степени приводятся к бинарному виду.

Внешний вид связи на диаграммах IDEF1X указывает на ее мощность, тип и обязательность.

Таблица 2 - Типы связей

Внешний вид	Тип и обязательность связи	Мощность связи справа
	Обязательная, идентифицирующая	1
	Обязательная, идентифицирующая	0 .. ∞
Z	Обязательная, идентифицирующая	0 или 1
P	Обязательная, идентифицирующая	1 .. ∞
<число>	Обязательная, идентифицирующая	<число>
	Обязательная, неидентифицирующая	0 .. ∞
	Необязательная, неидентифицирующая	0 .. ∞

4. Определение суперклассов и подклассов. В тех случаях, когда две и более сущностей по набору атрибутов незначительно отличаются друг от друга, можно применять в модели конструкцию – иерархию наследования (категорий), включающую в себя суперклассы и подклассы.

Суперкласс – сущность, включающая в себя подклассы.

Иерархия наследования представляет собой особый тип объединения сущностей, которые разделяют общие характеристики. Например, в

организации работают служащие, занятые полный рабочий день (постоянные служащие) и совместители. Из их общих свойств можно сформировать обобщенную сущность (родового предка) «Сотрудник» (рис. 8), чтобы представить информацию, общую для всех типов служащих. Специфическая для каждого типа информация может быть расположена в дополнительных сущностях (потомках) «Постоянный сотрудник» и «Совместитель» [33,34].

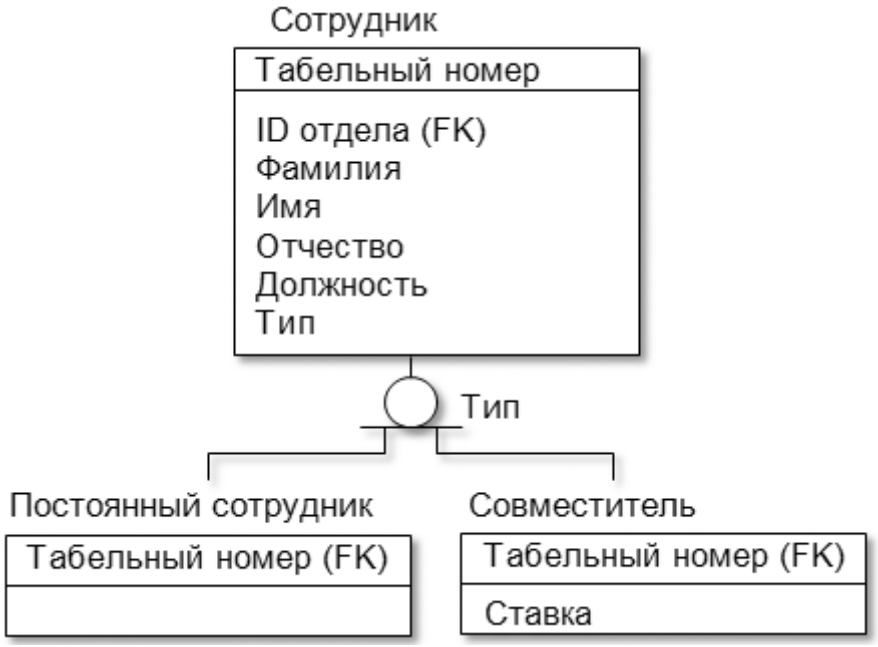


Рисунок 8 - Иерархия наследования (неполная категория)

Обычно иерархию наследования создают, когда несколько сущностей имеют общие по смыслу атрибуты, либо, когда сущности имеют общие по смыслу связи (например, если бы «Постоянный сотрудник» и «Совместитель» имели бы сходную по смыслу связь «работает в» с сущностью «Организация»).

Для каждой категории требуется указать *дискриминатор* – атрибут родового предка, который показывает, как отличить одну сущность от другой. В приведенном примере дискриминатор – атрибут «Тип».

Иерархии категорий делятся на два типа: неполные (рис. 8) и полные (рис. 9).

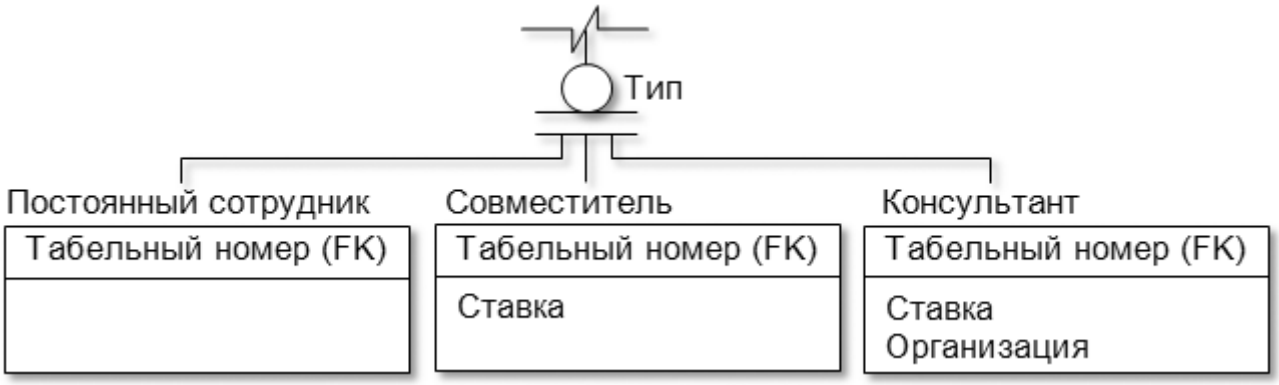


Рисунок 9 - Иерархия наследования (полная категория)

В *полной категории* одному экземпляру родового предка обязательно соответствует экземпляр в каком-либо потомке, то есть. в примере сотрудник обязательно является либо совместителем, либо консультантом, либо постоянным сотрудником.

Если категория еще не выстроена полностью и в родовом предке могут существовать экземпляры, которые не имеют соответствующих экземпляров в потомках, то такая категория будет *неполной*.

При построении модели возможны различные комбинации полных и неполных категорий. Например, первый уровень категории неполный, отдельные сущности которого дополняются вторым уровнем – полной категорией.

4. ***IDEF3*** – нотация моделирования и стандарт документирования процессов, происходящих в системе. Метод документирования технологических процессов представляет собой механизм документирования и сбора информации о процессах. IDEF3 показывает причинно-следственные связи между ситуациями и событиями в понятной эксперту форме, используя структурный метод выражения знаний о том, как функционирует система, процесс или предприятие.

В рамках стандарта IDEF3 выделяют два типа диаграмм, позволяющих описать процесс с разных точек зрения:

- • диаграмма описания последовательности этапов процесса (Process Flow Description Diagrams — PFDD), с помощью которой

моделируется последовательность действий, реализуемых в рамках бизнес-процесса;

- • диаграмма состояния и трансформации объекта в процессе (Object State Transition Network — OSTN), с помощью которой описываются изменения, происходящие с объектом в ходе его обработки.

Для описания и моделирования бизнес-процессов, где основной задачей стоит описание последовательностей действий, которые необходимо выполнить для достижения поставленных целей, больший интерес представляют диаграммы типа PFDD. Рассмотрим его подробнее.

Основными элементами диаграммы PFDD IDEF3 являются:

- • функциональный элемент;
- • стрелка;
- • перекресток.

Функциональный элемент (элемент поведения, единица работы) используется для обозначения действия, работы или события. Он отражается в виде прямоугольника, в центре которого указывается название действия (глагол или отглагольное существительное). Внизу блока указывается номер действия с учетом номера родительской диаграммы (рис. 10).

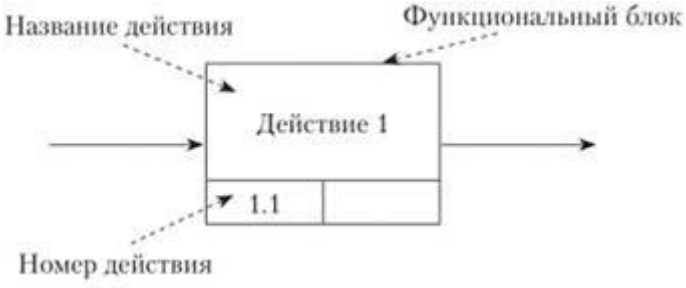


Рисунок 10 - Структура функционального элемента в IDEF3

*Стрелка (линия***)** используется для отражения последовательности выполнения работ (действий) и связей между ними. Все стрелки показывают движение в одну сторону: слева направо, таким образом, визуально соблюдая идею демонстрации последовательного выполнения операций процесса. Они могут выходить и входить с любой стороны блока, но предпочтение лучше

отдавать их горизонтальному расположению. Существуют три типа стрелок (рис. 11): временное предшествование, объектный поток, нечеткое отношение.

Рисунок 11 - Типы стрелок в нотации IDEF3

Стрелка типа «*Временное предшествование» показывает*, что действие, из которого она выходит, должно завершиться до того, как начнется действие, в которое она входит. Результат исходного действия не обязательно является инициатором для действия, куда входит стрелка. Главное значение данной стрелки — показать временную связь между действиями, т.е. показать, что одно действие не может начаться до того, пока предыдущее не закончится, независимо от результата его завершения. Такая связь обозначается простой стрелкой.

Стрелка типа *"Объектный поток"* показывает, что результат действия, из которого она выходит, является инициатором действия, в которое оно входит. Соответственно действие, в которое входит стрелка, не может начаться до тех пор, пока не закончится действие, из которого стрелка выходит. Такая связь обозначается стрелкой с двойным наконечником. В названии стрелки должно быть приведено название объекта, который передается от одной операции к другой.

Стрелка типа *"Нечеткое отношение"* показывает, что тип связи между двумя действиями задается индивидуально, может иметь переменчивый или уникальный характер. Такая связь обозначается пунктирной стрелкой.

специальных требований по ее наименованию нет. Такое изображение связей используется, когда нельзя применить связи, типа "Временное предшествование" и "Объектный поток".

Перекресток (условные символы ветвления) используется для отражения логики движения потоков между функциональными элементами (операциями). Перекресток позволяет указать события, которые могут или должны произойти для того, чтобы началось выполнение следующего действия. На диаграмме IDEF3 перекресток представляет собой прямоугольник с индикатором "J" и номером данного перекрестка на диаграмме (рис. 12). Существуют перекрестки, используемые для отражения слияния стрелок, и перекрестки, используемые для отражения разветвления стрелок. Стоит отметить, что один перекресток не может одновременно использоваться для слияния и для разветвления. В методологии IDEF3 выделяют: *разворачивающиеся соединения*, используемые для отражения связей, где завершение одного процесса инициирует запуск нескольких других процессов: *сворачивающиеся соединения*, используемые для отражения связей, где завершение нескольких процессов приводит к запуску следующего одного процесса.

Разворачивающиеся и сворачивающиеся соединения могут быть также нескольких типов:

- • "и" (обозначается квадратом с символом "&");
- • "исключающее “или”" (обозначается квадратом с символом "X");
- • "или" (обозначается квадратом с символом "O").

На рис. 12 приведен образец построения ШЕЕЗ-диаграммы.

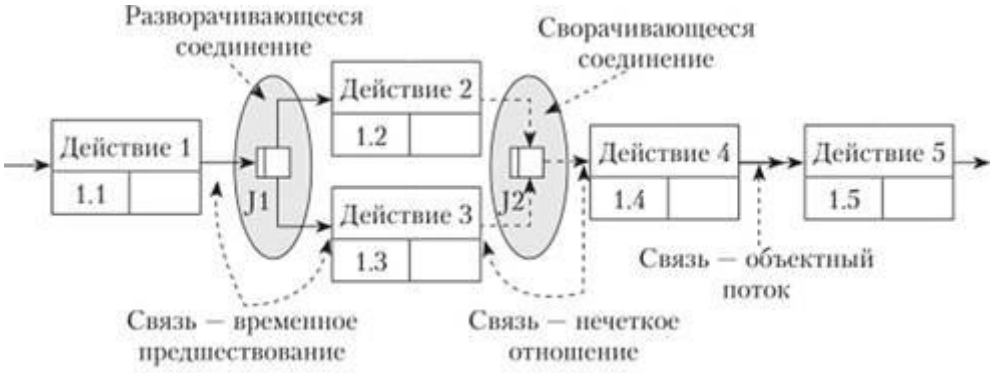


Рисунок 12 - Образец диаграммы в нотации IDEF3

Соединение типа **"и"** используется для описания ситуаций, когда:

- только после завершения нескольких действий может наступить следующее действие;
- после завершения действия одновременно запускаются несколько следующих действий.

Следует учитывать, что если соединение "и" инициирует выполнение последнего действия, то все действия, которые присоединяются к сворачиваемому соединению типа "и" должны быть выполнены полностью.

Например, процесс "Подготовка к продаже нового изделия" состоит из следующих подпроцессов (рис. 13):

- 1.1. Подготовка приказа о вводе в ассортимент нового продукта.
- 1.2. Закупка материалов для производства изделия.
- 1.3. Подготовка технической документации по изготовлению нового изделия.
- 1.4. Подготовка информационных материалов для продвижения и продажи.
- 1.5. Обучение производственного персонала изготовлению нового изделия.
- 1.6. Производство опытной партии нового изделия.

Процессы "Закупка материалов для производства изделия", "Подготовка информационных материалов для продвижения и продажи" и "Подготовка технической документации по изготовлению изделия" начинаются сразу после того, как выпущен приказ о вводе в ассортимент нового продукта. Процесс "Производство опытной партии нового изделия" может начаться только после того, как обучен производственный персонал и закуплен материал для производства.

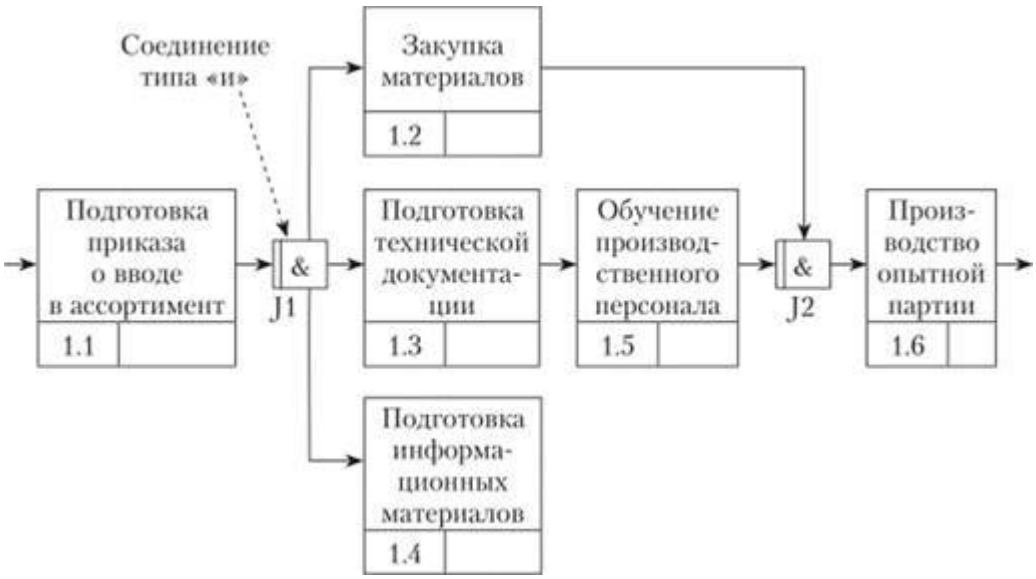


Рисунок 13 - IDEF3-диаграмма процесса "Подготовка к продаже нового изделия"

Соединение типа **"исключающее “или”"** используется для описания ситуаций, когда:

- • после завершения одного действия может начаться только одно из следующих действий;
- • следующее действие может начаться после завершения только одного из предыдущих действий.

Например, соединение "исключающее “или”" используется для того, чтобы показать, что результатом согласования проекта договора может быть: а) проект договора согласован; б) по проекту договора есть замечания и он отправлен на доработку (рис. 14). В первом случае, если он согласован, то осуществляется следующее действие — подписание договора. Во втором случае, когда по нему есть замечания, осуществляется его доработка. Здесь, "исключающее “или”" показывает, что в зависимости от результата выполнения первого действия потом будет выполняться второе или третье действие.

При использовании такого типа соединения целесообразно подписывать стрелки или делать комментарии к ним, показывая в каком случае, какое действие будет выполняться.

Рисунок 14 - Фрагмент IDEF3-диаграммы процесса "Управление договором"

Соединение типа "или" используется для описания ситуаций, при которых:

- • после завершения одного или нескольких предшествующих действий может наступить следующее действие;
- • после завершения одного действия может начаться одно или несколько следующих действий.

Примером использования такого типа соединения может служить фрагмент процесса заполнения анкеты, представленный на рис. 15.

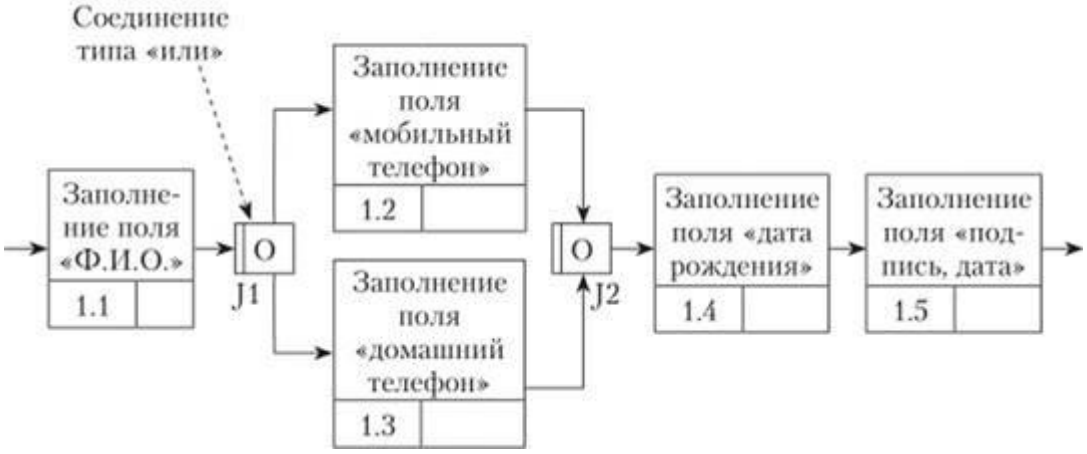


Рисунок 15 - Фрагмент IDEF3-диаграммы процесса "Заполнение анкеты"

На рис. 15 показан пример использования соединения типа "или", где после действия "Заполнение поля “Ф.И.О.”" может быть выполнено действие "Заполнение поля “мобильный телефон”" или действие "Заполнение поля “домашний телефон”" либо оба эти действия. Одно из них точно должно быть выполнено.

