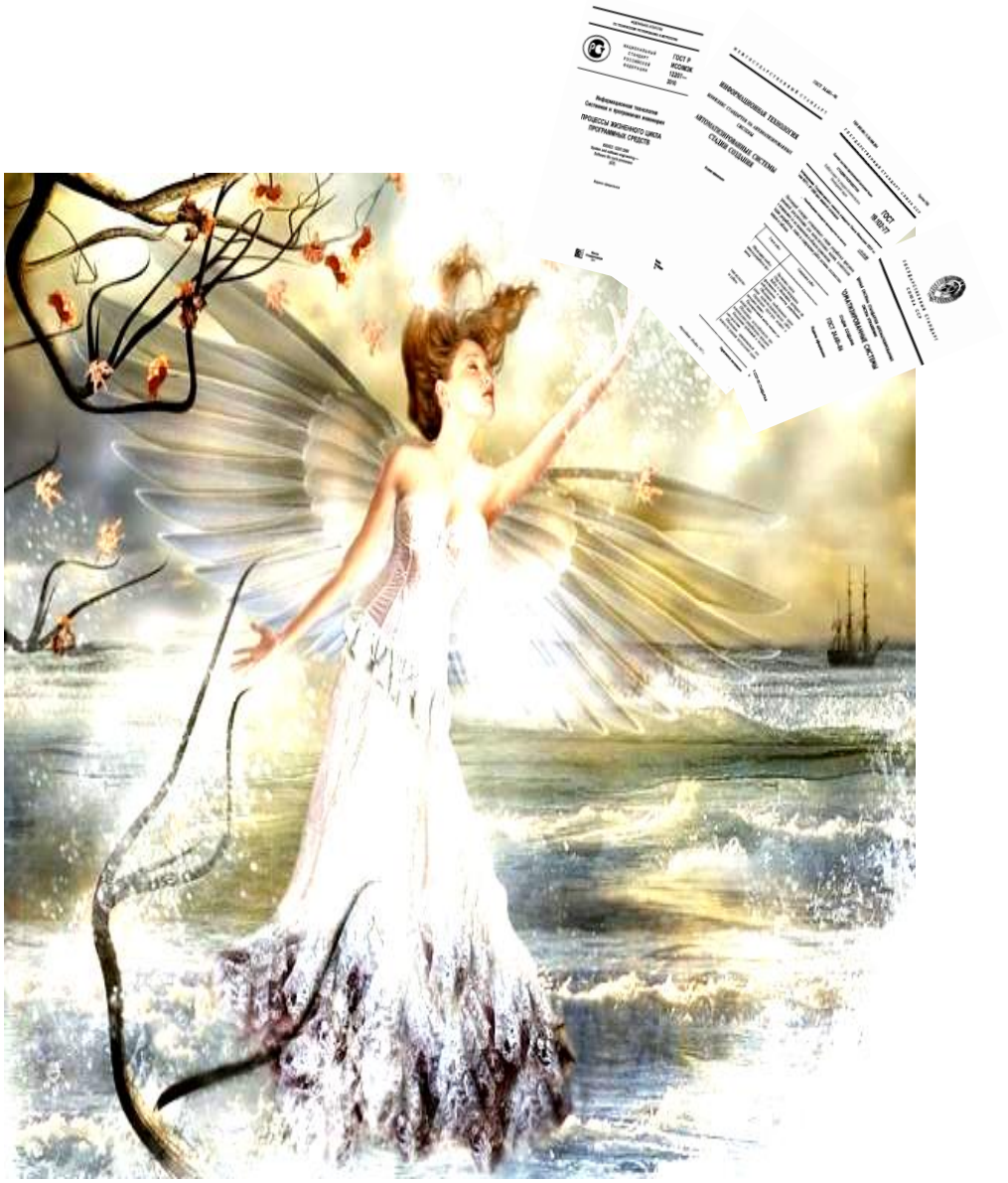
М. А. ИВАНОВСКИЙ, И. А. ГЛАЗКОВА

МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ



Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Тамбовский государственный технический университет»

М. А. ИВАНОВСКИЙ, И. А. ГЛАЗКОВА

МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

Утверждено Ученым советом университета в качестве учебного пособия
для студентов 2, 4 курсов направлений подготовки
09.03.02, 09.04.02 «Информационные системы и технологии»,
27.04.03 «Системный анализ и управление» всех форм обучения

Учебное электронное издание

Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2024

УДК 00.004.78
ББК 39.972.53
И18

Рецензенты:

Доктор технических наук, профессор, заслуженный работник высшей школы РФ, преподаватель цикла интеллектуального управления Межвидового центра подготовки и боевого применения войск РЭБ (учебного и испытательного)
А. В. Зайцев

Доктор технических наук, доцент, заведующий кафедрой «Мехатроника и технологические измерения» ФГБОУ ВО «ТГТУ»
П. В. Балабанов

Ивановский, М. А.

И18 Методы и средства проектирования информационных систем и технологий [Электронный ресурс] : учебное пособие / М. А. Ивановский, И. А. Глазкова. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2024. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод ; 3,0 Mb ; RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана.

ISBN 978-5-8265-2787-0

Содержит общие вопросы проектирования информационных систем, основы канонического проектирования. Рассмотрены CASE-технологии, в том числе, технология объектно-ориентированного проектирования.