Таким образом, можно выделить пять типов перекрестков. Каждый из них имеет свое обозначение. В таблице 3 приведено краткое описание всех типов перекрестков.

В приведенных выше примерах IDEF3-диаграмм используются асинхронные типы перекрестков, поскольку на практике они встречаются чаще, нежели синхронные.

Процессы, описанные с помощью IDEF3-диаграмм, могут быть также декомпозированы для более детального анализа.

Модели в нотации IDEF0 могут быть декомпозированы в виде IDEF0- и IDEF3-диаграмм, а модели IDEF3 могут быть декомпозированы только в виде IDEF3-диаграмм.

Используя диаграмму процесса в нотации IDEF0 в качестве родительской диаграммы, можно построить дочерние для ее функциональных блоков модели в нотации IDEF3. При нумерации функциональных элементов IDEF3-диаграмм необходимо учитывать номера функциональных блоков родительской IDEFO-диаграммы. Здесь работает правило декомпозиции методологии SADT.

Таблица 3 - Типы перекрестков в нотации IDEF3

Символ		Название	Соединение разворачивается	Соединение сворачивается
1	&	Асинхронное "и"	Все следующие процессы должны начаться	Все предшествующие процессы должны быть завершены
2	&	Синхронное "и"	Все следующие процессы должны	Все предшествующие процессы должны

			начаться одновременно	завершиться одновременно
3	O	Асинхронное "или"	Один или несколько процессов должны начаться	Один процесс или несколько предыдущих должны быть завершены
4	O	Синхронное "или"	Один или несколько процессов должны одновременно начаться	Один или несколько предыдущих процессов должны быть завершены одновременно
5	X	Исключающее "или"	Только один следующий процесс должен начаться	Только один предшествующий процесс может быть завершен

Однако стоит учитывать, что модели IDEF3 могут быть декомпозированы только в виде IDEF3-диаграмм.

Помимо этого, в настоящий момент к семейству IDEF можно также отнести следующие стандарты, которые менее популярны:

5. IDEF2. Simulation Model Design — методология динамического моделирования развития систем. В связи с весьма серьёзными сложностями анализа динамических систем от этого стандарта практически отказались, и его развитие приостановилось на самом начальном этапе. В настоящее время присутствуют алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе «раскрашенных сетей Петри» (CPN — Color Petri Nets).

6. IDEF4. Object-Oriented Design — методология построения объектно-ориентированных систем, позволяет отображать структуру объектов и заложенные принципы их взаимодействия и тем самым анализировать и оптимизировать сложные объектно-ориентированные системы.

7. IDEF5. Ontology Description Capture — Стандарт онтологического исследования сложных систем. С помощью методологии IDEF5 онтология системы может быть описана при помощи определённого словаря терминов и правил, на основании которых могут быть сформированы достоверные утверждения о состоянии рассматриваемой системы в некоторый момент

времени. На основе этих утверждений формируются выводы о дальнейшем развитии системы и производится её оптимизация.

8. IDEF6. Design Rationale Capture — Обоснование проектных действий. Назначение IDEF6 состоит в облегчении получения «знаний о способе» моделирования, их представления и использования при разработке систем управления предприятиями. Под «знаниями о способе» понимаются причины, обстоятельства, скрытые мотивы, которые обуславливают выбранные методы моделирования. Проще говоря, «знания о способе» интерпретируются как ответ на вопрос: «Почему модель получилась такой, какой получилась?» Большинство методов моделирования фокусируются на собственно получаемых моделях, а не на процессе их создания. Метод IDEF6 акцентирует внимание именно на процессе создания модели.

9. IDEF7. Information System Auditing — Аудит информационных систем. Этот метод определён как востребованный, однако так и не был полностью разработан.

10. IDEF8. User Interface Modeling — Метод разработки интерфейсов взаимодействия оператора и системы (пользовательских интерфейсов). Современные среды разработки пользовательских интерфейсов в большей степени создают внешний вид интерфейса. IDEF8 фокусирует внимание разработчиков интерфейса на программировании желаемого взаимного поведения интерфейса и пользователя на трёх уровнях: выполняемой операции (что это за операция); сценарии взаимодействия, определяемом специфической ролью пользователя (по какому сценарию она должна выполняться тем или иным пользователем); и, наконец, на деталях интерфейса (какие элементы управления, предлагает интерфейс для выполнения операции).

11. IDEF9. Scenario-Driven IS Design (Business Constraint Discovery method) — Метод исследования бизнес-ограничений был разработан для облегчения обнаружения и анализа ограничений в условиях, в которых действует предприятие. Обычно при построении моделей уделяется

недостаточное внимание описанию ограничений, оказывающих влияние на протекание процессов на предприятии. Знания об основных ограничениях и характере их влияния, закладываемые в модели, в лучшем случае остаются неполными, несогласованными, распределёнными нерационально, но часто их вовсе нет. Это не обязательно приводит к тому, что построенные модели нежизнеспособны, просто их реализация столкнётся с непредвиденными трудностями, в результате чего их потенциал будет не реализован. Тем не менее, в случаях, когда речь идёт именно о совершенствовании структур или адаптации к предсказываемым изменениям, знания о существующих ограничениях имеют критическое значение.

12. IDEF10. Implementation Architecture Modeling — Моделирование архитектуры выполнения.

13. IDEF11. Information Artifact Modeling.

14. IDEF12. Organization Modeling — Организационное моделирование.

15. IDEF13. Three Schema Mapping Design — Трёхсхемное проектирование преобразования данных.

Эти методы определены как востребованные, однако так и не были полностью разработаны.

16. IDEF14. Network Design — Метод проектирования компьютерных сетей, основанный на анализе требований специфических сетевых компонентов существующих конфигураций сетей. Также он обеспечивает поддержку решений, связанных с рациональным управлением материальными ресурсами, что позволяет достичь существенной экономии.

2.2 Методология быстрой разработки приложений RAD

Принципы RAD (Rapid Application Development) сформулированы в 1980 году сотрудником компании IBM Джеймсом Мартином. Они базировались на идеях Скотта Шульца и Барри Бойма при этом методология

реализовывалась в кратчайшие сроки небольшой группой разработчиков с использованием инкрементного прототипирования. Это позволяло на ранней стадии проектирования ИС продемонстрировать заказчику действующую интерактивную модель системы-прототипа, уточнить проектные решения, оценить эксплуатационные характеристики.

Особенностью этой методологии является реализация следующих подходов:

- короткий по времени, но тщательно проработанный график выполнения работ;

- небольшая команда программистов;

- цикличность разработки: на каждом цикле разработчики по мере того как приложение приобретает функциональность, запрашивают заказчика и реализуют в продукте новые требования.

Модель ИС, создаваемая по технологии RAD, позволяет заказчикам после каждой итерации в процессе проектирования видеть текущие возможности ИС и представлять, какие необходимы дополнительные вложения, трудозатраты и сроки, чтобы повысить функциональность системы.

Используемую в модели технологию по аналогии также называют технологией быстрой разработки приложений. Такое название она получила вследствие:

- возможности расширения программных прототипов;

- сокращения числа итераций благодаря поэтапному системному анализу принимаемых решений, выявления и, главное, предупреждения возможных ошибок и несоответствий;

— упрощения создания проектной документации;

- существенного снижения сроков проектирования и разработки ИС, поскольку поощряется использование имеющихся, успешно зарекомендовавших себя однотипных стандартизованных решений, блоков и модулей.

В настоящее время методология RAD стала общепринятой схемой для проектирования и разработки информационных систем. Средства разработки, основанные на RAD, очень популярны за счет использования таких программных сред разработки: IBM Lotus Domino Designer, Borland Delphi, Borland C++ Builder, Microsoft Visual Studio, Macromedia Flash и др.

В методологии RAD быстрая разработка приложений достигается за счет использования компонентно-ориентированного конструирования и применяется если:

- ✓ Бюджет проектируемой информационной системы ограничен.
- ✓ Нечетко определены требования к информационной системе.
- ✓ Требуется реализация проекта информационной системы в минимальные сроки.
- ✓ Интерфейс пользователя можно продемонстрировать в прототипе.
- ✓ Проект можно разделить на составляющие элементы по функциональному назначению.

Методология RAD имеет следующие стадии:

1. Моделирование информационных потоков между бизнес-функциями.
2. Моделирование данных.
3. Преобразование объектов данных, обеспечивающих реализацию бизнес-функций.
4. Генерация приложений.
5. Тестирование и объединение.

Недостатки методологии RAD:

1. Для больших информационных систем требуются большой коллектив разработчиков.
2. Применима для информационных систем, которые могут декомпозироваться на отдельные модули и в которых производительность не является критической величиной.

3. Не используется в случае применения новых технологий.

2.3 Методология ARIS

Основная цель Architecture of Integrated Information Systems (ARIS) — проектирование информационных систем и моделирование бизнес-процессов организаций.

Методология ARIS является мощным инструментом анализа и разработки решений. Позволяет оптимизировать организационную деятельность и структуры предприятия, применима для:

1) описания и управления организационной структурой организации;

2) моделирования и анализа бизнес-процессов организации;

3) формирования отчетов построенных моделей.

Методология ARIS является одним из подходов к формализации представлений о деятельности предприятия и визуализации ее в виде графических моделей, удобных для понимания и анализа. Методология разработана германским профессором А.-В. Шеером и предназначена для анализа и моделирования деятельности предприятия и разработки автоматизированных информационных систем.

В основе методологии ARIS лежит концепция интеграции, предлагающая комплексное описание бизнес-процессов и архитектуры предприятия. Методология позволяет моделировать и описывать практически все компоненты деятельности и архитектуры предприятий, и это отличает ее от других методологий.

Методология ARIS реализует принципы структурного анализа, представляя предприятие в виде большой и сложной системы, компоненты которой имеют различного рода взаимосвязи друг с другом.

В соответствии с ARIS предприятие рассматривается как бизнес-система, включающая организационную, функциональную,

информационную системы, систему целей, средств производства, человеческих ресурсов и т.д.

Такой подход позволяет использовать методологию для комплексного описания (моделирования) деятельности предприятия. Модели ARIS являются основой:

1) для анализа (выявление «узких мест»), оценки стоимости и оптимизации бизнес-процессов;

2) документирования корпоративных знаний предприятия;

3) формулирования требований к разработке и внедрению информационных систем;

4) подготовки предприятия к сертификации по международным стандартам качества.

Методология ARIS рассматривает предприятие с четырех точек зрения, формируя соответствующие взгляды на организационную структуру, структуру функций, структуру данных и структуру процессов предприятия. Соответственно, ARIS поддерживает четыре типа моделей, отражающих различные аспекты архитектуры анализируемой бизнес-системы (предприятия):

- организационные модели, представляющие организационную структуру предприятия — иерархию подразделений, должностей и конкретных сотрудников, различные связи между ними, а также их местонахождение или размещение;

- функциональные модели, содержащие иерархию целей, стоящих перед менеджментом предприятия, а также деревья функций, выполняемых для достижения этих целей;

- информационные модели, отражающие структуру информации, обрабатываемой при реализации бизнес-функций;

- модели управления, интегрирующие организационную структуру с функциями бизнес-процессов и данными, обеспечивающими эти бизнес-процессы.

Графическое представление структуры ARIS приведено на рис. 16 [1].

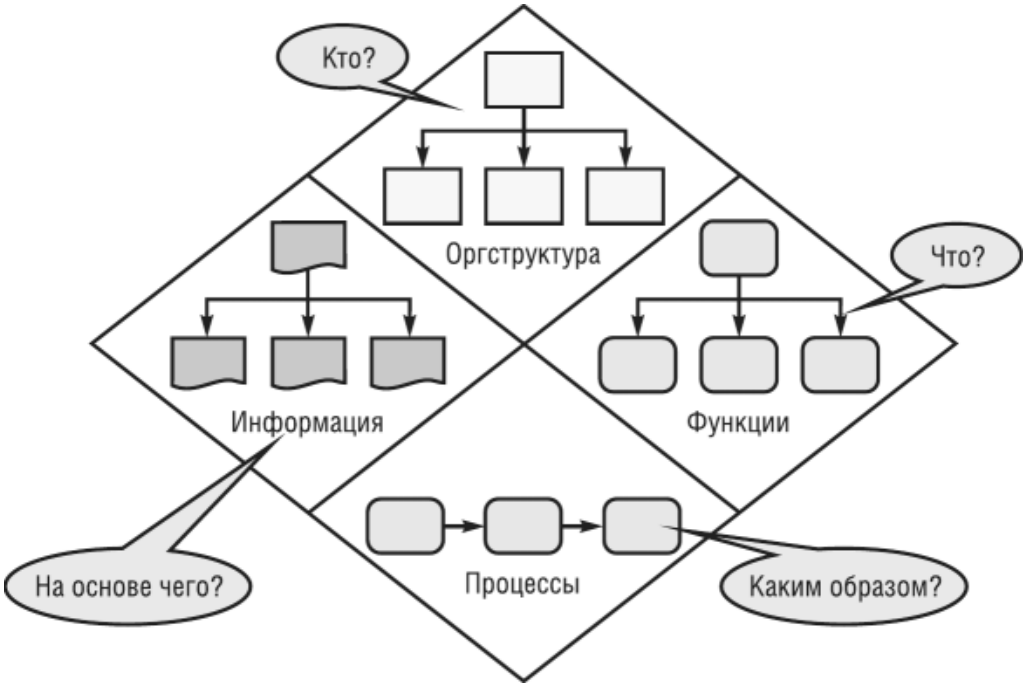


Рисунок 16 - Группы моделей методологии ARIS

В специальной литературе совокупность типов моделей ARIS иногда принято изображать в виде так называемого «здания» ARIS [26, 27] (рис. 17).

Таким образом, относительная независимость организационного, функционального и информационного взглядов позволяет понизить сложность модели и сосредоточиться на отдельных аспектах анализа предприятия. После детальной проработки на интегрированной модели можно исследовать существующие связи между информационными, функциональными и организационными моделями бизнес-процессов.

Рисунок 17 - «Здание» ARIS

В ARIS в качестве основного структурообразующего понятия используется понятие «объект», количество типов которых составляет несколько десятков (например, «функция», «класс», «организационная единица»). Объекты связаны различными типами отношений (например, «выполняет», «является входящим», «занимает позицию»).

При этом каждый тип модели, объекта, связи содержит набор атрибутов (например, «имя», «затраты», «время выполнения», «адрес») Полная версия методологии поддерживает значительное количество моделей (около сотни).

В методологии ARIS модели функций, организации, данных и выходов описывают архитектуру предприятия (декларативные знания). Модели управления отражают структурные связи этих моделей и описывают динамику бизнес-процессов (процедурные знания). При этом модель бизнес-процесса (управления)1 составляет основу методологии ARIS при моделировании организационно-технических систем (предприятий).

В методологии ARIS цвет имеет смысловое значение. Это повышает восприимчивость и читабельность моделей. Например, структурные подразделения по умолчанию изображаются желтым цветом, бизнес-процессы и функции — зеленым.

Следует заметить, что использование множества блоков и связей различных типов загромождает модель, делает ее плохо читаемой и малопонятной. Поэтому в конкретных проектах, как правило, используется ограниченное их число (до десяти моделей). В настоящем пособии рассматриваются только основные модели [26,27].

Составной частью методологии ARIS является нотация EPC (Event-driven Process Chain). Нотация моделирования EPC ориентирована на построение алгоритмов взаимодействия в процессе выполнения конкретной работы. Её главными элементами являются:

- события, которые запускают или завершают работу;
- действия (работа), которая переводит систему из одного состояния в другое;
- исполнители работы;
- ресурсы и результаты работы (входы и выходы).

Розовые фигуры – это события. Событие – это некоторое состояние, которое принимает система и определяет дальнейшее развитие одного или более бизнес-процессов. События могут активизировать функции или порождаться функциями.

Зелёные элементы – функции или функциональные блоки. Функциональный блок – это действие или подпроцесс, выполняемый с целью получения заданного результата и перевода системы из одного состояния в другое. Порядок выполнения функций задается на диаграмме сверху-вниз.

Также хорошо улавливается информация об исполнителях каждой работы, которые отображаются с помощью желтых овалов. Этот элемент называется «Субъект», он обозначает исполнителей, владельцев или участников. В качестве субъектов могут быть прикреплены должности исполнителей, подразделения и роли.

Помимо названных фигур на диаграмме используются серые прямоугольники или похожие на них фигуры, которые обозначают все многообразие используемых и создаваемых ресурсов и результатов в

процессе. Если мы захотим отобразить передачу денег, то также должны будем воспользоваться серым прямоугольником (рис.18).

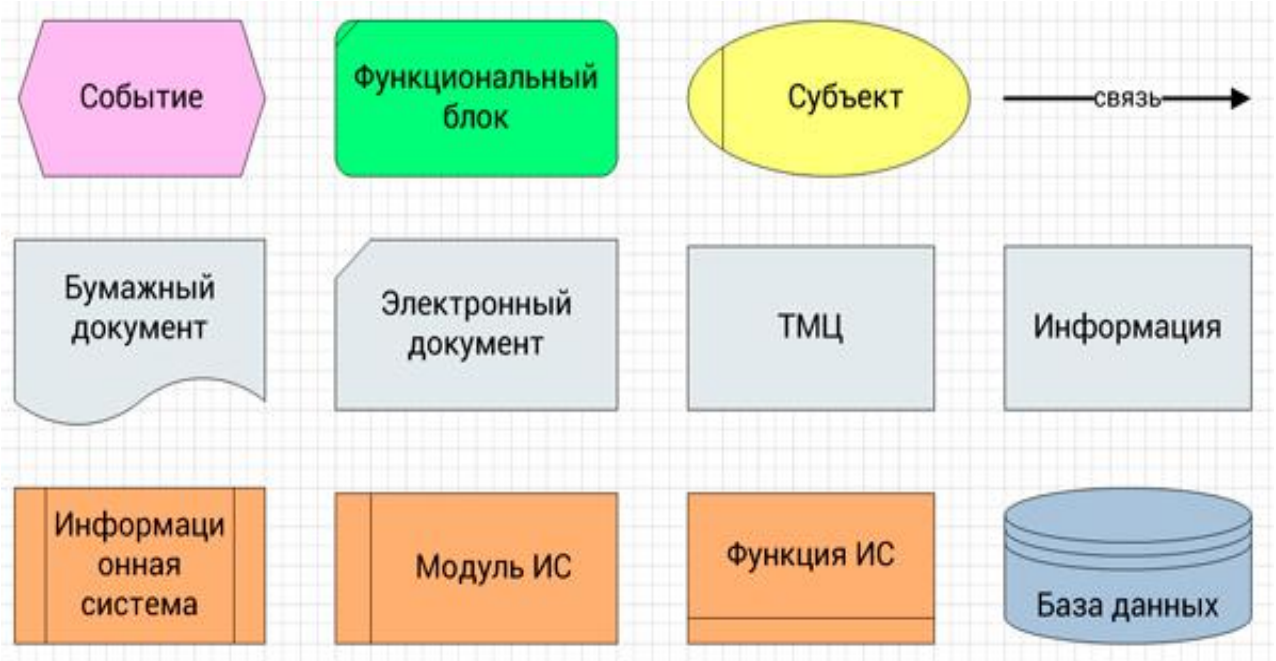


Рисунок 18 - Основные элементы нотации EPC

Для отображения логики переходов между функциями используются логические операторы, которые помогают конкретизировать условия выполнения параллельных работ или возникновения событий. Они показывают варианты слияния или ветвления как функций, так и событий. Логических операторов всего три: «И», «ИЛИ» и «Исключающее ИЛИ». В разных системах могут использоваться их разные графические обозначения (таблица 4).

Таблица 4 - Обозначение и использование логических операторов в нотации EPC

Оператор AND ("И") 	Используется если: • завершение выполнения функции должно инициировать одновременно несколько событий; • событие происходит только после обязательного завершения выполнения нескольких функций; • функция может начать выполняться только после того, как произойдут несколько событий; • одно событие инициирует одновременное выполнение нескольких функций.
Оператор OR ("ИЛИ") 	Используется если: • завершение выполнения функции может инициировать одно или несколько событий; • событие происходит после завершения выполнения одной или нескольких функций; • функция может начать выполняться после того, как произойдет одно или несколько событий.
Оператор XOR ("Исключающее ИЛИ") 	Используется если: • завершение выполнения функции может инициировать только одно из событий в зависимости от условия; • событие происходит сразу после завершения выполнения либо одной, либо другой функции; • функция может начать выполняться сразу после того, как произойдет либо одно событие, либо другое.

Несомненным преимуществом данной нотации является интуитивно понятный набор элементов и правил построения диаграмм. Для создания диаграмм процессов рекомендуется использовать специальные программы для моделирования процессов. Для EPC это, в первую очередь, система ARIS. Но она дорогая и достаточно сложная, и для небольших периодических проектов по упорядочиванию деятельности подразделений не используется.

Простота и популярность нотации стимулировало создание других инструментов для рисования бизнес-процессов, в том числе в нотации EPC. Самым простым из них является Visio – один из шаблонов в нем так и называется – «Схема EPC». Наиболее полезным инструментом для меня является система Бизнес Студио. В ней, кроме возможности нарисовать процесс, можно автоматически сгенерировать документ (Регламент процесса)

и рабочие инструкции для его участников, что заметно облегчает рутинную часть процесса разработки стандартов деятельности (рис.19).

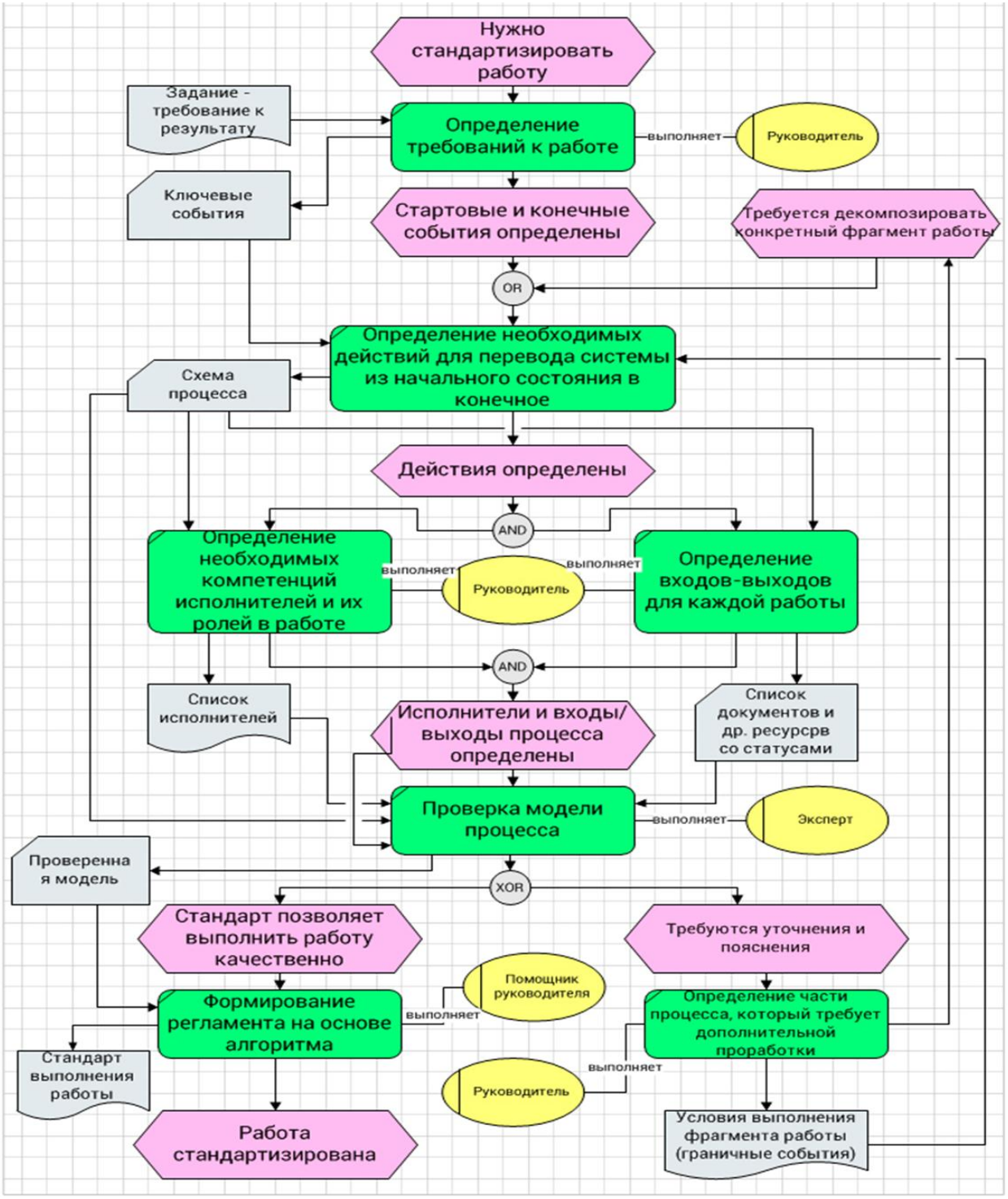


Рисунок 19 - Пример применения нотации EPC

Локальные рабочие процедуры, нарисованные в данной нотации, вполне удобны как для разработчика, так и для пользователя инструкции.

Нотация подойдет и для проект-менеджеров, поскольку позволяет им наглядно планировать распределение работ в проекте на интуитивно понятном для разных участников проекта языке. А для разработки более сложной многоуровневой модели деятельности предприятия больше подходят другие нотации моделирования.

2.4 Методология BPMN

Модель и нотация бизнес-процессов (BPMN, Business Process Model and Notation) – методология моделирования, анализа и реорганизации бизнес-процессов. Разработана Business Process Management Initiative (BPMI), с 2005 г. поддерживается и развивается Object Management Group (OMG). В отличие от других методологий бизнес-моделирования, имеющих статус «фирменного» (EPC) или «национального» (IDEF0) стандарта, BPMN получила «международный» статус – Международная организация по стандартизации опубликовала стандарт «ISO/IEC 19510:2013. Information technology - Object Management Group. Business Process Model and Notation».

Основной целью BPMN является обеспечение доступной нотацией описания бизнес-процессов всех пользователей: от аналитиков, создающих схемы процессов, и разработчиков, ответственных за внедрение технологий выполнения бизнес-процессов, до руководителей и обычных пользователей, управляющих этими бизнес-процессами и отслеживающих их выполнение. Таким образом, BPMN нацелен на устранение расхождения между моделями бизнес-процессов и их реализацией [38].

По заявлению разработчиков стандарта BPMN, он вобрал в себя лучшие идеи, что имеются в следующих нотациях и методологиях моделирования:

- UML (Unified Modeling Language, Унифицированный язык моделирования);

- IDEF (SADT);

- ebXML (Electronic Business eXtensible Markup Language, расширяемый язык разметки для электронного бизнеса);

- BPSS (Business Process Specification Schema, схемы спецификации бизнес-процессов);

- ADF (Activity-Decision Flow, поток «деятельность-результат») Diagram;

- RosettaNet;

- LOVEM (Line of Visibility Engineering Methodology, визуальная методология проектирования);

- EPC.

В стандарте также отмечается, что для большей читабельности и гибкости в методологии BPMN продолжены традиции блок-схем.

Поддержка и дальнейшее развитие BPMN организацией OMG наложило свой «отпечаток» на данную методологию. Одним из ключевых направлений OMG является продвижение UML, предназначенного для моделирования объектно-ориентированных систем. В связи с этим, в BPMN при моделировании (разработке диаграмм), помимо понятий и концепций структурного подхода (действие, поток управления, объект данных и т.д.), используются такие характерные для объектно-ориентированного подхода понятия, как сообщение, обмен сообщениями и поток сообщений.

Элементы (символы) графической нотации BPMN по назначению объединены в категории:

- объекты потока (Flow Objects);

- данные (Data);

- зоны ответственности (Swimlanes);

- соединяющие элементы (Connecting Objects);

- артефакты (Artifacts).

В бизнес-моделировании процессы можно условно разделить на два вида — исполняемые, которые действительно будут работать при помощи специального обеспечения, например, Bizagi, и неисполняемые, т.е.бизнес-модели, необходимые только для изучения и демонстрации вариантов работы предприятия.

В принципе, между их построением нет особой разницы, здесь важен исключительно желаемый результат. Либо бизнес-модель будет применяться только для облегчения взаимопонимания между заказчиком (руководителем) и консультантом (исполнителем). Либо эта нотация будет впоследствии использоваться в какой-либо программной среде для организации работы компании. В обычных руководствах вы этого разделения на две части не найдете. Но я лично считаю, что имеет смысл условно так делить бизнес-процессы, так как при различном желаемом результате потребуется различная глубина проработки деталей и выбор возможных инструментов для работы.

Исполняемые бизнес-процессы обязательно должны быть выстроены в строгом соответствие всем правилам нотации BPMN, так как в противном случае программное обеспечение не сможет работать корректно с составленной бизнес-моделью. Исполняемые процессы нужны, например, на предприятиях, где принят процессный подход к деятельности. Программное обеспечение позволяет вести контроль всех процессов в режиме реального времени, и на основе получаемых на каждом из этапов данных, руководитель компании и подразделений всегда смогут понимать, на каком этапе находится работа по тому или иному процессу. Подобный метод позволяет значительно повысить эффективность управления.

Неисполняемые бизнес-процессы нужны исключительно для демонстрации какой-либо бизнес-модели. Это может быть диаграмма, отображающая реальное положение дел на предприятии, может быть наглядной иллюстрацией к предложенным изменениям при реинжиниринге. В этом случае, конечно, можно использовать любые удобные инструменты, в

том числе, традиционный для многих IDEF0. А соблюдение правил языка моделирование необходимо исключительно для достижения взаимопонимания.

На начальном этапе работы с BPMN рекомендуется создавать неисполняемые бизнес-процессы. Это действительно очень удобная нотация для того, чтобы иллюстрировать свои идеи и предложения, демонстрировать «узкие места» в бизнесе, даже просто для себя разбираться в структуре работы той или иной компании очень удобно с использованием нотаций. Наглядная графика и строгие правила в этом очень помогают.

Исполняемый вариант требует глубоких знаний BPMN, а также внимательного отношения к каждой детали, так как, по сути, создается программа (алгоритм).

2.5 Методология Scrum

Scrum («скрам», от англ. — толкучка) — это методология управления проектами, которая применяется при проектировании ИС и позволяет реализовать гибкую технологию разработки ПО. Scrum представляет собой набор подходов, на основе использования которых реализуется процесс разработки, позволяющий в фиксированные и достаточно короткие по времени итерации — спринты (sprints) предоставлять пользователю работающее ПО с реализованными новыми возможностями. Характеристики возможностей ПО, реализуемые в очередном спринте, устанавливаются в начале спринта на этапе планирования. Эти предполагаемые возможности не могут изменяться на всем протяжении реализации итерации. Гибкость процесса разработки становится возможной благодаря строго фиксированной относительно короткой длительности спринта. Благодаря такому подходу Scrum позволяет выполнять качественный контроль процесса разработки.

Scrum может применяться не только для управления проектами по разработке ПО, но также в работе команд, обеспечивающих поддержку ПО, или как подход управления разработкой и сопровождением программ.

Одна из причин популярности методики — простота. Основные используемые инструменты: bumdown (диаграмма), backlog (истории пользователей).

Основное назначение диаграммы — отображать фактическую оставшуюся трудоемкость по задачам итерации и сопоставлять ее с идеальной оставшейся трудоемкостью.

Основные термины, которые используются в методологии:

Владелец продукта (Product owner) — человек, который имеет непосредственный интерес в качественном конечном продукте, он понимает, как это продукт должен выглядеть/работать. Этот человек не работает в команде, он работает на стороне заказчика/клиента (это может быть, как другая компания, так и другой отдел), но этот человек работает с командой. И это тот человек, который расставляет приоритеты для задач.

Scrum-мастер — это человек, которого можно назвать руководителем проекта, хотя это не совсем так. Главное, что это человек, «зараженный Scrum-бациллой» на столько, что несет ее как своей команде, так и заказчику, и соответственно следит за тем, чтобы все принципы Scrum соблюдались.

Scrum-команда — это команда, которая принимает все принципы Scrum и готова с ними работать.

Спринт — отрезок времени, который берется для выполнения определенного (ограниченного) списка задач. Рекомендуется брать 2-4 недели (длительность определяется командой один раз).

Бэклог (backlog) — это список всех работ. Различают 2 вида бэклогов: Product-бэклог и спринт-бэклог.

Product-бэклог — это полный список всех работ, при реализации которых мы получим конечный продукт.

Спринт-бэклог — это список работ, который определила команда и согласовала с Владельцем продукта, на ближайший отчетный период (спринт). Задания в спринт-бэклог берутся из product-бэклога.

Планирование спринта — это совещание, на котором присутствуют все (команда, Scrum-мастер, Владелец продукта). В течение этого совещания Владелец продукта определяет приоритеты заданий, которые он хотел бы увидеть выполненными по истечении спринта. Команда оценивает по времени, сколько из желаемого они могут выполнить. В итоге получается список заданий, который не может меняться в течение спринта и к концу спринта должен быть полностью выполнен.

2.6 Методология DSDM

DSDM (Dynamic Systems Development Method, динамический метод разработки систем) представляет собой методику разработки ПО, основанную на концепции быстрой разработки приложений (RAD). Начиная с 2007 г. DSDM стал одним из признанных подходов к проектированию и созданию приложений. DSDM реализует итеративный и инкрементный подход, который предполагает активное участие в процессе проектирования и разработки конечного пользователя.

Цель метода — сдать готовый проект вовремя и уложиться в бюджет, но в то же время регулируя изменения требований к проекту во время его разработки. DSDM входит в семейство гибкой методологии разработки программного обеспечения, а также разработок, не входящих в сферу информационных технологий.

Существует 9 принципов, состоящих из 4 основных и 5 начальных точек.

1. Вовлечение пользователя - это основа ведения эффективного проекта, где разработчики делят с пользователями рабочее пространство и поэтому принимаемые решения будут более точными.

2. Команда должна быть уполномочена принимать важные для проекта решения без согласования с начальством.

3. Частая поставка версий результата, с учётом такого правила, что «поставить что-то хорошее раньше - это всегда лучше, чем поставить всё идеально сделанное в конце». Анализ поставок версий с предыдущей итерации учитывается на последующей.

4. Главный критерий - как можно более быстрая поставка программного обеспечения, которое удовлетворяет текущим потребностям рынка. Но в то же время поставка продукта, который удовлетворяет потребностям рынка, менее важна, чем решение критических проблем в функционале продукта.

5. Разработка - итеративная и инкрементальная. Она основывается на обратной связи с пользователем, чтобы достичь оптимального с экономической точки зрения решения.

6. Любые изменения во время разработки - обратимы.

7. Требования устанавливаются на высоком уровне прежде, чем начнётся проект.

8. Тестирование интегрировано в жизненный цикл разработки.

9. Взаимодействие и сотрудничество между всеми участниками необходимо для его эффективности.

Чтобы успешно использовать DSDM, необходимо чтобы был выполнен ряд предпосылок. Во-первых, необходимо организовать взаимодействие между проектной командой, будущими пользователями и высшим руководством. Во-вторых, должна присутствовать возможность разбиения проекта на меньшие части, что позволит использовать итеративный подход.

Два примера проектов, для которых DSDM не очень подходит:

- проекты, критичные по безопасности - расширенное тестирование и утверждение в таких проектах конфликтуют с целью метода DSDM уложиться в сроки и в бюджет.

- проекты, чья цель произвести компоненты многоразового использования - требования в таких проектах слишком высоки и не укладываются в принцип 20/80.

Фреймворк DSDM состоит из трёх последовательных стадий, которые называются предпроектная стадия, стадия жизненного цикла проекта и постпроектная стадия. Стадия жизненного цикла проекта - самая продуманная и детально разработанная из всех остальных. Она состоит из пяти этапов, которые формируют итеративный, инкрементный подход к разработке информационных систем. Эти три фазы и соответствующие этапы будут более подробно описаны в последующих разделах. Для каждой стадии или этапа будут рассмотрены самые важные функции и будут представлены результаты.

Стадия 1 – Предпроектная. На этой стадии определяются вероятные проекты, происходит выделение средств и определение проектной команды. Решение задач на этой стадии поможет избежать проблем на более поздних стадиях проекта.