Предназначено для студентов 2, 4 курсов направлений подготовки 09.03.02, 09.04.02 «Информационные системы и технологии», 27.04.03 «Системный анализ и управление» всех форм обучения.

УДК 00.004.78
ББК 39.972.53

ISBN 978-5-8265-2787-0

ВВЕДЕНИЕ

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности информационных систем (ИС), создаваемых в различных областях экономики. Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

– сложность описания, требующая тщательного моделирования и анализа данных и процессов;

– наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

– отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;

– необходимость интеграции существующих и вновь разрабатываемых приложений;

– функционирование в неоднородной среде на нескольких аппаратных платформах;

– разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

– существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

Для успешной реализации проекта объект проектирования (ИС) должен быть, прежде всего, адекватно описан, должны быть построены полные и не-

противоречивые функциональные и информационные модели ИС. Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось, в основном, на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

В 1970-х и 1980-х годах при разработке ИС достаточно широко применялась структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания ИС и принимаемых технических решений. Она основана на наглядной графической технике: для описания различного рода моделей ИС используются схемы и диаграммы. Наглядность и строгость средств структурного анализа позволяла разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако, широкое применение этой методологии и следование ее рекомендациям при разработке конкретных ИС встречалось достаточно редко, поскольку при неавтоматизированной (ручной) разработке это практически невозможно. Действительно, вручную очень трудно разработать и графически представить строгие формальные спецификации системы, проверить их на полноту и непротиворечивость, и тем более изменить. Если все же удается создать строгую систему проектных документов, то ее переработка при появлении серьезных изменений практически неосуществима. Ручная разработка обычно порождала следующие проблемы:

- неадекватная спецификация требований;
- неспособность обнаруживать ошибки в проектных решениях;

– низкое качество документации, снижающее эксплуатационные качества;

– затяжной цикл и неудовлетворительные результаты тестирования.

С другой стороны, разработчики ИС исторически всегда стояли последними в ряду тех, кто использовал компьютерные технологии для повышения качества, надежности и производительности в своей собственной работе.

Перечисленные факторы способствовали появлению программно-технологических средств специального класса – CASE-средств, реализующих CASE-технологию создания и сопровождения ИС. Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом. Теперь под термином CASE-средства понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

Появлению CASE-технологии и CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т.д. Кроме того, появлению CASE-технологии способствовали и такие факторы, как:

– подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;

– широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;

– внедрение сетевой технологии, предоставившей возможность объединения усилий отдельных исполнителей в единый процесс проектирования путем использования разделяемой базы данных, содержащей необходимую информацию о проекте.

CASE-технология представляет собой методологию проектирования ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах разработки и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методологиях структурного (в основном) или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

1. ВВЕДЕНИЕ В ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ. ИНФОРМАЦИОННАЯ СИСТЕМА, ПРИЗНАКИ КЛАССИФИКАЦИИ И АРХИТЕКТУРА

Информационная система – автоматизированная система, результатом функционирования которой является представление выходной информации для последующего использования [ГОСТ РВ 51987].

Комплекс понятий, связанных с автоматизированными системами, приведен в [ГОСТ 34.003–90].

Автоматизированная система (АС): система, состоящая из персонала и комплекса средств автоматизации его деятельности, реализующая информационную технологию выполнения установленных функций.

Интегрированная автоматизированная система (ИАС): совокупность двух или более взаимоувязанных АС, в которой функционирование одной из них зависит от результатов функционирования другой (других) так, что эту совокупность можно рассматривать как единую АС.

Комплекс средств автоматизации автоматизированной системы (КСА АС): совокупность всех компонентов АС, за исключением людей.

Компонент автоматизированной системы (компонент АС): часть АС, выделенная по определенному признаку или совокупности признаков и рассматриваемая как единое целое.

Защищаемая автоматизированная информационная система: автоматизированная информационная система, предназначенная для сбора, хранения, обработки, передачи и использования защищаемой информации с требуемым уровнем ее защищенности [Р 50.1.053-2005].

[ISO/IEC 2382-1:1993] дает определение: информационная система – система обработки информации и соответствующие организационные ресурсы (человеческие, технические, финансовые и т.д.), которые обеспечивают и распространяют информацию.

[ФЗ № 149] крайне узко трактует: информационная система – совокупность содержащейся в базах данных информации и обеспечивающих ее обработку информационных технологий и технических средств.

Определение технического понятия «ИТ-система», которое близко к понятию «комплекс средств автоматизации», приведено в [ГОСТ Р ИСО/МЭК ТО 10000-1–99]. Информационно-технологическая система (IT system): набор информационно-технологических ресурсов, обеспечивающий услуги по одному или нескольким интерфейсам.

[ГОСТ Р 53622–2009] определяет: информационно-вычислительная система – совокупность данных (или баз данных), систем управления базами данных и прикладных программ, функционирующих на вычислительных средствах как единое целое для решения определенных задач.

Информационная система предназначена для обеспечения эффективного функционирования объекта управления путем автоматизированного выполнения функций управления.

Степень автоматизации функций управления определяется производственной необходимостью, возможностями формализации процесса управления и должна быть экономически или (и) социально обоснована.

Основными классификационными признаками, определяющими вид информационной системы, являются:

– сфера функционирования объекта управления (промышленность, строительство, транспорт, сельское хозяйство, непромышленная сфера и т.д.)

– вид управляемого процесса (технологический, организационный, экономический и т.д.);

– уровень в системе управления, включения управлением хозяйством.

Функции информационной системы устанавливают в техническом задании на основе анализа целей управления, заданных ресурсов для их достижения, ожидаемого эффекта от автоматизации.

Каждая функция информационной системы реализуется совокупностью комплексов задач, отдельных задач и операций.

Функции информационной системы в общем случае включают в себя следующие элементы (действия):

– планирование и(или) прогнозирование;

– учет, контроль, анализ;

– координацию и(или) регулирование.

Необходимый состав элементов выбирают в зависимости от вида конкретной информационной системы.

Функции информационной системы можно объединять в подсистемы по функциональному и другим признакам.

В состав информационной системы входят следующие виды обеспечений: информационное, программное, техническое, организационное, метрологическое, правовое и лингвистическое. В процессе создания информационной системы используют математическое обеспечение.

В состав информационного обеспечения информационной системы входят классификаторы технико-экономической информации, нормативно-справочная информация, форма представления и организация данных в системе, в том числе формы документов, видеограмм, массивов и логические интерфейсы.

В состав программного обеспечения информационной системы входят программы с программной документацией на них, необходимые для реализации всех функций информационной системы в объеме, предусмотренном в техническом задании.

В состав технического обеспечения информационной системы входят технические средства, необходимые для реализации ее функций. В общем случае оно включает средства получения, ввода, подготовки, обработки, хранения, регистрации, вывода, отображения, использования, передачи информации и средства реализации управляющих воздействий.

В состав организационного обеспечения информационной системы входят документы, определяющие функции подразделений управления, действия и взаимодействие персонала информационной системы.

В состав метрологического обеспечения информационной системы входят метрологические средства и инструкции по их применению.

В состав правового обеспечения информационной системы входят нормативные документы, определяющие правовой статус информационной системы, персонала информационной системы, правил функционирования информационной системы и нормативы на автоматически формируемые документы, в том числе, на машинных носителях информации.

Правовое обеспечение информационной системы в составе функционирующей системы реализуется в виде документов организационного обеспечения информационной системы.

В состав лингвистического обеспечения информационной системы входят тезаурусы и языки описания и манипулирования данными. Лингвистическое обеспечение функционирующей информационной системы может присут-

ствовать в ней самостоятельно или в виде решений по информационному обеспечению и в документах организационного обеспечения.

В состав математического обеспечения информационной системы входят методы решения задач управления, модели и алгоритмы.

В функционирующей системе математическое обеспечение реализовано в составе программного обеспечения.

Структуры информационной системы характеризуют внутреннее строение системы, описывают устойчивые связи между ее элементами.

Возможно использование американских (международных) стандартов планирования производственных процессов (MRP/ERP системы):

1. MRP (Material Requirement Planning) – планирование потребностей в материалах и ресурсах.

2. MRP II (Manufacturing Resource Planning) – планирование производственных ресурсов.

3. ERP (Enterprise Resource Planning) – система планирования ресурсов организации.

4. CSRP (Customer Synchronized Resource Planning) – планирование ресурсов организации, синхронизированное на потребителя.

5. ERP II (Enterprise Resource and Relationship Processing) – управление внутренними ресурсами и внешними связями организации.

В модели MRP/ERP предусматривается сквозное планирование, согласование и оперативная корректировка планов и действий снабженческих, производственных и сбытовых звеньев предприятия.

Подсистема планирования реализует следующие функции:

1. Product Line Planning (PLP) – финансовое планирование товарно-номенклатурных групп (ТНГ);

2. Master Scheduling Planning (MSP) – главный календарный график или объемно-календарное планирование;

3. Distribution Resource Planning (DRP) – планирование распределения ресурсов (RCP);

4. Materials Requirements Planning (MRP) – планирование потребности материалов;

5. Capacity Requirements Planning (CRP) – планирование потребления мощностей.

Типизация информационных систем (ИС) приведена на рис. 1.

Рассматривается 6 типов – TPS, OAS, KWS, DSS, MIS, ESS.