Стадия 2 - Жизненный цикл проекта. На рисунке 20 изображена данная стадия. На нём показаны 5 этапов, которые нужно пройти проекту, чтобы стать информационной системой. Первые два этапа, исследование реализуемости и исследование экономической целесообразности, идут последовательно и дополняют друг друга. После завершения этих этапов, происходит итеративная и инкрементная разработка системы в этапах: создание функциональной модели, проектирование и разработка, этап реализации. Итеративная и инкрементная природа DSDM будет описана далее.

Стадия 3 – Постпроектная. На этой стадии обеспечивается эффективная работа системы. Это достигается за счёт поддержания проекта, его улучшения и исправления ошибок согласно принципам DSDM. Поддержка проекта осуществляется продолжением разработки, основанной на итеративной и инкрементной природе DSDM. Вместо того, чтобы

закончить проект за один цикл, обычно возвращаются к предыдущим стадиям или этапам, чтобы улучшить продукт.

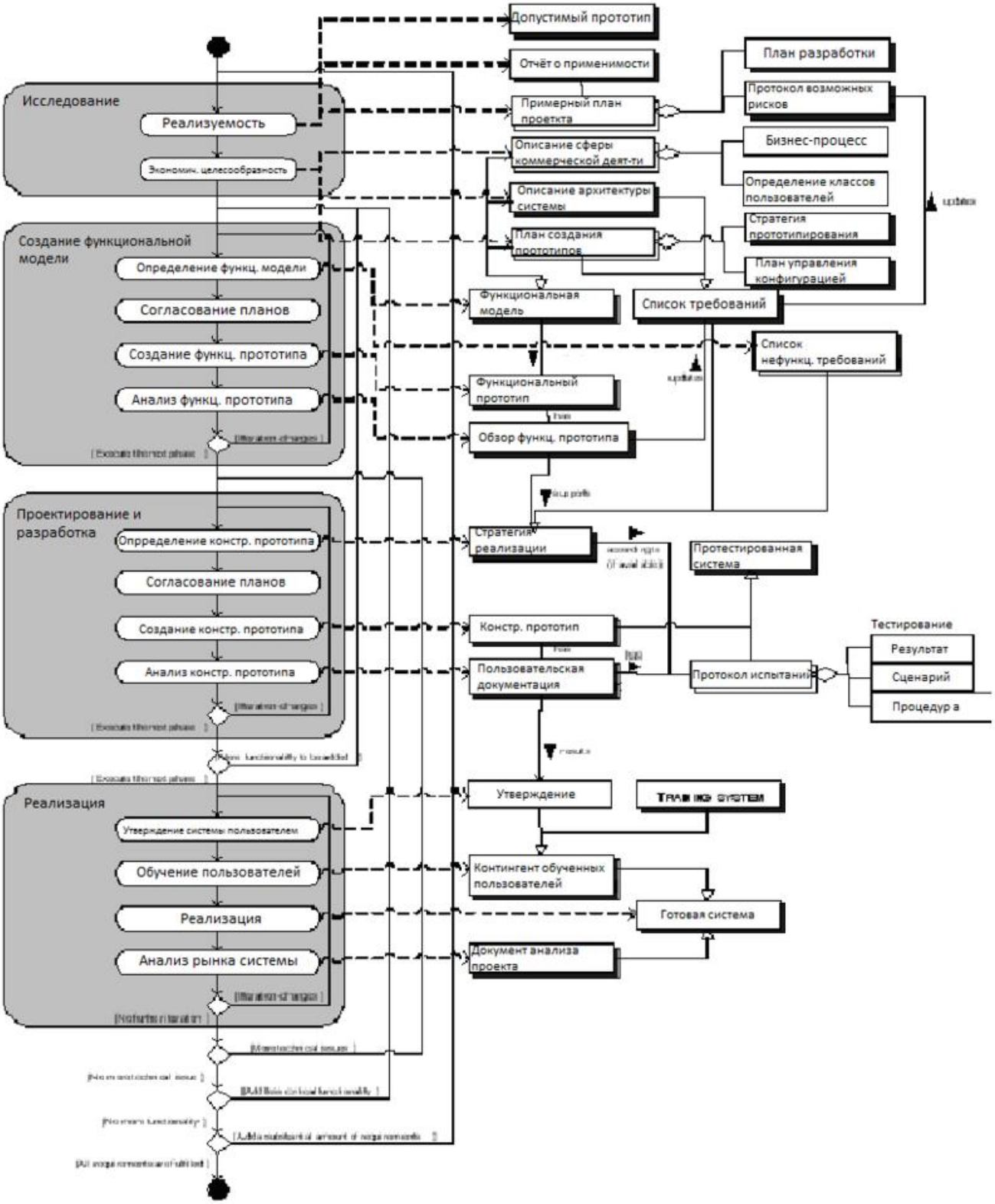


Рисунок 20 - Диаграмма жизненного цикла проекта

2.7 Методология extreme Programming

extreme Programming (экстремальное программирование) — методология гибкой разработки, ориентированная на использование таких практик как TDD (Test Driven Development, разработка через тестирование), парное программирование, непрерывная интеграция, рефакторинг, коллективное владение кодом (включая парное программирование), описание требуемой функциональности при помощи историй пользователей и др. Эта методология использует практику разработки программ, при которой вначале пишется тестовый код, а затем его реализация, при этом постоянно выполняется проверка работоспособности программного кода, с использованием написанных тестов. Эта практика позволяет создавать автоматически тестируемый код, безболезненно проводить рефакторинг и свободнее менять архитектуру или реализацию разрабатываемого приложения.

Автор методики — американский разработчик Кент Бек. В конце 90-х годов он руководил проектом Chrysler Comprehensive Compensation System и там впервые применил практики экстремального программирования. Свой опыт и созданную концепцию он описал в книге Extreme Programming Explained, опубликованной в 1999 году. За ней были выпущены другие книги, в которых подробно описывались практики XP. К становлению методологии причастны также Уорд Каннингем, Мартин Фаулер и другие.

Цель методики XP — справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки. Поэтому XP хорошо подходит для сложных и неопределенных проектов.

Методология XP строится вокруг четырех процессов: кодирования, тестирования, дизайна и слушания (рис.21). Кроме того, экстремальное программирование имеет ценности: простоту, коммуникацию, обратную связь, смелость и уважение.

Рисунок 21 – Практики экстремального программирования

13 практик экстремального программирования

1. Вся команда

Все участники проекта с применением XP работают как одна команда. В нее обязательно входит представитель заказчика, лучше, если это будет реальный конечный пользователь продукта, разбирающийся в бизнесе. Заказчик выдвигает требования к продукту и расставляет приоритеты в реализации функциональности. Ему могут помогать бизнес-аналитики. Со стороны исполнителей в команду входят разработчики и тестировщики, иногда коуч, направляющий команду, и менеджер, который обеспечивает команду ресурсами.

2. Игра в планирование

Планирование в XP проводят в два этапа — планирование релиза и планирование итераций.

На планировании релиза команда программистов встречается с заказчиком, чтобы выяснить, какую функциональность он хочет получить к следующему релизу, то есть через 2-6 месяцев. Так как требования заказчика часто размытые, разработчики конкретизируют их и дробят на части, реализация которых занимает не более одного дня. Важно, чтобы заказчик разбирался в операционной среде, в которой будет работать продукт.

Задачи записываются на карточки, а заказчик определяет их приоритет. Дальше разработчики оценивают, сколько времени уйдет на каждую задачу. Когда задачи описаны и оценены, заказчик просматривает документацию и дает добро на начало работы. Для успеха проекта критично, чтобы заказчик и команда программистов играли на одном поле: заказчик выбирает действительно необходимую функциональность в рамках бюджета, а программисты адекватно сопоставляют требования заказчика со своими возможностями.

В XP если команда не успевает выполнить все задачи к дате релиза, то релиз не отодвигается, а режется часть функционала, наименее важная для заказчика.

3. Частые релизы версий

В XP версии выпускаются часто, но с небольшим функционалом. Во-первых, маленький объем функциональности легко тестировать и сохранять работоспособность всей системы. Во-вторых, каждую итерацию заказчик получает часть функционала, несущую бизнес-ценность.

4. Пользовательские тесты

Заказчик сам определяет автоматизированные приемочные тесты, чтобы проверить работоспособность очередной функции продукта. Команда пишет эти тесты и использует их для тестирования готового кода.

5. Коллективное владение кодом

В XP любой разработчик может править любой кусок кода, т.к. код не закреплен за своим автором. Кодом владеет вся команда.

6. Непрерывная интеграция кода

Это значит, что новые части кода сразу же встраиваются в систему. Во-первых, сразу видно, как последние изменения влияют на систему. Если новый кусок кода что-то сломал, то ошибку найти и исправить в разы проще, чем спустя неделю. Во-вторых, команда всегда работает с последней версией системы.

7. Стандарты кодирования

Когда кодом владеют все, важно принять единые стандарты оформления, чтобы код выглядел так, как будто он написан одним профессионалом. Можно выработать свои стандарты или принять готовые.

8. Метафора системы

Метафора системы — это ее сравнение с чем-то знакомым, чтобы сформировать у команды общее видение. Обычно метафору системы продумывает тот, кто разрабатывает архитектуру и представляет систему целиком.

9. Устойчивый темп

XP команды работают на максимуме продуктивности, сохраняя устойчивый темп. При этом экстремальное программирование негативно относится к переработкам и пропагандирует 40-часовую рабочую неделю.

10. Разработка, основанная на тестировании

Одна из самых трудных практик методологии. В XP тесты пишутся самими программистами, причем до написания кода, который нужно протестировать. При таком подходе каждый кусок функционала будет покрыт тестами на 100%. Когда пара программистов заливают код в репозиторий, сразу запускаются модульные тесты. Тогда разработчики будут уверены, что движутся в правильном направлении.

11. Парное программирование

Представьте двух разработчиков за одним компьютером, работающих над одним куском функциональности продукта. Это и есть парное программирование, самая спорная практика XP. Старая поговорка «одна голова хорошо, а две лучше» отлично иллюстрирует суть подхода. Из двух

вариантов решения проблемы выбирается лучший, код оптимизируется сразу же, ошибки отлавливаются еще до их совершения. В итоге имеем чистый код, в котором хорошо разбираются сразу двое разработчиков.

12. Простой дизайн

Простой дизайн в XP означает делать только то, что нужно сейчас, не пытаясь угадать будущую функциональность. Простой дизайн и непрерывный рефакторинг дают синергетический эффект — когда код простой, его легко оптимизировать.

13. Рефакторинг

Рефакторинг — это процесс постоянного улучшения дизайна системы, чтобы привести его в соответствие новым требованиям. Рефакторинг включает удаление дублей кода, повышение связности и снижение сопряжения. XP предполагает постоянные рефакторинги, поэтому дизайн кода всегда остается простым.

Преимущества экстремального программирования имеют смысл, когда команда полноценно использует хотя бы одну из практик XP:

- заказчик получает именно тот продукт, который ему нужен, даже если в начале разработки сам точно не представляет его конечный вид;

- команда быстро вносит изменения в код и добавляет новую функциональность за счет простого дизайна кода, частого планирования и релизов;

- код всегда работает за счет постоянного тестирования и непрерывной интеграции;

- команда легко поддерживает код, так как он написан по единому стандарту и постоянно рефакторится;

- быстрый темп разработки за счет парного программирования, отсутствия переработок, присутствия заказчика в команде;

- высокое качество кода;

- снижаются риски, связанные с разработкой, так как ответственность за проект распределяется равномерно и уход/приход члена команды не разрушит процесс;

- затраты на разработку ниже, так как команда ориентирована на код, а не на документацию и собрания.

Несмотря на все плюсы, XP не всегда работает и имеет ряд слабых мест:

– успех проекта зависит от вовлеченности заказчика, которой не так просто добиться;

– трудно предугадать затраты времени на проект, так как в начале никто не знает полного списка требований;

– успех XP сильно зависит от уровня программистов, методология работает только с senior специалистами;

– менеджмент негативно относится к парному программированию, не понимая, почему он должен оплачивать двух программистов вместо одного;

– регулярные встречи с программистами дорого обходятся заказчикам;

– требует слишком сильных культурных изменений, чтобы не контролировать каждую задачу;

– из-за недостатка структуры и документации не подходит для крупных проектов;

– так как гибкие методологии функционально-ориентированные, нефункциональные требования к качеству продукта сложно описать в виде пользовательских историй.

2.8 Методология FDD

FDD (Feature Driven Development — разработка, управляемая функциональностью) также реализует собой гибкую методологию разработки, в основе которой лежит итеративный подход к разработке ПО.

FDD является попыткой объединить методики, которые наиболее признаны в индустрии разработки ПО. В основе этой методологии — нацеленность на реализацию важной для заказчика функциональности разрабатываемого ПО. Особенность этой методологии, как и других гибких методологий, в том, что разработка ведется короткими итерациями, в каждой из которых добавляется новая «фича» (полезное поведение с точки зрения заказчика). Методология ориентирована на корпоративную разработку.

В рамках методологии FDD все проектные работы требуется разделить на пять процессов (рис. 22). Во время нулевой итерации (первичной проектной деятельности в терминах FDD) происходит реализация трех процессов. Разрабатывается общая модель, составляется иерархический список необходимых функций, производится оценка функций с точки зрения трудозатрат.

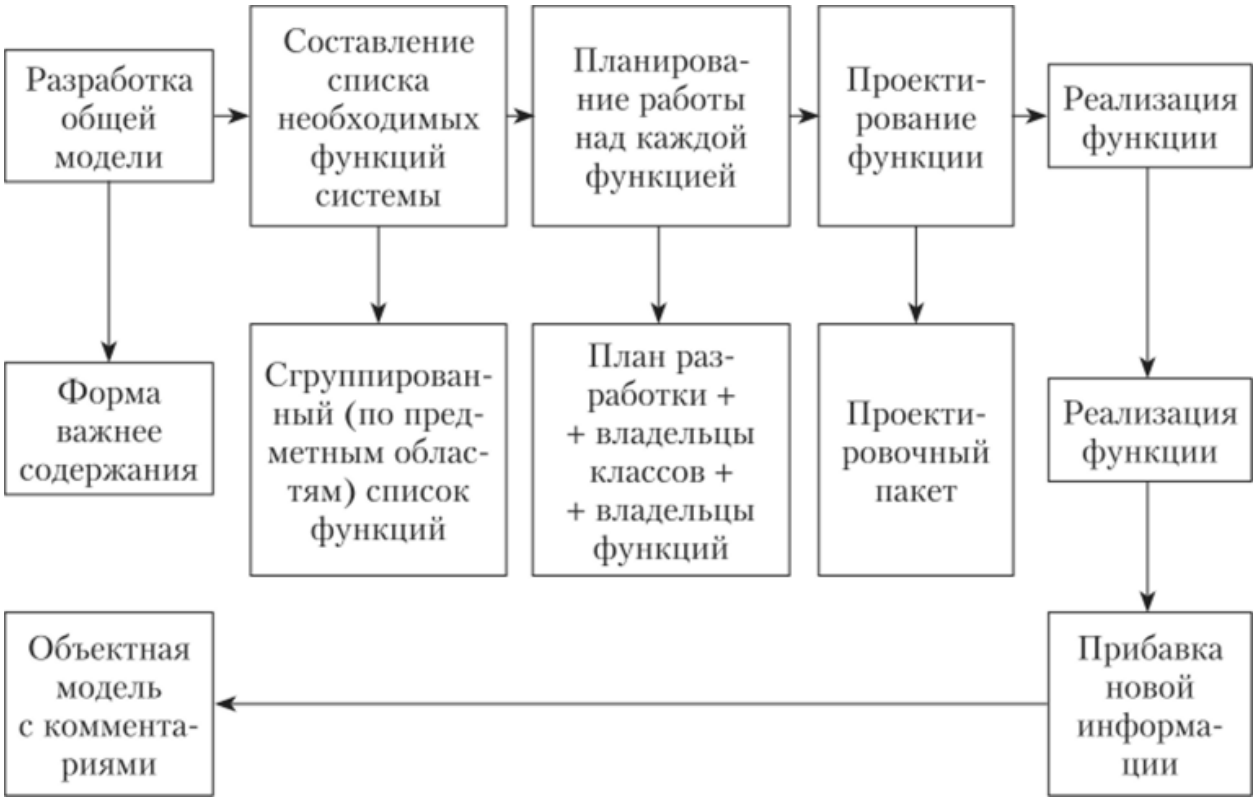


Рисунок 22 - Схема методологии FDD

После выделения основных функций для клиента начинается процесс их реализации, который происходит итеративно (рис. 23). Ранее созданная модель системы может корректироваться и уточняться. В конце каждой итерации результатом становится работающая функция, которая уже установлена на промышленный сервер. Особый акцент в FDD делается на проектировании модели системы, в связи с чем высоко значение так называемого моделирования в цвете при помощи UML.

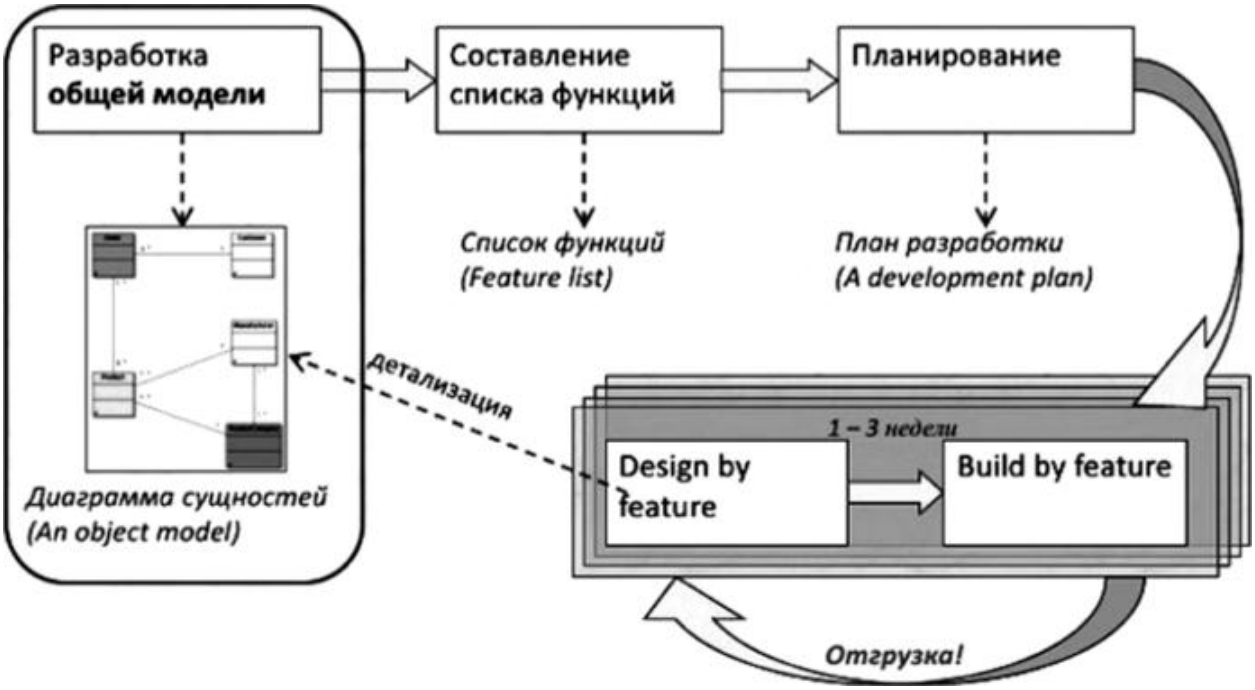


Рисунок 23 - Схема процесса FDD

Па этапе моделирования создается несколько конкурирующих моделей за авторством разных команд, независимых друг от друга. После этого совместными усилиями строится общая модель, которая содержит лучшие решения из каждой модели. За каждую отдельную итерацию рассматривается только один кусок модели, который предварительно уточняется и детализируется (производятся так называемые дизайн-сессии).