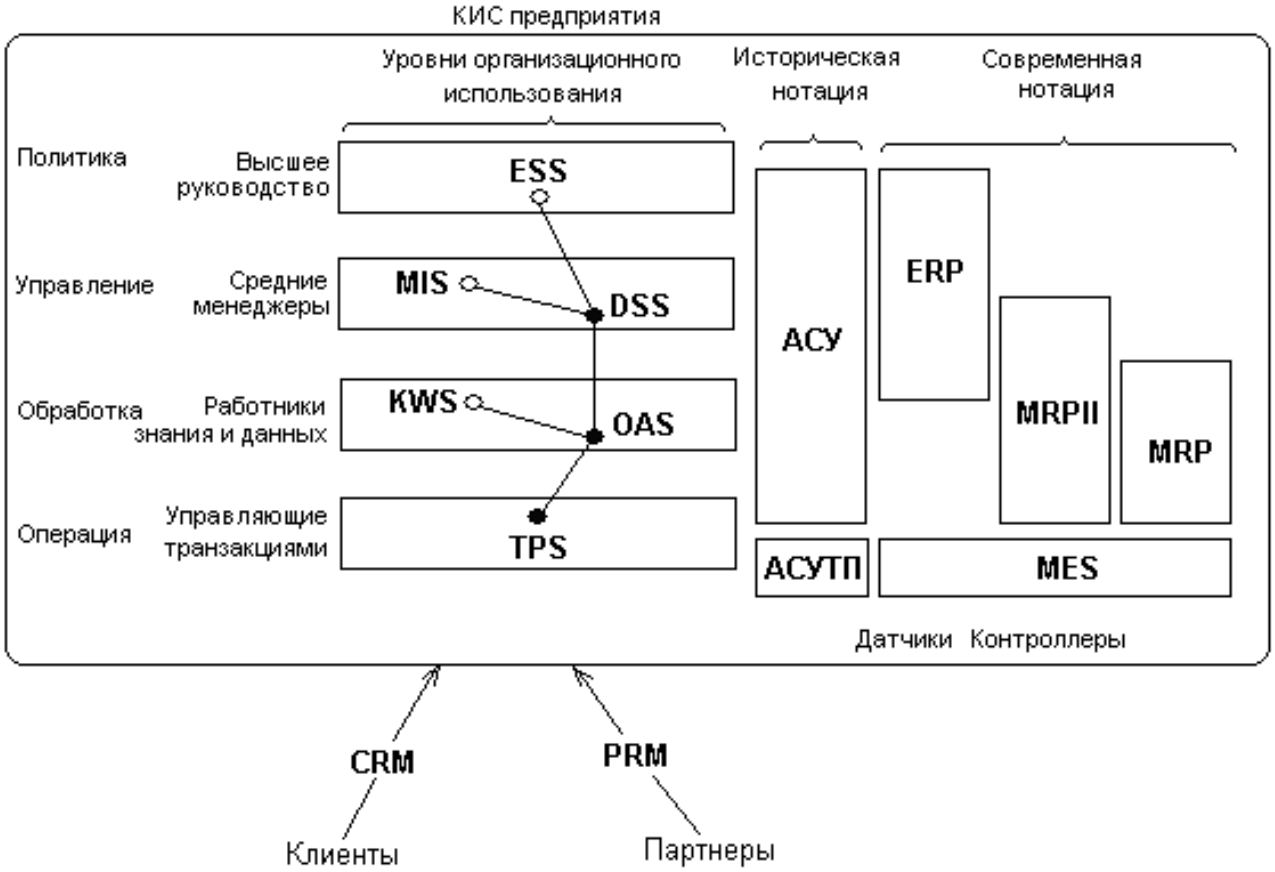


Рис. 1. Типизация информационных систем (ИС)

Системы стратегического планирования – ESS – Executive Support Systems.

Системы поддержки принятия решений – DSS – Decision Support Systems.

Управляющие информационные системы – MIS – Management Information Systems.

Системы работы знания – KWS – Knowledge Work Systems.

Системы автоматизации делопроизводства – OAS – Office Automation Systems.

Системы диалоговой обработки запросов – TPS – Transaction Processing Systems.

АСУТП – Автоматизированная ИС технологического процесса/ технологии производства.

ERP – система планирования/управления ресурсами компании (Enterprice Resource Planning).

MRPII – система планирования производств.ресурсами (Manufactory Resource Planning).

MRP – система планирования материальными ресурсами (Material Resource Planning).

MES – система управления производственными операциями (Management Executive System).

CRM – система управления сделками клиентов (Client Relationship Management).

PRM – система связи с партнерами (Partner Relationship Management).

Таким образом, информационная система представляет собой организационно-техническую систему, обеспечивающую выработку решений на основе автоматизации информационных процессов в различных сферах деятельности или их сочетаниях.

Одним из видов автоматизированных систем является автоматизированная система управления (АСУ). Она предназначена для решения задач управления некоторым объектом. В составе АСУ (АС) иногда выделяют информационно-расчетную и информационно-справочную системы.

Реализация правил содержательной обработки информации в ИС (АС) возлагается на специальное программное обеспечение (СПО), основу которого составляет программная реализация специального математического обеспечения управления.

2. ОСОБЕННОСТИ СОЗДАНИЯ ПРОГРАММНЫХ ИЗДЕЛИЙ

Специальное математическое обеспечение управления и его программная реализация в виде специального программного обеспечения являются неотъемлемой частью любой автоматизированной системы (АС), так как именно оно обеспечивает автоматизированное выполнение процессов обработки информации. В отличие от общего программного обеспечения (ПО), разрабатываемого вне связи с созданием конкретной АС, специальное математическое и программное обеспечение всегда создается в интересах конкретной автоматизируемой системы управления. Таким образом, создание специального математического и программного обеспечения (СМПО) и его «погружение» в вычислительную среду той или иной автоматизированной системы представляет собой неотъемлемый и во многом определяющий компонент процесса автоматизации.

Методология построения средств и комплексов СМПО во многом зависит от уровня автоматизации. Понятие уровней автоматизации служит для обозначения сущности преобразований существующей системы управления в ходе работ по ее автоматизации с позиций преследуемых целей автоматизации и используемых при этом методов и средств.

В общем случае можно выделить следующие пять уровней автоматизации:

1. Технический уровень – установка вычислительной техники. Как правило, вместе с техническими средствами автоматизации устанавливается некоторое типовое программное обеспечение – текстовые и графические редакторы и т.д. Если автоматизация управления ограничивается только техническим уровнем, то речь можно вести фактически только о некоторой автоматизации документооборота в его самой примитивной форме – как процессы изготовления и тиражирования документов машинным способом. Подлинный эффект от автоматизации может быть достигнут только посредством перехода к другим, более высоким уровням.

2. Уровень автономных задач – разработка и внедрение программных средств, для которых характерны следующие особенности:

– ориентация на решение одной или нескольких частных задач, которые не охватывают всю совокупность процессов управления и функций должностных лиц, использующих данные средства;

– решение хорошо формализуемых и структурируемых задач, заключающееся в выполнении расчетов по известным правилам и алгоритмам;

– автономность решения задач, заключающаяся в отсутствии программно реализуемых связей по управлению и по информации (по данным) между отдельными средствами решения различных задач (пара «пользователь – программное средство решения задачи» образует замкнутый автономный контур).

3. Функциональный уровень соответствует созданию и внедрению информационно-справочных и информационно-расчетных систем, относящихся к классу «систем, помогающих принимать решения». При этом степень «помощи» может быть различной – от предоставления справочной информации до прогноза развития ситуаций и выработки рекомендаций по воздействию на объект управления. Для средств данного уровня характерны либо функциональная полнота в обеспечении деятельности некоторого должностного лица (структурного подразделения органа управления), либо ориентация на автоматизированное выполнение отдельно взятых одной или нескольких функций управления. Таким образом, функциональный уровень автоматизации соответствует автоматизации ряда функций управленческой деятельности органов управления или отдельно взятых должностных лиц. Примером автоматизации на функциональном уровне могут служить всевозможные автономно разрабатываемые АРМ различного назначения, а также системы сбора данных обстановки, доведения сигналов и команд управления и т.д.

4. Системно-аналитический уровень проведения работ по автоматизации предполагает выполнение комплексной автоматизации всей системы управления. Он реализуется через создание средств функционального уровня, которые функционально, информационно и технически согласованы между собой. Специфической особенностью данного уровня является автоматизация существующей системы управления без существенных содержательных изменений в ее функциональной и организационной структурах – автоматизируется «то, что есть».

5. Организационно-технологический уровень автоматизации принципиально отличается от системно-аналитического тем, что работы по автоматизации системы управления рассматриваются как составная часть более объемного и глубокого процесса преобразований автоматизируемой системы. Примером подхода к осуществлению работ такого уровня автоматизации является получившая в последнее время широкое распространение методология реинжиниринга бизнес-процессов (Business Process Reengineering (BPR) – направление

деятельности, включающее «фундаментальное переосмысление и радикальное перепланирование критических бизнес-процессов с целью улучшения их эффективности в отношении затрат, качества выполнения и скорости»). При этом под бизнес-процессом понимается любой процесс деятельности организационно-технической системы, направленный на достижение этой системой поставленных перед ней целей. В соответствии с методологией BPR реинжиниринг существующей системы управления завершается созданием и внедрением автоматизированной системы, обеспечивающей эффективное функционирование перепроектированной системы управления.

Вышеперечисленные уровни достаточно сильно взаимосвязаны. В частности, требуемые свойства более высокого уровня являются целеполагающими для более низкого. В обратном направлении подобные связи носят характер ограничений.

Очевидно, что работы по автоматизации четвертого и пятого уровней предполагают создание и внедрение систем автоматизации управления, для обозначения которых в настоящее время более часто применяется термин «информационные системы».

В зависимости от своего масштаба, информационные системы (ИС) подразделяются на одиночные, групповые и корпоративные.

Наиболее сложными и в то же время наиболее полно соответствующими комплексной автоматизации, находящей свое выражение в создании и внедрении АС, являются корпоративные ИС. Именно такие ИС и их СМПО будут являться основным объектом дальнейшего рассмотрения. В составе корпоративных ИС в общем случае выделяют: телекоммуникационную платформу, обеспечивающую обмен информацией между удаленно расположенными узлами вычислительной среды; вычислительную среду, образуемую совокупностью средств вычислительной техники и общего ПО; информационную базу, которую составляют локальные и распределенные базы данных и системы управления ими; совокупность приложений.

Под приложением понимается прикладное программное средство, ориентированное на сбор, хранение, поиск и обработку текстовой и(или) фактографической информации. Подавляющее большинство приложений работает в режиме диалога с пользователем. В общем случае типовые программные компоненты приложения включают: диалоговый ввод-вывод, логику диалога, прикладную логику обработки данных, логику управления данными, операции манипулирования файлами и(или) базами данных, коммуникационный сервис.

В большинстве приложений можно выделить часть, специфичную для конкретной предметной области и решаемой задачи: именно эта часть и представляет собой СМПО. Таким образом, построение СМПО в рамках корпоративной ИС является элементом формирования уникальной части приложений и определяется:

– типом приложения с точки зрения задач управления, решаемых на основе его применения;

– архитектурой корпоративной ИС;

– требованиями, обусловленными спецификой автоматизируемых процессов управления и предъявляемыми к ИС и к ее СМПО.