Процесс разработки контролируется за счет плана, который составляется заранее. Также для контроля определяют ответственных лиц. Методология FDD предполагает, что результат должен быть видимым, иначе

менеджеру будет тяжело понять, на каком этапе находятся работы по проекту.

Наличие ответственных лиц снижает коммуникационную проблему внутри команды и позволяет эффективно работать географически распределенным группам. Ответственным за последовательность выполнения считается ведущий разработчик. Он планирует работы, на основании которых ответственные по классам самоорганизуются с целью получить требуемый результат.

FDD уделяет пристальное внимание проверкам на этапе разработки. Использование проверок несет в себе следующие преимущества:

• обмен опытом и культурой разработки;

• соответствие стандартам.

Несмотря на очевидные плюсы, методология FDD обладает ярко выраженными недостатками. Прежде всего, в методологии отсутствует практика проведения встреч между разработчиками. В результате страдает доверие к коллегам.

Другой недостаток — наличие правил только по вопросам управления разработкой и проектированием.

2.9 Методология объектного проектирования на языке UML

UML (англ. *Unified Modeling Language* — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной моделисистемы, называемой *UML-моделью*. UML был создан для определения, визуализации, проектирования и документирования в основном программных систем. UML не является языком программирования, но в средствах выполнения UML-моделей как интерпретируемого кода возможна кодогенерация [30,31,32].

Использование UML не ограничивается моделированием программного обеспечения. Его также используют для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (generalization), объединение (aggregation) и поведение), и больше сконцентрироваться на проектировании и архитектуре [28].

В UML используются следующие виды диаграмм (для исключения неоднозначности приведены также обозначения на английском языке) (рис.24).

Structure Diagrams:
- Class diagram
- Component diagram
- Composite structure diagram
  - Collaboration (UML2.0)
- Deployment diagram
- Object diagram
- Package diagram
- Profile diagram (UML2.2)

Behavior Diagrams:
- Activity diagram
- State Machine diagram
- Use case diagram
- **Interaction Diagrams:**
  - Communication diagram (UML2.0) / Collaboration (UML1.x)
  - Interaction overview diagram (UML2.0)
  - Sequence diagram
  - Timing diagram (UML2.0)

Структурные диаграммы:
- Диаграмма классов
- Диаграмма компонентов
- Композитной/составной структуры
  - Диаграмма кооперации (UML2.0)
- Диаграмма развёртывания
- Диаграмма объектов
- Диаграмма пакетов
- Диаграмма профилей (UML2.2)

Диаграммы поведения:
- Диаграмма деятельности
- Диаграмма состояний
- Диаграмма прецедентов
- **Диаграммы взаимодействия:**
  - Диаграмма коммуникации (UML2.0) / Диаграмма кооперации (UML1.x)
  - Диаграмма обзора взаимодействия (UML2.0)
  - Диаграмма последовательности
  - Диаграмма синхронизации (UML2.0)

Рисунок 24 – Виды диаграмм

Структуру диаграмм UML 2.3 можно представить на диаграмме классов UML (рис.25).

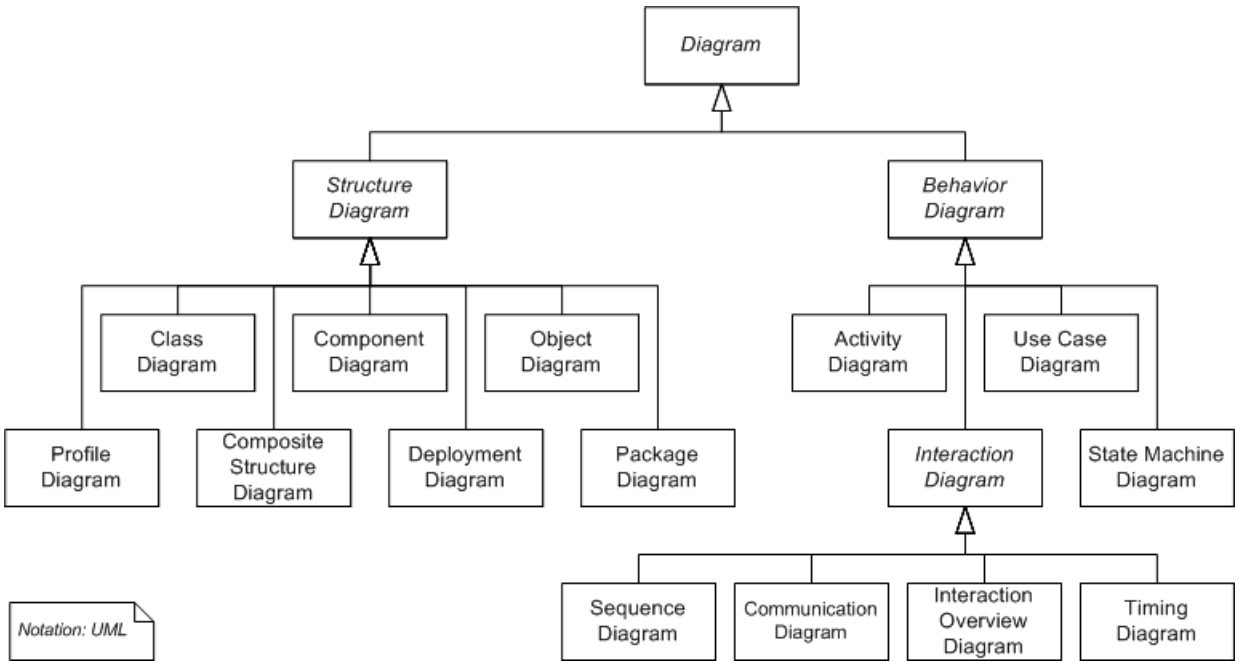


Рисунок 25 - Структура диаграмм UML 2.3

Диаграмма классов (Class diagram) — статическая структурная диаграмма, описывающая структуру системы, она демонстрирует классы системы, их атрибуты, методы и зависимости между классами.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

- концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
- точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;
- точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма компонентов (Component diagram) — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонент могут выступать файлы, библиотеки,

модули, исполняемые файлы, пакеты и т. п.

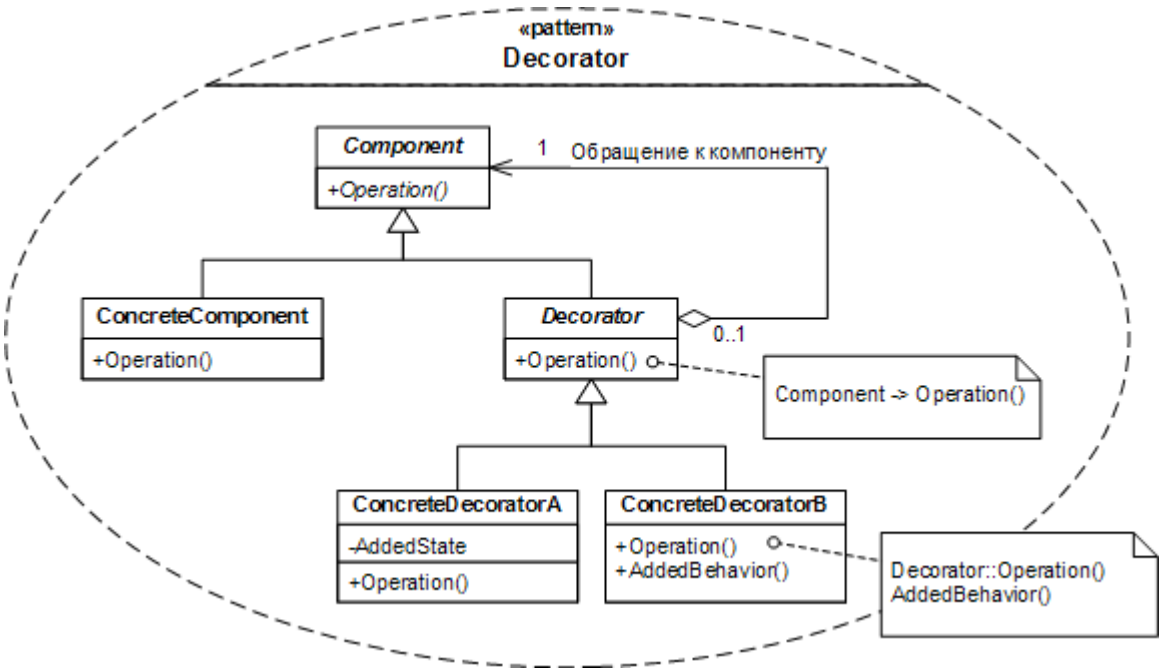


Рисунок 26 - Шаблон проектирования

Диаграмма композитной/составной структуры (Composite structure diagram) — статическая структурная диаграмма, демонстрирует внутреннюю структуру классов и, по возможности, взаимодействие элементов (частей) внутренней структуры класса.

Подвидом диаграмм композитной структуры являются *диаграммы кооперации* (Collaboration diagram, введены в UML 2.0), которые показывают роли и взаимодействие классов в рамках кооперации. Кооперации удобны при моделировании шаблонов проектирования [46].

Диаграммы композитной структуры могут использоваться совместно с диаграммами классов.

Диаграмма развёртывания (Deployment diagram) — служит для моделирования работающих узлов (аппаратных средств, англ. *node*) иартефактов, развёрнутых на них. В UML 2 на узлах разворачиваются артефакты (англ. *artifact*), в то время как в UML 1 на узлах разворачивались компоненты. Между артефактом и логическим элементом (компонентом),

который он реализует, устанавливается зависимость манифестации.

Диаграмма объектов (Object diagram) — демонстрирует полный или частичный снимок моделируемой системы в заданный момент времени. На диаграмме объектов отображаются экземпляры классов (объекты) системы с указанием текущих значений их атрибутов и связей между объектами.

Диаграмма пакетов (Package diagram) — структурная диаграмма, основным содержанием которой являются пакеты и отношения между ними. Жёсткого разделения между разными структурными диаграммами не проводится, поэтому данное название предлагается исключительно для удобства и не имеет семантического значения (пакеты и диаграммы пакетов могут присутствовать на других структурных диаграммах). Диаграммы пакетов служат, в первую очередь, для организации элементов в группы по какому-либо признаку с целью упрощения структуры и организации работы с моделью системы.

Диаграмма деятельности (Activity diagram) — диаграмма, на которой показано разложение некоторой *деятельности* на её составные части. Под деятельностью (англ. *activity*) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных *действий* (англ. *action*), соединённых между собой потоками, которые идут от выходов одного узла к входам другого.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

Аналогом диаграмм деятельности являются схемы алгоритмов по ГОСТ 19.701-90.

Диаграмма автомата (State Machine diagram, *диаграмма конечного автомата*, *диаграмма состояний*) — диаграмма, на которой представлен *конечный автомат* с простыми состояниями, переходами и композитными состояниями.

Конечный автомат (англ. *State machine*) — спецификация последовательности состояний, через которые проходит объект или взаимодействие в ответ на события своей жизни, а также ответные действия объекта на эти события. Конечный автомат прикреплён к исходному элементу (классу, кооперации или методу) и служит для определения поведения его экземпляров.

Диаграмма вариантов использования (Use case diagram) — диаграмма, на которой отражены отношения, существующие между акторами и вариантами использования.

Основная задача — представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы.

Диаграммы коммуникации и последовательности транзитивны, выражают взаимодействие, но показывают его различными способами и с достаточной степенью точности могут быть преобразованы одна в другую.

Диаграмма коммуникации (Communication diagram, в UML 1.x — *диаграмма кооперации*, *collaboration diagram*) — диаграмма, на которой изображаются взаимодействия между частями композитной структуры или ролями кооперации. В отличие от диаграммы последовательности, на диаграмме коммуникации явно указываются отношения между элементами (объектами), а время как отдельное измерение не используется (применяются порядковые номера вызовов).

Диаграмма последовательности (Sequence diagram) — диаграмма, на которой изображено упорядоченное во времени взаимодействие объектов. В частности, на ней изображаются участвующие во взаимодействии объекты и последовательность сообщений, которыми они обмениваются.

Диаграмма сотрудничества — Этот тип диаграмм позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта и типы этих

сообщений.

По причине того, что диаграммы Sequence и Collaboration являются разными взглядами на одни и те же процессы, Rational Rose позволяет создавать из Sequence диаграммы диаграмму Collaboration и наоборот, а также производит автоматическую синхронизацию этих диаграмм [47].

Диаграмма обзора взаимодействия (Interaction overview diagram) — разновидность диаграммы деятельности, включающая фрагменты диаграммы последовательности и конструкции потока управления.

Этот тип диаграмм включает в себя диаграммы Sequence diagram (диаграммы последовательностей действий) и Collaboration diagram (диаграммы сотрудничества). Эти диаграммы позволяют с разных точек зрения рассмотреть взаимодействие объектов в создаваемой системе.

Диаграмма синхронизации (Timing diagram) — альтернативное представление диаграммы последовательности, явным образом показывающее изменения состояния на линии жизни с заданной шкалой времени. Может быть полезна в приложениях реального времени.

3 ОРГАНИЗАЦИЯ И ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

3.1 Каноническое проектирование

Каноническое проектирование ИС направлено на отражение особенностей технологии индивидуального (оригинального) проектирования. Среди основных характерных особенностей канонического проектирования можно выделить такие особенности, как:

- отражение особенностей ручной технологии проектирования;
- ориентация на индивидуальное (оригинальное) проектирование;
- осуществление на уровне исполнителей; возможность интеграции выполнения элементарных операций; применение, как правило, для сравнительно небольших, локальных ИС;
- использование инструментальных средств универсальной компьютерной поддержки.

Каноническое проектирование направлено на минимальное использование типовых проектных решений. Адаптация проектных решений при каноническом проектировании осуществляется только путем перепрограммирования соответствующих программных модулей.

Организация канонического проектирования ИС основана на использовании каскадной модели жизненного цикла и предусматривает набор определенных стадий и этапов. Принцип деления процесса проектирования на стадии и этапы направлен на то, чтобы проектировать систему «сверху-вниз» и постепенно разрабатывать - изначально укрупненные, затем детализированные – проектные решения.

Поскольку объекты автоматизации имеют различную сложность и набор задач для создания решения для конкретной ИС, стадии и этапы работ также могут различаться по трудоемкости: существует возможность объединять последовательные этапы, исключать определенные из них на

любой стадии проекта, а также до окончания предыдущей стадии начинать выполнение следующей.

Стадии и этапы разработки ИС, которые выполняют организации-участники, оформляются в договорах и технических заданиях на выполнение работ. Каноническое проектирование основано на ряде стандартов, таких, как [10-20]:

1. ГОСТ 34.003 – термины и определения основных понятий в области автоматизированных систем;
2. ГОСТ 34.201 – виды, комплектность и обозначение документов при создании автоматизированных систем;
3. ГОСТ 34.601 – стадии создания автоматизированных систем;
4. ГОСТ 34.602 – техническое задание на создание ИС;
5. ГОСТ 34.603 – виды испытаний автоматизированных систем;
6. РД 50-34.698 – требования к содержанию документов;
7. ГОСТ 2.105 – общие требования к текстовым документам.

По отношению к проекту разработки ИС можно выделить 3 укрупненные стадии проектирования:

- предпроектную (стадии 1-3);
- проектную (стадии 4-6);
- послепроектную (стадии 7-8).

Стадии и этапы создания ИС, выполняемые организациями-участниками, фиксируются в договорах и технических заданиях на выполнение работ.

Предпроектная стадия направлена на предпроектное обследование и разработку технического задания на ИС. Характерными результатами этого этапа являются: определение целей и задач системы, формирование общих требований к ее созданию, разработка программы проведения обследования, в ходе которого должны быть изучены структура и бизнес-процессы организации, модель управления, задачи, подлежащие автоматизации,

технико- экономические характеристики, ориентировочных состав технических средств.

Перечень 8 этапов работ (стадий), в соответствии с ГОСТ 34.601 и дополнительными пояснениями, представлен ниже:

Стадия 1. Формирование требований к ИС.

- обследование объекта и обоснование необходимости создания ИС;

- формирование требований пользователя к ИС;

- оформление отчета о выполненной работе и заявки на разработку ИС (ТТХ).

Стадия 2. Разработка концепции ИС.

- изучение объекта;

- проведение необходимых научно- исследовательских работ;

- разработка вариантов концепции ИС, удовлетворяющих требованиям пользователей;

- оформление отчета о проделанной работе.

Стадия 3. Техническое задание.

- разработка и утверждение технического задания на создание ИС.

Важным документом, фиксирующим результаты определения стратегии внедрения ИС, является технико-экономическое обоснование проекта. В этом документе должно быть четко определены результаты выполнения проекта для заказчика, а также указаны графики выполнения работ и график финансирования на разных этапах выполнения проекта. Дополнительно в таком документе отражаются сроки, время окупаемости проекта, ожидаемые выгода и экономический эффект проекта.