В настоящее время выделяют следующие типы приложений: Приложения организационного управления. Такие приложения отражают актуальное состояние предметной области (ПрО) в любой момент времени. В ИС с такими приложениями преобладает режим оперативной обработки транзакций OLTP (OnLine Transaction Processing) (под транзакцией понимается неделимый набор операций с БД). Для систем OLTP характерен регулярный поток простых транзакций, оперативно отражающих изменения состояния ПрО. Важными требованиями в этих системах являются высокая производительность обработки транзакций и гарантированная доставка информации при удаленном доступе.

Приложения поддержки принятия решений (DSS – Decision Support System). Эти приложения обеспечивают с помощью сложных запросов отбор и анализ данных в различных аспектах: временном, географическом или по иным показателям. Они характеризуются тем, что извлекают данные из разнородных источников, включая неструктурированные, производят многомерный анализ данных, включают обработку статистики и элементы прогностики и моделирования, в частности анализ «что, если». В общем случае в ИС с приложениями DSS преобладают сложные транзакции и аналитическая обработка. В этом классе приложений выделяются системы оперативной аналитической обработки OLAP (OnLine Analysis Processing), приложения аналитического анализа данных (Data Mining) и разного рода экспертные системы.

Информационно-справочные приложения. Пользовательский интерфейс таких приложений часто строится на принципах гипертекста и представляет собой группу логически связанных текстовых и графических материалов. К этому типу как подтипы могут быть отнесены системы электронной документации, обучающие системы и географические информационные системы (GIS),

а также системы, основанные на использовании гипертекстового представления информации и мультимедиа.

Приложения автоматизации документооборота. Современные системы автоматизации документооборота нацелены на перевод бумажных документов в электронный вид, обеспечение пользователей средствами индексирования и поиска документов. Приложения этого типа могут включать средства коллективной работы с документами, использовать электронную почту, электронные бланки, различные редакторы.

Групповые и корпоративные ИС могут строиться на основе различных архитектур, основными из которых являются:

- многотерминальные централизованные вычислительные системы;
- системы на основе локальной сети;
- системы с архитектурой клиент-сервер;
- системы с распределенными вычислениями;
- офисные системы;
- системы на основе Internet/Intranet-технологий.

Влияние архитектуры на построение приложений заключается в том, что от выбранной архитектуры зависит распределение функциональных компонентов между средствами на автоматизированном рабочем месте пользователя и сервер приложений. Соответственно, разрабатываемые средства СМПО должны включать в себя именно тот набор функциональных компонентов, который определяется распределением функций обработки информации данной архитектуре.

Типовыми функциональными компонентами приложений, реализуемых средствами общего и специального программного обеспечения, являются:

1. PS (Presentation Services) – средства представления. Обеспечиваются средствами, принимающими ввод от пользователя и отображающим то, что сообщает ему компонент логики представления PL,

2. PL (Presentation Logic) – логика представления. Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя по выбору альтернативы меню, по нажатию кнопки или при выборе элемента из списка.

3. BL (Business or Application Logic) – прикладная логика. Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение, т.е. собственно содержательная часть СМПО – программная реа-

лизация математических моделей и методов (алгоритмов) решения задач управления.

4. DL (Data Logic) – логика управления данными. Операции с базой данных, которые нужно выполнить для реализации прикладной логики управления данными.

5. DS (Data Services) – операции с базой данных. Действия системы управления базой данных, вызываемые для выполнения логики управления данными, такие как манипулирование данными, определение данных, фиксация или откат транзакций и т.п.

6. FS (File Services) – файловые операции. Дисковые операции чтения и записи данных, обычно являющиеся функциями операционной системы.

3. ОСНОВНЫЕ СТАНДАРТЫ И РУКОВОДЯЩИЕ ДОКУМЕНТЫ, РЕГЛАМЕНТИРУЮЩИЕ ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Проектирование информационных систем определяется основными системами и комплексами стандартов в области создания АС:

1. Система разработки и постановки продукции на производство ГОСТ Р 15. ХХХ.
2. Информационная технология. Комплекс стандартов на автоматизированные системы ГОСТ 34.ХХХ.
3. Единая система программной документации ГОСТ 19.ХХХ.
4. Единая система конструкторской документации ГОСТ 2.ХХХ.

Система разработки и постановки продукции на производство определена стандартом ГОСТ Р 51189–98 Средства программные систем вооружения. Порядок разработки (Software for military systems The order of creating). Основные ее стандарты:

1. ГОСТ РВ 15.201–2003 Тактико-техническое задание на выполнение опытно-конструкторских работ.
2. ГОСТ РВ 15.203–2001 Порядок выполнения опытно-конструкторских работ по созданию изделий и их составных частей. Основные положения.
3. ГОСТ РВ 15.210–2001 Испытания опытных образцов изделий и опытных ремонтных образцов изделий. Основные положения.
4. ГОСТ РВ 15.211–2002 Порядок разработки программ и методик испытаний опытных образцов изделий. Основные положения.

Требования к построению, содержанию и изложению ТЗ на ОКР определены ГОСТ РВ 15.201–2003. ТЗ на ОКР должно состоять из разделов, располагаемых в следующем порядке:

- наименование, шифр ОКР, основание, исполнитель и сроки выполнения ОКР;
- цель выполнения ОКР, наименование и индекс изделия;
- технические требования к изделию;
- технико-экономические требования;
- требования к видам обеспечения;
- требования к сырью, материалам и КИМП;

– требования к консервации, упаковке и маркировке;

– требования к учебно-тренировочным средствам;

– специальные требования;

– требования защиты государственной тайны при выполнении ОКР;

– этапы выполнения ОКР;

– порядок выполнения и приемки этапов ОКР.

ТЗ на ОКР может быть дополнено приложениями.

В раздел «Технические требования к изделию» включают следующие подразделы:

– состав изделия;

– требования назначения;

– требования радиоэлектронной защиты;

– требования живучести и стойкости к внешним воздействиям;

– требования надежности;

– требования эргономики, обитаемости и технической эстетики;

– требования к эксплуатации, хранению, удобству технического обслуживания и ремонта;

– требования транспортабельности;

– требования безопасности;

– требования обеспечения режима секретности;

– требования зашиты от ИТР;

– требования стандартизации, унификации и каталогизации;

– требования технологичности;

– конструктивные требования.

При необходимости изложения специфических требований допускается вводить и другие подразделы.

При этом предусмотрены стандарты для этапов работ:

– Разработка эскизного проекта – ГОСТ 2.119–73.

– Разработка технического проекта – ГОСТ 2.120–73.

– Разработка рабочей конструкторской документации для изготовления опытного образца изделия – ЕСКД, ЕСПД.

– Изготовление опытного образца изделия (опытного образца СЧ изделия) и проведение предварительных испытаний – ЕСКД, ГОСТ РВ 15.210–2001.

– Проведение государственных испытаний опытного образца изделия (межведомственных испытаний опытного образца СЧ изделия) – ГОСТ РВ 15.210–2001.

– Утверждение рабочей конструкторской документации для организации промышленного (серийного) производства изделий – ГОСТ 2.902–2005.

Эскизный проект разрабатывают с целью установления принципиальных решений изделия, дающих общее представление о принципе работы и(или) устройстве изделия, когда это целесообразно сделать до разработки технического проекта или рабочей документации. Он включает следующие этапы работ:

– выполнение вариантов возможных решений;

– выбор и обоснование оптимального варианта изделия;

– принятие принципиальных решений;

– разработка и анализ макетов с целью проверки принципов работы изделия и(или) его составных частей;

– разработка и обоснование технических решений, направленных на обеспечение показателей надежности.

Технический проект разрабатывают с целью выявления окончательных технических решений, дающих полное представление о конструкции изделия.

Он включает следующие этапы работ:

– разработка конструктивных решений изделия и его основных составных частей;

– выполнение необходимых принципиальных схем, схем соединений и др.

– разработка и обоснование технических решений, обеспечивающих показатели надежности, установленные техническим заданием;

– оценка эксплуатационных данных изделия (удобства обслуживания, ремонтопригодности, возможности быстрого устранения отказов, контроля качества работы изделия, обеспеченность средствами контроля технического состояния и др.);

– выявление номенклатуры покупных изделий.

Предварительные испытания проводят с целью оценки соответствия опытного образца изделия требованиям ТЗ, а также для определения готовности опытного образца изделия к приемочным испытаниям.

Приемочные испытания проводят с целью подтверждения соответствия опытного образца изделия требованиям ТЗ и для определения возможности принятия его в эксплуатацию.

Комплекс стандартов и руководящих документов на автоматизированные системы включает:

– ГОСТ 34.601–90 Автоматизированные системы. Стадии создания;

– ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы;

– ГОСТ 34.201–89 Виды, комплектность и обозначение документов при создании автоматизированных систем;

– РД 50-34.698–90 Автоматизированные системы. Требования к содержанию документов;

– Р 50-34.126–92 Правила проведения работ при создании автоматизированных систем.

4. СТАДИИ И ЭТАПЫ СОЗДАНИЯ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ПО ГОСТ 34.601–90

Процесс создания АС представляет собой совокупность упорядоченных во времени, взаимосвязанных, объединенных в стадии и этапы работ, выполнение которых необходимо и достаточно для создания АС, соответствующей заданным требованиям.

Стадии и этапы создания АС выделяются как части процесса создания по соображениям рационального планирования и организации работ, заканчивающихся заданным результатом.

Работы по развитию АС осуществляют по стадиям и этапам, применяемым для создания АС, приведены в табл. 1.