Ориентировочно технико-экономическое обоснование содержит:

- все риски и ограничения, влияющие на успешность проекта;

- условия эксплуатации будущей системы: архитектурные, программные, аппаратные требования, требования к компонентам ПО и СУБД;

- пользователи системы;

- функции, выполняемые системой;

интерфейсы и распределение функций между человеком и системой;

сроки завершения этапов, форма приемки/ сдачи работ;

рамки проекта; возможности развития системы.

По результатам обследования формируется техническое задание на информационную систему.

В соответствии с ГОСТ 34.602-89, техническое задание (ТЗ) – основной документ, определяющий требования и порядок создания (развития или модернизации) автоматизированной системы, в соответствии с которым проводится разработка ИС и ее приемка при вводе в действие.

Разработка технического задания предусматривает описание следующих разделов:

- общие сведения; назначение и цели создания (развития) системы;
- характеристика объектов автоматизации;
- требования к системе;
- состав и содержание работ по созданию системы;
- порядок контроля и приемки системы;
- требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;
- требования к документированию;
- источники разработки.

Проектная стадия главным образом ориентирована на разработку технического и рабочего проектов. Процесс разработки технического задания включает обследование объекта автоматизации (организации или подразделения) и его систем управления. Для решения задач информационного обеспечения необходимо проанализировать информационные потоки, формы документации, системы кодирования, а также все связанное со структурой баз данных и СУБД, что определяет состав исходных технологических требований.

Стадия 4. Эскизный проект.

- разработка предварительных проектных решений по системе и ее частям;

- разработка эскизной документации на ИС и ее части.

Если для ИС конкретного объекта автоматизации проектные решения выбраны ранее или являются очевидными, стадия эскизного проекта может быть исключена из последовательности работ. Таким образом, эта стадия не является строго обязательной.

На этапе эскизного проекта, в том числе, должны быть определены:

- цели, функции ИС и подсистем;

- состав комплексов задач и отдельных задач;

- концепция и структура информационной базы;

- функции СУБД;

- функции и параметры основных программных средств;

- ожидаемый эффект от ее внедрения.

Документация, содержащая результаты работ по совокупности принятых проектных решений, согласовывается, утверждается и используется в дальнейшем для выполнения работ по созданию ИС.

На основании технического задания (в тои числе, при наличии эскизного проекта) разрабатывается технический проект ИС.

Стадия 5. Технический проект.

- разработка проектных решений по системе и ее частям;

- разработка документации на ИС и ее части;

- разработка и оформление документации на поставку изделий для комплектования ИС и (или) технических требований (технических заданий) на их разработку;

- разработка заданий на проектирование в смежных частях проекта объекта автоматизации.

На этапе технического проекта проводятся работы научно-исследовательского и экспериментального характера для выбора основных

проектных решений, а также рассчитывается экономическая эффективность системы.

Важным аспектом разработки технического проекта является анализ всей используемой информации на предмет таких характеристик, как полнота, отсутствие дублирования и избыточности, непротиворечивость и т.д., а также определение форм выходных документов. Документация должна быть оформлена в соответствии с требованиями ГОСТ 34-201 и РД 50-34.698.

Стадия 6. Рабочая документация.

- разработка рабочей документации на систему и ее части;
- разработка или адаптация программ.

Один из основных этапов стадии рабочего проектирования – разработка рабочей документации на информационное обеспечение ИС, в состав которой входят: технический проект ИС, описание баз данных, перечень исходных и выходных данных и документов.

Стадия технического проектирования завершается подготовкой и оформлением документации на поставку для комплектования ИС и определением технических требований и составлением ТЗ на разработку ИС.

Стадия «Рабочая документации» предполагает создание, как программного продукта, так и всей сопровождающей документации, которая должна предоставлять все сведения, обеспечивающие выполнение работ на стадиях ввода ИС в действие и эксплуатации ИС, в том числе, сведения для поддержания уровня качества ИС (соблюдения эксплуатационных характеристик).

Послепроектная стадия включает в себя реализацию мероприятий по внедрению, подготовку помещений и технических средств, обучение персонала. Также производится эксплуатация системы с решением конкретных задач, анализируются результаты испытаний, реализуются мероприятия по сопровождению ИС.

Стадия 7. Ввод в действие.

- подготовка объекта автоматизации к вводу ИС в действие;

- подготовка персонала;

- комплектация ИС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями);

- строительно-монтажные работы;

- пусконаладочные работы;

- проведение предварительных испытаний;

- проведение опытной эксплуатации;

- проведение приемочных испытаний.

Основными видами испытаний для ИС являются такие, как: предварительные испытания, опытная эксплуатация и приемочные испытания, которые при необходимости могут быть расширены дополнительными испытаниями ИС и ее составляющих частей.

В ходе предварительных испытаний, регламентируемых соответствующей программой и методикой, главным образом проводятся испытания системы на работоспособность и соответствие ТЗ, а также устранение неисправностей и внесение изменений в документацию на ИС.

На следующем этапе происходит процесс проведения опытной эксплуатации, анализируются ее результаты, и при необходимости проводится доработка ПО и дополнительная наладка технических средств ИС.

В процессе проведения приемочных испытаний реализуются испытания на соответствие ТЗ, анализируются результаты испытания системы, устраняются недостатки, которые были выявлены при испытаниях.

При всех видах испытаний оформляются соответствующие акты о приемке системы в опытную эксплуатацию, ее завершении и о приемке системы в постоянную эксплуатацию.

Стадия 8. Сопровождение ИС.

- выполнение работ в соответствии с гарантийными обязательствами;

- послегарантийное обслуживание.

Основными процессами этой стадии являются осуществление работ по устранению недостатков, выявленных при эксплуатации системы в течение гарантийных сроков, а также анализ функционирования системы, выявление отклонений и их причин, устранение причин отклонений и недостатков, обеспечение стабильности эксплуатационных характеристик.

3.2 Типовое проектирование

Методы типового проектирования направлены на выполнение проектирования ИС с использованием типовых проектных решений [22, 29, 41].

Типовое проектное решение – проектное решение, пригодное к многократному использованию (тиражируемое проектное решение).

Применение методов типового проектирования имеет свои особенности. Основным условием для использования таких методов является возможность декомпозиции проектируемой ИС на составляющие компоненты (подсистемы, программные модули, комплексы выполняемых задач и т.д.), для реализации которых можно выбрать типовые проектные решения, существующие на рынке, которые будут настроены на нужды конкретного предприятия.

Помимо собственно функциональных (программных, аппаратных) элементов, типовое решение подразумевает наличие необходимой документации, в которой дается детальное описание ТПР (в том числе, процедур настройки), отвечающее требованиям проектируемой системы.

По уровню декомпозиции системы можно выделить такие классы ТПР, как:

- элементные ТПР – ТПР по отдельному элементу (задаче, виду обеспечения);

- подсистемные ТПР – ТПР по отдельным подсистемам;

- объектные ТПР – отраслевые ТПР, включающие весь набор подсистем ИС.

Выделенные классы ТПР имеют свои достоинства и недостатки. Рассмотрим наиболее характерные из них.

К достоинству элементных ТПР можно отнести реализацию модульного подхода к проектированию ИС. В то же время это приводит к большим затратам на доработку ТПР конкретных элементов и к затратам на объединение разных элементов вследствие их несовместимости.

Подсистемные ТПР также позволяют реализовать модульный подход к проектированию ИС. Кроме того, они позволяют осуществлять параметрическую настройку компонентов на объекты различных уровней управления; взаимосвязанные компоненты и высокая степень интеграции элементов ИС приводят к минимизации затрат на проектирование и программирование. Однако в случае нескольких производителей программного обеспечения появляются проблемы в объединении различных функциональных подсистем; помимо этого, с точки зрения непрерывного реинжиниринга процессов адаптивность ТПР является недостаточной.

Объектные ТПР имеют такие преимущества, как:

- масштабируемость (допускаются конфигурации ИС для различного числа рабочих мест);

- методологическое единство компонентов ИС;

- совместимость компонентов ИС;

- открытость архитектуры (возможность развертывания ТПР на платформах разного типа);

- конфигурируемость (возможность использовать необходимое подмножество компонентов системы).

К недостаткам объектных ТПР можно отнести проблемы реализации типового проекта в оригинальном объекте управления, что приводит в определенных ситуациях к необходимости смены организационной структуры объекта автоматизации.

При реализации типового проектирования применяются такие подходы, как: ***параметрически-ориентированное*** и ***модельно-ориентированное*** проектирование.

Этапами ***параметрически-ориентированного*** проектирования являются:

- постановка задач и определение пригодности пакетов прикладных программ (ППП) для их решения через систему критериев оценки;
- анализ доступных ППП исходя из критериев; выбор и приобретение подходящего ППП;
- настройка параметров приобретенного ППП.

Среди критериев оценки ППП выделяют [39] следующие группы:

- назначение и возможности пакета;
- отличительные признаки и свойства пакета;
- требования к техническим и программным средствам;
- документация пакета; факторы финансового порядка;
- особенности установки пакета;
- особенности эксплуатации пакета;
- помощь поставщика по внедрению и поддержанию пакета;
- оценка качества пакета и опыт его использования;
- перспективы развития пакета.

Отметим, что каждая из перечисленных групп критериев может быть детализирована на совокупность частных показателей, дающих дополнительную информацию для каждого аспекта анализа, выбранного ППП. Значения критериев определяются с использованием методов экспертного оценивания.

Другим подходом реализации типового проектирования является модельно-ориентированное проектирование, сущность которого состоит в адаптации существующих характеристик типовой ИС, исходя из модели объекта автоматизации, построение которой предполагает использование специального программного инструментария.

При таком подходе технология проектирования должна иметь средства как для работы с моделью конкретного предприятия, так и с моделью типовой ИС.

В репозитории типовой ИС содержится модель объекта автоматизации, которая является основой для конфигурирования программного обеспечения. Кроме того, в репозитории содержится базовая (ссылочная) модель ИС и типовая (референтная) модели ее определенных классов.

Базовая модель ИС описывает бизнес-процессы, организационную структуру, бизнес-объекты, бизнес- функции, для поддержки которых предназначены программные модули типовой ИС.

Типовые модели предназначены для описания конфигурации ИС для тех или иных отраслей, типов производства.

Модель конкретного предприятия может быть построена либо в результате выбора фрагментов типовой модели с учетом особенностей объекта автоматизации (BAAN Enterprise Modeler), либо с использованием автоматизированной адаптации этих модулей с учетом мнений экспертов (SAP Business Engineering Workbench). Модель предприятия, на основе которой осуществляется автоматическое конфигурирование и настройка ИС, хранится в репозитории и может быть откорректирована в случае необходимости.

Внедрение типовой ИС начинается с анализа результатов предпроектного обследования предприятия, сформированных в виде требований к конкретной ИС, для оценки которых может быть использована методика оценки ППП. На следующем этапе необходимо построить предварительную модель ИС, которая должна полно отражать особенности реализации ИС для конкретного объекта автоматизации. Предварительная модель – основа для выбора типовой модели системы, а также для формирования перечня компонентов, для реализации которых потребуются другие программные средства или инструментальные средства, имеющиеся в составе типовой ИС.

При реализации типового проекта имеет место выполнение следующих операций [43]:

- установку глобальных параметров системы;
- задание структуры объекта автоматизации;
- определение структуры основных данных;
- задание перечня реализуемых функций и процессов; описание интерфейсов;
- описание отчетов;
- настройку авторизации доступа;
- настройку системы архивирования.

Типовое проектирование в настоящее время широко представлено в современных средствах.

3.3 Технология Rational Unified Process

Среди всех фирм-производителей CASE-средств компания IBM Rational Software Corp. (до августа 2003 года – это самостоятельная фирма Rational Software Corp.) одна из первых осознала стратегическую перспективность развития объектно-ориентированных технологий анализа и проектирования программных систем. Эта компания выступила инициатором унификации языка визуального моделирования в рамках консорциума OMG, что привело к появлению первых версий языка UML. Эта же компания первой разработала инструментальное объектно-ориентированное CASE-средство, в котором был реализован язык UML, как базовая нотация визуального моделирования [30, 45]. Графическое представление методологии RUP из Википедии изображено на рисунке 27.

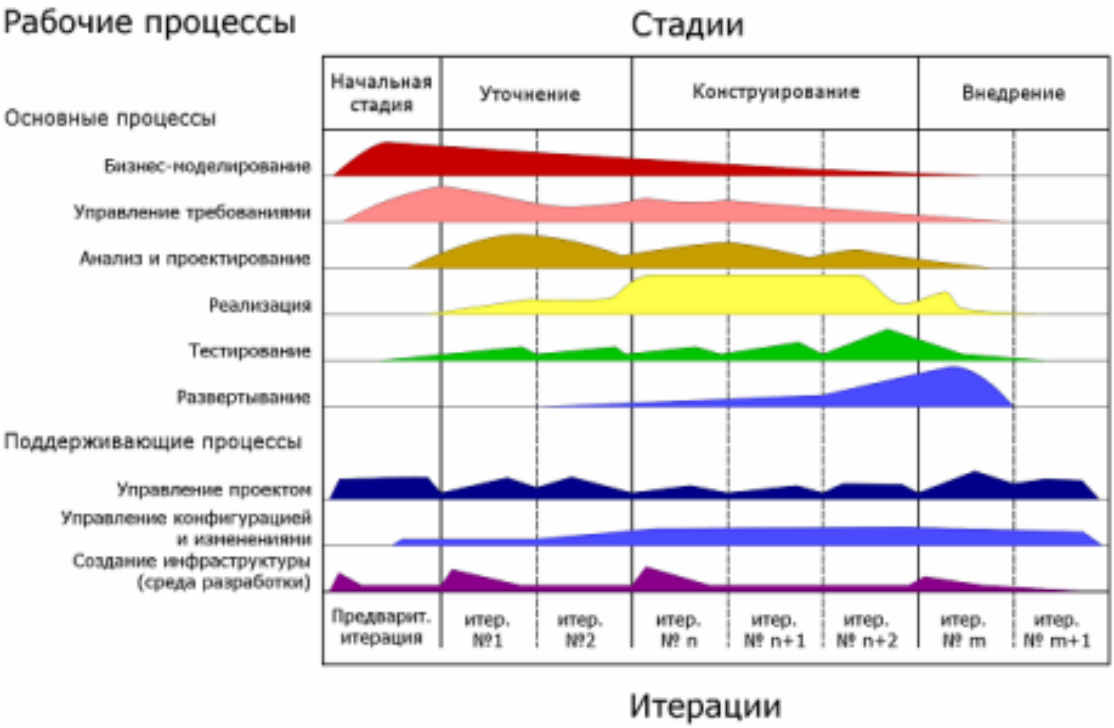


Рисунок 27 – Представление технологии RUP

Одна из самых популярных технологий - Rational Unified Process (RUP). В определенном плане эта методология становится международным стандартом, разработанный компанией Rational Software, которая в настоящее время входит в состав IBM. Авторами UML считаются сотрудники фирмы Rational Software: Гради Буч, Айвар Якобсон, Джемс Рамбо. RUP полностью соответствует стандартам, определяющим проектные работы в процессе жизненного цикла информационных систем. В методологии RUP реализуются следующие подходы:

1. Итерационный и инкрементный (наращиваемый).
2. Построение системы на базе архитектуры информационной системы.
3. Планирование и управление проектом на основе функциональных требований к информационной системе.

Разработка информационной системы выполняется итерациями. Это отдельные проекты небольшие по объему и содержанию, которые включают свои собственные этапы анализа требований, проектирования, реализации,

тестирования, интеграции. Заканчиваются итерации созданием работающей информационной подсистемы.

Итерационный цикл характеризуется периодической обратной связью и может адаптироваться к ядру разрабатываемой системы. Создаваемая информационная система постепенно растет и совершенствуется.

Таким образом, итеративный подход, реализуемый в RUP, позволяет уточнять и дополнять требования к разрабатываемой системе на каждой итерации и реализовать их в предлагаемых решениях. Итеративный процесс разработки обеспечивает необходимую гибкость в случае изменения требований и появлении новых, что часто бывает при коррективах бизнес-целей прикладной области. Эта гибкость позволяет заблаговременно идентифицировать и более эффективно снижать проектные риски. Итеративный подход обеспечивает управление требованиями и их изменениями таким образом, чтобы между всеми участниками проекта было единое понимание предполагаемых функциональных возможностей создаваемого продукта, обеспечивался требуемый уровень качества, эффективное управление проектом.

Основными особенностями RUP являются следующие.

1. RUP ориентирован на визуальное моделирование и представление проекта разрабатываемой системы в виде набора визуальных моделей, которые семантически полно отражают представление о возможностях и конфигурации разрабатываемой ИС. RUP акцентирует внимание на таком факторе, как создание и последующее развитие надежной и гибкой архитектуры, которая реализуется идеей компонентного подхода. Этот фактор создает условия для параллельной разработки, минимизирует издержки в случае необходимости внесения изменений, создает возможность для многократного использования компонентов и, следовательно, повышает надежность эксплуатации системы. Такой подход создает предпосылки для планирования использования программных компонентов в других проектах и управления их последующим развитием.

2. RUP предлагает отображать действия разрабатываемой системы с помощью прецедентов, в совокупности определяющих функционал системы. Прецеденты и сценарии работы системы способствуют эффективному управлению последовательностью выполнения работ, начиная от бизнес-моделирования и спецификации требований вплоть до сдачи в эксплуатацию. Прецеденты обеспечивают связанные и доступные для анализа направления разработки и развертывания системы.

3. RUP ориентирован на объектно-ориентированный подход к проектированию. Объектно-ориентированный подход основан на понятиях объектов, классов и связей между ними. Модели, создаваемые по методологии RUP, подобно другим искусственным объектам (артефактам), в качестве единого стандарта для представления модели проектируемой системы используют UML.

4. RUP обеспечивает компонентно-ориентированный подход к проектированию. Компоненты — это оригинальные модули или подсистемы, которые выполняют конкретную, только им присущую функцию. Компоненты могут быть использованы многократно как в текущем проекте, так и в других новых проектах.

5. RUP — адаптируемый и конфигурируемый процесс. Способность RUP к адаптации позволяет использовать его как маленьким группам разработчиков, так и большим организациям.

6. RUP поддерживает управление качеством. Оценка качества всех работ, выполняемых любыми участниками проекта, использует объективные метрики и критерии. Методология RUP создавалась с целью поддержки управления качеством в соответствии с требованиями стандарта SEI CMM/CMMI. Подводя итог вышеизложенному, следует привести основные принципы, которые положены в основу RUP:

- идентификация на ранних этапах и непрерывный контроль и устранение всех основных рисков;

- акцент на выполнении требований заказчиков к разрабатываемой системе;

— постоянная готовность к изменению в требованиях, в проектных решениях, а также их реализации в процессе разработки системы;

- поддержка компонентной архитектуры, которая реализуется и тестируется еще на ранних стадиях проектирования;

— эффективное управление качеством на всех этапах разработки проекта.

3.4 Технология OpenUP

Также, как и RUP, Open UP использует принципы итеративности и инкрементальности в рамках структурированного жизненного цикла.

С одной стороны, OpenUP предлагает набор практик, в числе которых:

- концепция микрошагов, реализующая принцип непрерывного создания работающего программного продукта;

— концепция раннего тестирования;

- методика гибкого моделирования, предполагающая быстрое создание только той документации, которая будет нужна в ближайшее время.

С другой стороны, OpenUP предлагает:

а) шаблоны для:

— описания видения проектируемой системы;

— измерения состояния проекта;

— создания требований, архитектуры и тестовой документации;

— контрольные списки для проверки корректности полноты создаваемых артефактов;

б) упрощенный с точки зрения RUP процесс;

в) перечень ролей, описание их зон ответственности, создаваемых и потребляемых ими артефактов.

Впервые версия OpenUp 1.0 была опубликована в 2007 г. как проект с открытым исходным кодом, фокусирующийся на методологиях создания ПО. Метод развивается в рамках проекта Eclipse Process Framework (EPF).

OpenUP — это независимый от инструментов, мало регламентированный процесс, который можно расширить для адаптации к широкому диапазону типов проектов.

OpenUP, также, как и RUP, подразумевает, что проект состоит из итераций. Итерации — планируемые, ограниченные во времени интервалы, длительность которых достаточна коротка. План итерации определяет, какой результат должен быть получен, но ее завершении. Результатом итерации является версия, работу которой можно продемонстрировать или передать для ознакомления и оценки заинтересованным лицам.

Итерации в OpenUP организуются как набор фаз. Каждая фаза завершается контрольной точкой, предназначенной для обеспечения контроля путем постановки и ответов на ряд вопросов, которые обычно являются важными для заинтересованных лиц.

Личная работа в проекте OpenUP организована по принципу микрошагов (рис. 28). Микрошаги представляют собой небольшие единицы работы, которые формируют постоянное измеряемое приращение выполнения проекта (обычно их продолжительность составляет от нескольких часов до несколько дней). Процесс предполагает интенсивную совместную работу, поскольку разработка системы осуществляется постепенно, заинтересованным, построенным по принципу самоорганизации, коллективом. Микрошаги обеспечивают исключительно короткий цикл обратной связи, который делает возможным принятие адаптивных решений в ходе каждой итерации.

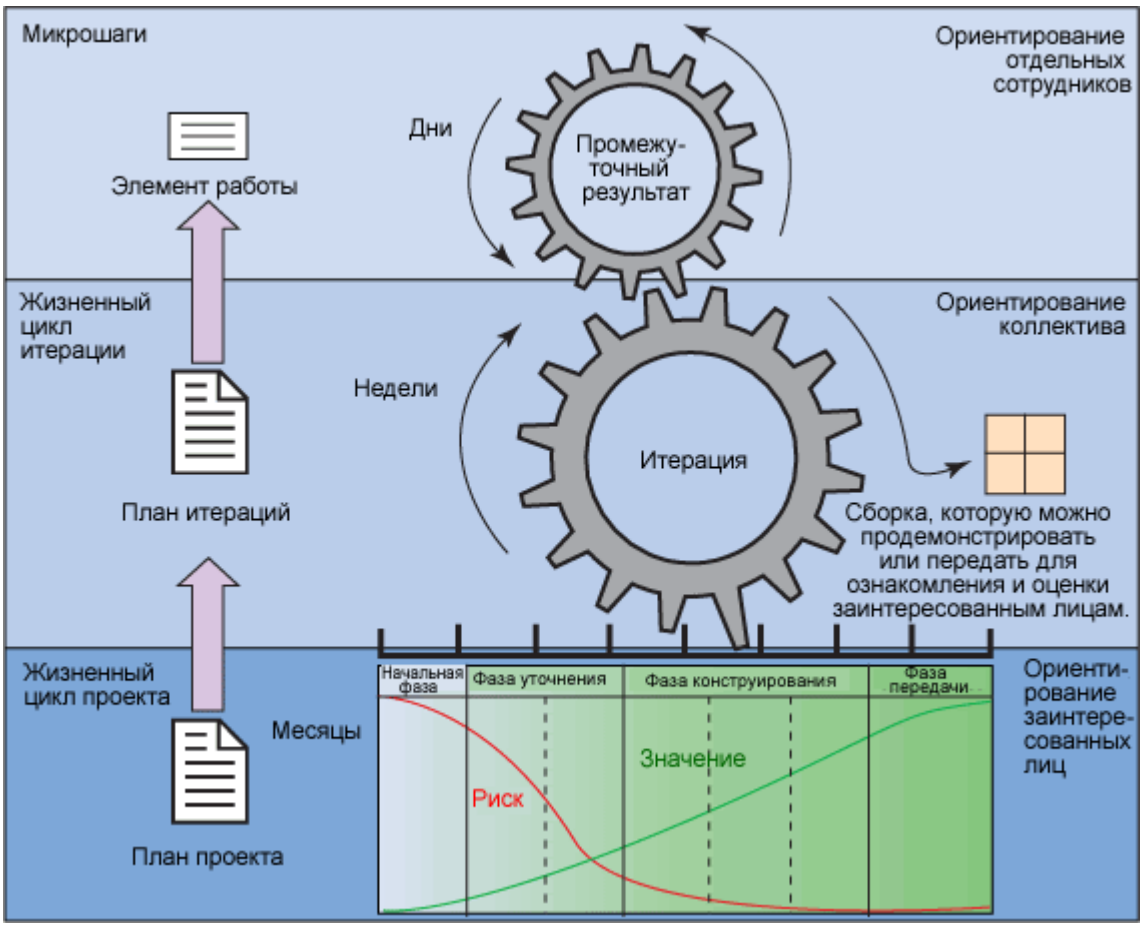


Рисунок 28 - Уровни OpenUP: микрошаги, жизненный цикл итерации и жизненный цикл проекта

OpenUP делит проект на итерации: планируемые, ограниченные во времени интервалы, длительность которых обычно измеряется неделями. Итерации ориентируют коллектив на поставку постепенно возрастающей потребительской ценности заинтересованным лицам предсказуемым образом. План итерации определяет, что именно должно быть сдано по окончании итерации, а результатом итерации является версия, работу которой можно продемонстрировать или передать для ознакомления и оценки заинтересованным лицам. Коллективы разработчиков OpenUP строятся по принципу самоорганизации, решая вопросы выполнения задач итераций и передачи результатов. Для этого они сначала определяют, а затем решают хорошо детализированные задачи из списка элементов работ. В OpenUP используется понятие жизненного цикла итерации, определяющее,

каким образом использовать микрошаги для поставки стабильных, все более совершенных сборок системы, которая постепенно приближается к цели итерации.

OpenUP делит жизненный цикл проекта на четыре фазы: начальная фаза, фазы уточнения, конструирования и передачи. Жизненный цикл проекта обеспечивает предоставление заинтересованным лицам и членам коллектива точек ознакомления и принятия решений на протяжении всего проекта. Это позволяет эффективно контролировать ситуацию и вовремя принимать решения о приемлемости результатов. План проекта определяет жизненный цикл, а конечным результатом является окончательное приложение.

Итерации в OpenUP поддерживают в коллективе разработчиков нацеленность на поставку постоянно растущей потребительской ценности каждые несколько недель путем сдачи полностью протестированной сборки (промежуточного продукта), которую можно продемонстрировать или передать для ознакомления и оценки заинтересованным лицам. Это создает полезную заинтересованность в создании потребительской ценности продукта для заинтересованных лиц, благодаря чему все, что делается, приближает коллектив к достижению цели. Принятие решений должно осуществляться быстрее, потому что время не расходуется на бесконечные обсуждения. Итеративная разработка нацелена на создание работоспособного кода, что снижает риск аналитического паралича. Частые демонстрации работоспособного кода поддерживают механизмы обратной связи, которые при необходимости позволяют вносить коррективы по ходу работы.

САМОСТОЯТЕЛЬНАЯ РАБОТА ПО ВЫПОЛНЕНИЮ ПРОЕКТА

Выполнение заданий имеет цель дать магистрантам практические навыки:

- проводить анализ предметной области;

- создавать функционально-ориентированные модели предметной области;

- создавать инфологическую, логическую и физические модели баз данных;

- использовать информационные технологии для проведения этапа анализа предметной области и проектирования базы данных;

- разрабатывать пользовательские интерфейсы систем.

Все проекты выполняются по вариантам. Варианты представлены ниже (табл.5). Разрешается сформулировать собственную задачу на проектирование и согласовать с преподавателем.

Таблица 5 – Варианты заданий

Тема	ФИО
Вариант 1. Проектирование информационной системы «Отпуск изделий».	
Вариант 2. Проектирование информационной системы «Успеваемость».	
Вариант 3. Проектирование информационной системы «Научно- исследовательская работа».	
Вариант 4. Проектирование информационной системы «Направления ВУЗа».	
Вариант 5. Проектирование информационной системы «Здравоохранение».	
Вариант 6. Проектирование информационной системы «Трудоустройство».	
Вариант 7. Проектирование информационной системы «Предприятие общепита».	

Вариант 8. Проектирование информационной системы «ГИБДД».	
Вариант 9. Проектирование информационной системы «Учет материальных ценностей».	
Вариант 10. Проектирование информационной системы «Подписка».	
Вариант 11. Проектирование информационной системы «Сессия».	
Вариант 12. Проектирование информационной системы «Договор».	
Вариант 13. Проектирование информационной системы «Санкции ГИБДД».	
Вариант 14. Проектирование информационной системы «Отдел кадров».	
Вариант 15. Проектирование информационной системы «Стандартизация».	
Вариант 16. Проектирование информационной системы «Грузоперевозки».	
Вариант 17. Проектирование информационной системы «Налогообложение».	
Вариант 18. Проектирование информационной системы «Общежитие».	
Вариант 19. Проектирование информационной системы «Недвижимость».	
Вариант 20. Проектирование информационной системы «Учет заявок на производство изделий».	
Вариант 21. Проектирование информационной системы «Медицинская страховая компания».	
Вариант 22. Проектирование информационной системы «Биржа труда».	
Вариант 23. Проектирование информационной системы «Справочник потребителя».	
Вариант 24. Проектирование информационной системы «Справочник покупателя».	
Вариант 25. Проектирование информационной системы «Магазин с одним продавцом».	
Вариант 26. Проектирование информационной системы «Отдел кадров».	

Вариант 27. Проектирование информационной системы «Складской учет».	
Вариант 28. Проектирование информационной системы «Обмен жилья».	
Вариант 29. Проектирование информационной системы «Сбербанк»	
. Вариант 30. Проектирование информационной системы «Ломбард».	
Вариант 31. Проектирование информационной системы «Справочник коммерческих банков».	
Вариант 32. Проектирование информационной системы «Очередь на жилье».	
Вариант 33. Проектирование информационной системы «Медицинский кооператив».	
Вариант 34. Проектирование информационной системы «Учет аудиторного фонда университета».	
Вариант 35. Проектирование информационной системы «Обслуживания работы конференции».	
Вариант 36. Проектирование информационной системы «Обслуживание склада».	

1 Анализ предметной области

Цель работы: сформировать практические умения проводить анализ деятельности предприятия: определять его цели и задачи, формулировать задачу на автоматизацию, проектировать текущие бизнес-процессы предприятия.

Постановка задачи для самостоятельной работы

Провести анализ предметной области (в соответствии с вариантом). Предметной областью для предмета проектирования систем является деятельность предприятия (или его отдельные бизнес процессы), для которого будет разрабатываться информационная систем.

Этапы выполнения задания:

1. Описание общих сведений о предприятии: название, область деятельности, общие цели предприятия.

Например: Компания ООО «Информационные системы» занимается оказанием услуг в сфере информационных технологий. Область деятельности предприятия: разработка автоматизированных рабочих мест и информационных систем.

Общие цели компании представлены на стратегической карте (рис. 29).

2. Постановка общей задачи на автоматизацию (для определения масштаба проекта): разработка корпоративной системы, автоматизация деятельности отдельного отдела предприятия, разработка рабочего места специалиста.

Например: Целью проекта на автоматизацию является автоматизации деятельности отдела кадров, которая будет включать в себя единую информационную систему по учету кадров на предприятии и два типа рабочих мест: специалист отдела кадров и начальник отдела кадров.

3. Анализ (создание) организационной структуры предприятия – описание иерархии подразделений, отделов, цехов, лабораторий, рабочих групп предприятия.

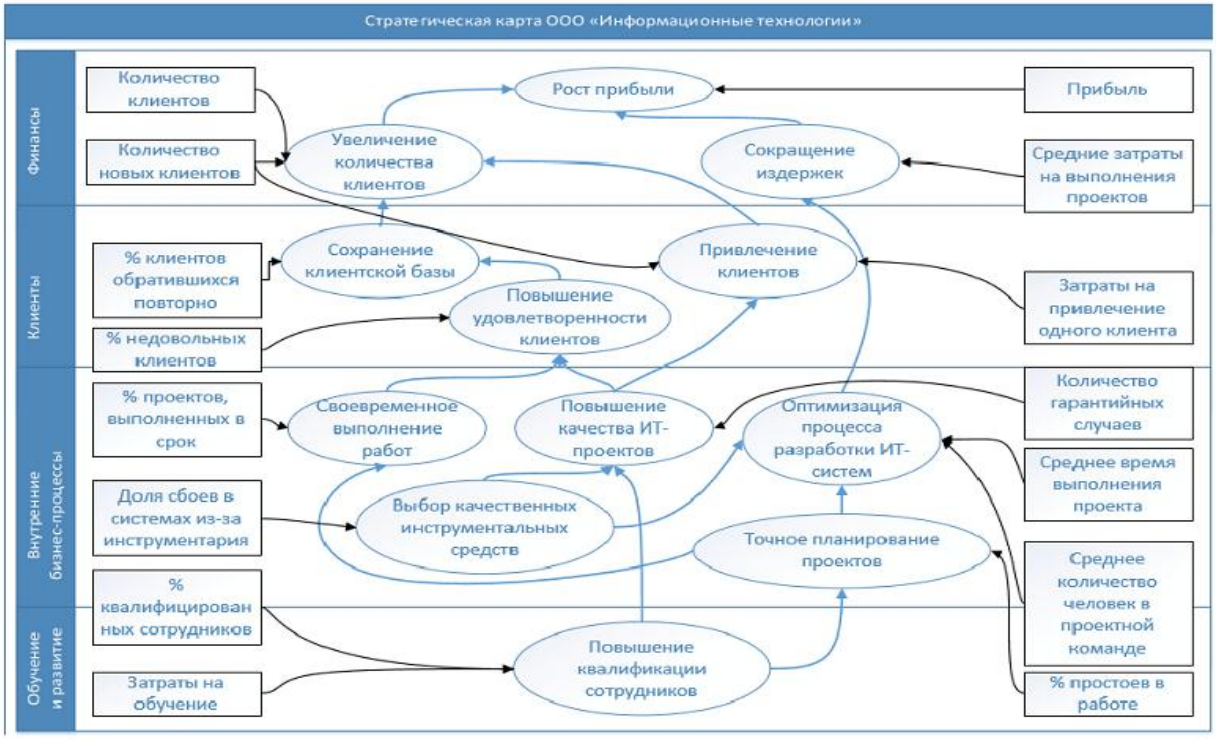


Рисунок 29 - Пример стратегической карты

Например: На рисунке 30 представлена организационная структура предприятия ОАО «Мечта».

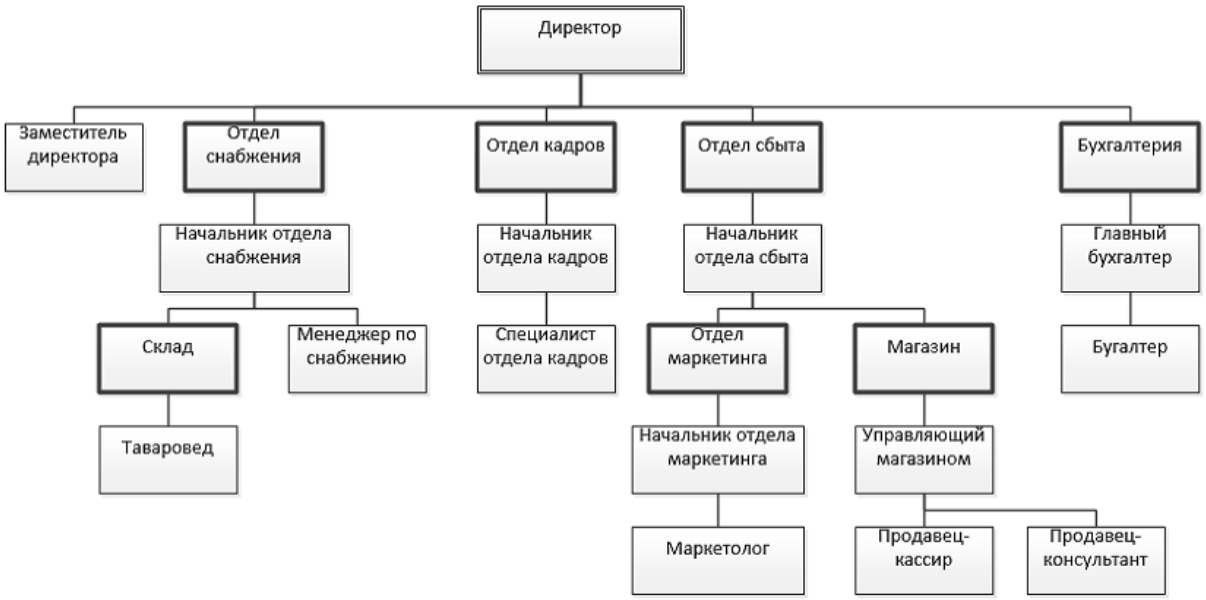


Рисунок 30 - Пример организационной структуры

4. Проектирование модели текущих бизнес-процессов предприятия (отдела, сотрудника) – модель AS-IS. Для проектирования бизнес-процессов компании необходимо использовать методологию BPMN.

Результатом этого пункта работы должно быть описание бизнес-процессов, подлежащих автоматизации (рис. 31).

5. Описание документов, которые необходимы для реализации бизнес-процессов или является результатами этих процессов. Перечисляются все документы, которые используются или формируются при выполнения рассмотренных в предыдущем пункте бизнес-процессах. По возможности прилагается шаблоны (или примеры) этих документов.

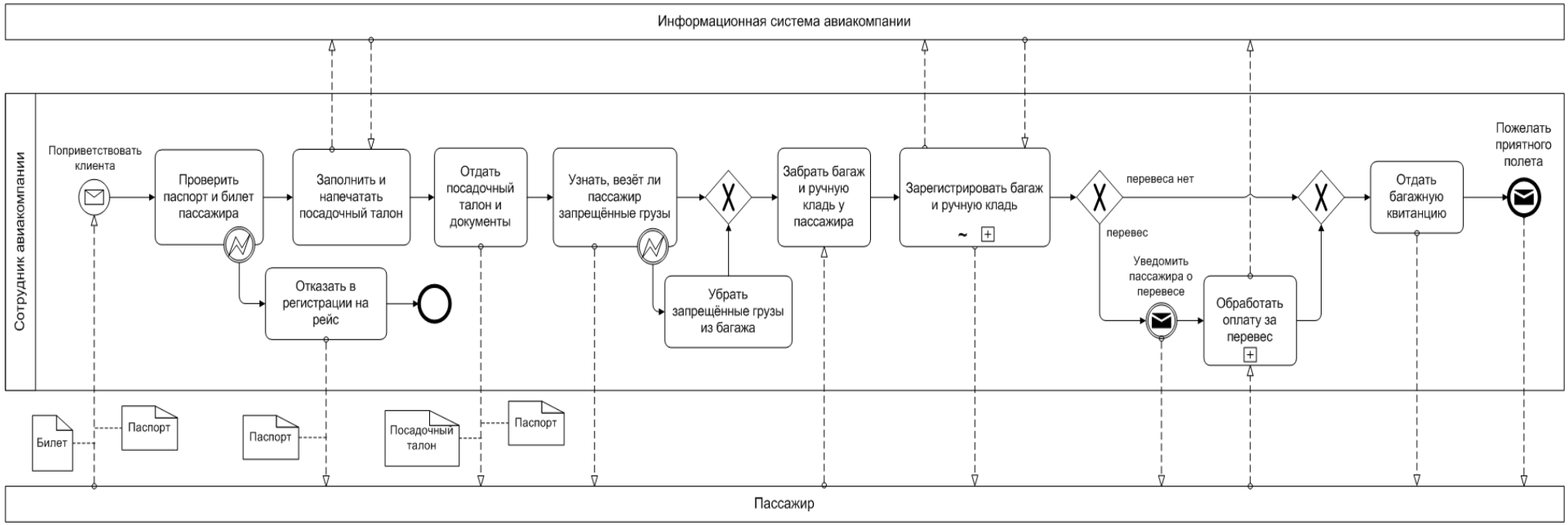


Рисунок 31 - Принцип декомпозиции при построении модели

2 Формирование требований к информационной системе

Цель работы: сформировать практические умения выявлять информационные потребности предприятия, формировать требования к автоматизации работы предприятия на основе моделей текущих бизнес-процессов.

Постановка задачи для самостоятельной работы

Сформировать функциональные требования к будущей информационной системе (в соответствии с вариантом).

Этапы выполнения задания:

1. Определение целей и показателей, на достижение которых направлено внедрение информационной системы.

Необходимо цели внедрения информационной системы и определить показатели достижения этих целей. Для этого необходимо составить таблицу по шаблону (табл. 6).

Таблица 6 - Цели внедрения информационной системы и их показатели

№	Цели	Показатели	Единица измерения показателя	Периодичность фиксации	Желаемый тренд
1					
2					
…					

Например: Основными целями внедрения информационной системы call-центра автосалона является повышение удовлетворенности клиентов и оптимизация расходов на телефонную связь. В таблице 7 представлены показатели достижения этой цели.

Таблица 7 - Цели и показатели бизнес-процессов call-центра

№	Цели	Показатели	Единица измерения показателя	Периодичность фиксации	Желаемый тренд
1	Повышение удовлетворенности клиентов	Процент обработанных вызовов	%	1 раз в месяц	увеличение
		Процент не довольных клиентов	%	1 раз в месяц	уменьшение
2	Оптимизация расходов на телефонную связь	Средняя продолжительность звонка	мин	1 раз в месяц	уменьшение

2. Проектирование новых (усовершенствование старых) бизнес-процессов предприятия, на автоматизацию которых направлена информационная система, – модель TO-BE.

Если внедрение информационной системы на предприятие меняет структуру его текущих бизнес-процессов, то необходимо разработать модель TO-BE. Как правило, модель «как должно быть» изменяет регламент выполнения операций отдельными должностными лицами, поэтому изменения вносятся только на диаграммы нижних уровней.

Для проектирования бизнес-процессов «TO-BE» используются следующие методологии: IDEF0, DFD, IDEF3, EPC.

3. Выделение бизнес-процессов нижнего уровня, подлежащих автоматизации с указанием:

- владельца процесса,
- исполнителя процесса,
- входных данных для реализации процесса,
- результат процесса,
- функция информационной системы, отвечающая за автоматизацию данного бизнес-процесса.

Необходимо внести в таблицу (таблица 8) данные обо всех автоматизируемых операциях.

Таблица 8 - Автоматизируемые операции

№ бизнес-процесса	Название бизнес-процесса	Исполнитель	Входы (данные вносимые в информационную систему)	Выходы (данные и отчеты, генерируемые системой)	Функция информационной системы

Например: Автоматизируемые процессы для корпоративного портала представлены в таблице 9.

Таблица 9 - Бизнес-процессы, реализованные в корпоративном портале предприятия

№ бизнес-процесса	Название бизнес-процесса	Исполнитель	Входы (данные вносимые в информационную систему)	Выходы (данные и отчеты, генерируемые системой)	Функция (модуль) информационной системы
		ров	телефон, пол, возраст, email		
A4.1.1	Авторизация	Сотрудник (любая должность)	логин, пароль		1. Учет сотрудников
A4.1.3	Формирование заявки	Сотрудник (любая должность)	текст заявки, исполнитель заявки	код заявки	2. Управление заявками
A5.2	Отчет о проектах	Директор		отчет о проектах	3. Управление проектами
…					

4. Определение функций будущей информационной системы. При необходимости определение модулей информационной системы и функций каждого из модулей. Необходимо составить таблицу по шаблону (табл. 10).

5.

Таблица 10 - Модули и функции информационной системы

№ п/п	Модули	Функции
1		
2		
…		

Например: Модули и функции информационной системы управления торговлей (табл. 11).

Таблица 11 - Модули и функции информационной системы управления торговлей

№ п/п	Модули	Функции
1	Управление продажами	Учет клиентов
		Учет заказов
		Формирование накладных
		Формирование счетов
2	Управление закупками	Учет поставщиков
		Формирование заказов на поставку
		Формирование актов приемки товаров
…		

5. Формирование характеристик автоматизированных рабочих мест. Необходимо заполнить таблицу по шаблону (таблица 12). Данные компонуются в соответствии с таблицей 8.

Таблица 12 - Автоматизируемые рабочие места

№ п/п	Должность	Процесс	Функция

3 Проектирование базы данных для информационной системы

Цель работы: сформировать практические умения проектировать базы данных.

Постановка задачи для самостоятельной работы

Спроектировать базу данных для информационной системы (в соответствии с вариантом).

Этапы выполнения задания:

1. Анализ требований к базе данных.

На основе пункта 5 из раздела №1, пункта 3 из раздела №2 и пункта 2 из раздела №3 необходимо определить какую информацию необходимо хранить в базе данных.

Здесь необходимо перечислить информационные объекты, с которыми работает система (клиенты, товары, заказы, поставщики, сотрудники, и т.д.). Кроме того, необходимо перечислить основные типы запросы к базе данных.

2. Концептуальное проектирование базы данных (разработка инфологической модели базы данных).

Необходимо:

1) определить сущности, с которыми будет работать информационная система. Для каждой сущности задать уникальное имя, список атрибутов, идентификатор;

2) определить связи между сущностями. Для каждой связи определить степень связи.

Результатом этого этапа должна стать инфологическая модель базы данных. Модель необходимо выполнить в нотации Чена.

3. Выбор системы управления базами данных.

Проанализировав требования к системе необходимо указать, какая СУБД будет выбрана для реализации базы данных и обосновать этот выбор.

4. Датологическое проектирование.

Необходимо нормализовать (минимум до третьей нормальной формы) инфологическую модель. Можно использовать как метод декомпозиции отношений, так и метод синтеза отношений.

Результатом этого этапа должна стать нормализованная логическая модель, выполненная в соответствии с требованиями выбранной СУБД.

5. Физическое проектирование.

На этом этапе необходимо:

1) реализовать базу данных в выбранной СУБД;

2) создать не менее двух запросов к базе данных.

4 Проектирование пользовательского интерфейса системы

Цель работы: сформировать практические умения осуществлять проектирование пользовательского интерфейса информационных систем.

Постановка задачи для самостоятельной работы

Разработать пользовательский интерфейс информационной системы (в соответствии с вариантом) в среде Delphi для работы с базой данных в выбранной СУБД.

Этапы выполнения задания:

Для создания в Delphi программ, работающих с базами данных InterBase/FireBird, можно рекомендовать использовать механизм доступа к БД InterBaseExpress. Другие варианты имеет смысл использовать, только если подразумевается возможность в дальнейшем перевода БД на другую СУБД. InterBaseExpress включает в себя компоненты, располагающиеся на вкладке InterBase палитры компонентов Delphi.

Последовательное выполнение четырех работ, представленных в практикуме, должно привести к созданию целостного проекта будущей информационной системы и, тем самым, формированию у студентов навыков решения следующих профессиональных задач в соответствии с видами профессиональной деятельности:

- проведение обследования прикладной области в соответствии с профилем подготовки;

- моделирование прикладных и информационных процессов;

- формирование требований к информатизации и автоматизации прикладных процессов;

- техническое проектирование ИС в соответствии со спецификой профиля подготовки.

ЗАКЛЮЧЕНИЕ

На стадиях анализа и проектирования закладывается фундамент разрабатываемой системы. Ошибки и просчеты, допущенные на этих стадиях, гораздо труднее исправить, чем ошибки программирования, что, в свою очередь, ведет к серьезным затратам по доведению системы до требуемого уровня функциональности и надежности. Применение единой методологии разработки информационных систем, единого языка описания моделей системы и согласованного набора инструментальных средств призвано обеспечить не только преемственность и автоматизированное использование результатов, получаемых на различных стадиях разработки, но и создание более качественного продукта.

В заключении следует отметить, что в настоящее время наметилась тенденция к объединению в одном инструменте функций CASE-средства и среды разработки программ. Так, последние версии программных продуктов (Delphi, JBuilder, Eclipse, Together Architect, IBM Rational Application Developer и др.) снабжены не только функциями, предназначенными для программистов (написание кода, запуск и отладка программ), но и функциями проектирования и визуализации архитектуры разрабатываемых продуктов. В этих продуктах широко развит инструмент синхронизации моделей системы с исходным кодом программ. Так, любые изменения диаграмм мгновенно приводят к автоматической корректировке кода и наоборот. Таким образом, надежды разработчиков по более тесной интеграции работ и их результатов для стадий проектирования и программирования становятся реальностью.

ЛИТЕРАТУРА

1. Анисимов В.В. Проектирование информационных систем: курс лекций. В 2 ч. Ч.1. Структурный подход / В.В. Анисимов, В.В. Трофимов. - Хабаровск: Изд-во ДВГУПС, 2006. - 112 с.
2. Боггс У. UML и Rational Rose / У. Боггс, М. Боггс. - М.: Издательство «ЛОРИ», 2001. - 582 с.
3. Бучэ Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч. - М.: «Издательство Бином», 2001. - 560 с.
4. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Якобсон. - СПб.: Питер, 2004. - 432 с.
5. Вендров А.М. Case-технологии. Современные методы и средства проектирования информационных систем. - М.: Финансы и статистика, 1998. – 98 с.
6. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: Учебное пособие. - М.: Финансы и статистика, 2006. – 192 с.
7. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник - 2-е изд., перераб. и доп. - М.: Финансы и статистика, 2005. – 192 с.
8. Власов А.И., Лыткин С.Л., Яковлев В.Л. Краткое практическое руководство разработчика информационных систем на базе СУБД Oracle

URL: http://citforum.ru/database/ oraclepr/oraclepr_02.shtml (дата обращения 25.11.2019).

9. Галямина И.Г. Управление процессами - СПб.: Питер, 2013. - 304 с.

10. ГОСТ 24.202-80. Требования к содержанию документа «Технико-экономическое обоснование создания АСУ.

11. ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения.

12. ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем.

13. ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадия создания.

14. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированных систем.

15. ГОСТ 34.603-92. Виды испытаний автоматизированных систем.

16. ГОСТ Р ИСО 15288-2005 Информационная технология. Системная инженерия. Процессы жизненного цикла систем.

17. ГОСТ Р ИСО/МЭК 12207-02. Информационная технология. Процессы жизненного цикла программных средств.

18. ГОСТ Р ИСО/МЭК 12207-99 Информационная технология. Процессы жизненного цикла программных средств.

19. ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.

20. ГОСТ РВ 51987-2002. Информационная технология. Комплекс стандартов на автоматизированные системы. Требования и показатели

качества функционирования информационных систем (ИС). Общие положения.

21. Гвоздева Т.В., Баллод Б.А. Проектирование информационных систем / Т.В. Гвоздева, Б.А. Баллод – Ростов-на-Дону: Феникс, 2009. – 508 с.

22. Гранд М. Шаблоны проектирования в Java / М. Гранд. - М.: Новое знание, 2004. - 559 с.

23. Грекул В.И. Проектирование информационных систем. Интернет-университет информационных технологий / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина - ИНТУИТ.ру, 2005. – 296с. - www.intuit.ru.

24. Ивлев В.А., Попова Т.В., Чекаленко Ю.А. Два подхода к проектированию информационных систем URL: http://consulting.ru/econs_art_417882730 (дата обращения 25.03.2019)

25. Информационные технологии и информационные системы / URL: http://www.itstan.ru/it-i-is (дата обращения 23.03.2019).

26. Каменкова М.С. Моделирование бизнеса. Методология ARIS / М.С. Каменкова, А.И. Громов, М.М. Ферапонтов, А.Е. Шматалюк - М.: Весть-Метатехнология, 2004. – 289 с.

27. Ковалев С.М. Современные методологии описания бизнес-процессов — просто о сложном. Методология ARIS /С.М. Ковалев, В.М. Ковалев // Консультант директора. 2004. № 12. Июнь. URL: http://www.betec.ru/index.php?id=6&sid-33

28. Крачтен Ф. Введение в Rational Unified Process / Ф. Кратчен. - М.: Издательский дом «Вильямс», 2002. - 240 с.

29. Ларман К. Применение UML и шаблонов проектирования: Уч. Пос / К. Ларман. - М.: Издательский дом «Вильямс», 2001. - 496 с.

30. Леоненков А.В. Визуальное моделирование в среде IBM Rational Rose 2003 / А.В. Леоненков. – www.intuit.ru.

31. Леоненков А.В. Объектно-ориентированный анализ и проектирование с использованием UML / А.В. Леоненков. – www.intuit.ru.

32. Леоненков А.В. Самоучитель UML / А.В. Леоненков. – СПб.:

БХВ - Петербург, 2001. – 304с.

33. Магазов С.С. Лекции и практические занятия по технологии баз данных. – М.: КомКнига, 2006. – 112 с.

34. Маклаков С.В. BPwin и ERwin. CASE-средства разработки информационных систем. - 2-е изд., испр. и дополн. - М.: Диолог-МИФИ, 2001.

35. Маклаков С.В. Моделирование бизнес-процессов с AIIFusion Process Modeler. - М.: Диолог-МИФИ, 2004.

36. Марков А.С. Базы данных. Введение в теорию и методологию: учеб. пособие для вузов /. А.С. Марков, К.Ю. Лисовский – М.: Финансы и статистика, 2006. – 512 с.

37. Методология функционального моделирования IDEF0: руководящий документ. URL: http://www.nsu.ru/smk/files/idef.pdf.

38. Официальная документация по программе Business Studio [Электронный ресурс]. URL: http://www.businessstudio.ru/wiki/ docs/current/doku.php/ru/csdesign/csdesign.

39. Петров В.Н. Информационные системы – СПб.: Питер, 2003. – 688 с.

40. Рогозов Ю.И. Системный подход к созданию метода разработки информационных объектов на основе метамоделей // Информатизация и связь. – 2011. – № 7. – С. 57-62.

41. Рогозов Ю.И. Метод быстрого построения корпоративных бизнес приложений / Ю.И. Рогозов., А.С. Свиридов, А.А. Дегтярев // Материалы Второй МНТК «Технологии разработки информационных систем – ТРИС 2011». – Таганрог: Изд-во ТТИ ЮФУ, 2011. – Т. 1. – С. 102-109.

42. Терра-Лексикон: Иллюстрированный энциклопедический словарь. – М.: ТЕРРА, 1998. - 672 с.

43. Фаулер М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: Издательский дом «Вильямс», 2004. – 544 с.

44. Федеральный закон Российской Федерации от 27 июля 2006 г. N 149-ФЗ. Об информации, информационных технологиях и о защите информации.

45. Шемсединов Т.Г. Введение мета-уровня URL: http://blog.meta-systems.com.ua/2011/01/ blog-post.html (дата обращения 30.01.2019).

46. Элиенс А. Принципы объектно-ориентированной разработки программ / А. Элиенс. – М.: Издательский дом «Вильямс», 2002. – 496 с.

47. Якобсон А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. - СПб.: Питер, 2002. - 496 с.

Надежда Владимировна Бендик
Асалханов Петр Георгиевич

Методологии и технологии проектирования информационных систем
Учебное пособие

Лицензия на издательскую деятельность
ЛР № 070444 от 11.03.98 г.
Подписано в печать 10.04.2020 г.
Тираж 30 экз.

Издательство Иркутского государственного аграрного
университета имени А.А. Ежевского
664038, Иркутская обл., Иркутский р-н, пос. Молодежный