1. Стадии и этапы создания АС

Стадии	Этапы работ
1. Формирование требований к АС	1.1. Обследование объекта и обоснование необходимости создания АС
	1.2. Формирование требований пользователя к АС
	1.3. Оформление отчета о выполненной работе и заявки на разработку АС (тактико- технического задания)
2. Разработка концепции АС	2.1. Изучение объекта
	2.2. Проведение необходимых научно-исследовательских работ
	2.3. Разработка вариантов концепции АС, удовлетворяющего требованиям пользователя
	2.4. Оформление отчета о выполненной работе
3. Техническое задание	Разработка и утверждение технического задания на создание АС
4. Эскизный проект	4.1. Разработка предварительных проектных решений по системе и ее частям
	4.2. Разработка документации на АС и ее части

Продолжение табл. 1

Стадии	Этапы работ
5. Технический проект	5.1. Разработка проектных решений по системе и ее частям
	5.2. Разработка документации на АС и ее части
	5.3. Разработка и оформление документации на поставку изделий для комплектования АС и(или) технических требований (технических заданий) на их разработку
	5.4. Разработка заданий на проектирование в смежных частях проекта объекта автоматизации
6. Рабочая документация	6.1. Разработка рабочей документации на систему и ее части
	6.2. Разработка или адаптация программ
7. Ввод в действие	7.1. Подготовка объекта автоматизации к вводу АС в действие
	7.2. Подготовка персонала
	7.3. Комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями)
	7.4. Строительно-монтажные работы
	7.5. Пусконаладочные работы
	7.6. Проведение предварительных испытаний
	7.7. Проведение опытной эксплуатации
	7.8. Проведение приемочных испытаний
8. Сопровождение АС	8.1. Выполнение работ в соответствии с гарантийными обязательствами
	8.2. Послегарантийное обслуживание

5. СТАДИИ И ЭТАПЫ СОЗДАНИЯ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ПО ГОСТ 24.601–86

Стандарт распространяется на автоматизированные системы (АС), используемые в различных видах деятельности (исследование, проектирование, управление), включая их сочетания (исследование – проектирование – управление, проектирование – управление), создаваемые в организациях, объединениях и на предприятиях (далее – организациях).

Стандарт устанавливает стадии и этапы создания и развития АС и основные результаты выполнения работ на каждой стадии.

Создание (развитие) АС представляет собой совокупность упорядоченных во времени, взаимно связанных, объединенных в стадии и этапы работ, выполнение которых необходимо и достаточно для создания АС, соответствующей заданным требованиям.

Стадии и этапы работ приведены в табл. 2.

2. Стадии и этапы работ

Стадии	Этапы работ
1. Исследование и обоснование создания АС	1.1. Обследование (сбор и анализ данных) автоматизированного объекта, включая сбор сведений о зарубежных и отечественных аналогах
	1.2. Разработка и оформление требований к системе (технико-экономическое обоснование, тактико-техническое задание, заявка)
2. Техническое задание	2.1. Научно-исследовательские работы*
	2.2. Разработка аванпроекта
	2.3. Разработка технического задания на АС в целом и, при необходимости, частных ТЗ на подсистемы АС
3. Эскизный проект	3.1. Разработка предварительных решений по выбранному варианту АС и отдельным видам обеспечения
4. Технический проект	4.1. Разработка окончательных решений по общесистемным вопросам, в том числе по структурам АС (функциональной, организационной); процедурам (задачам), реализуемым системой; процессу функционирования системы и, при необходимости, выдача частных технических заданий на разработку видов обеспечения АС или видов обеспечения подсистемы АС

Продолжение табл. 2

Стадии	Этапы работ
4. Технический проект	4.2. Разработка решений по организационному обеспечению, включая разработку плана мероприятий по подготовке к внедрению АС
	4.3. Разработка решений по техническому обеспечению
	4.4. Разработка или выбор алгоритмов автоматизируемой деятельности
	4.5. Разработка решений по информационному обеспечению
	4.6. Разработка решений по лингвистическому обеспечению
	4.7. Разработка решений по программному обеспечению
	4.8. Разработка решений по методическому обеспечению
	4.9. Разработка проектно-сметной строительной документации
	4.10. Согласование решений по связям видов обеспечения между собой и разработка общесистемной документации на АС в целом
	4.11. Составление заказной документации на поставляемые компоненты и комплексы средств автоматизации или технических заданий на их разработку
5. Рабочая документация	5.1. Разработка рабочей документации по информационному обеспечению
	5.2. Разработка рабочей документации по организационному обеспечению
	5.3. Разработка рабочей документации по методическому обеспечению
	5.4. Разработка рабочей документации по лингвистическому обеспечению
	5.5. Разработка или адаптация программ и программной документации
	5.6. Разработка документации на технические средства разового изготовления
	5.7. Разработка проектно-сметной строительной документации
6. Изготовление несерийных компонентов комплекса средств автоматизации (КСА)	6.1. Изготовление компонентов КСА
	6.2. Автономная отладка и испытание компонентов КСА

Окончание табл. 2

Стадии	Этапы работ
7. Ввод в действие	7.1. Подготовка организации к вводу АС в действие, обучение персонала пользователя*
	7.2. Строительно-монтажные работы*
	7.3. Комплектация АС* поставляемыми комплексами средств автоматизации, техническими средствами, программными средствами и др.
	7.4. Пуско-наладочные работы* (комплексная отладка КСА)
	7.5. Проведение опытной эксплуатации АС
	7.6. Проведение приемочных испытаний (государственных, межведомственных или ведомственных)
	7.7. Устранение замечаний, выявленных при испытаниях АС
	7.8. Приемка АС в промышленную эксплуатацию (внедрение АС)

Результатом выполнения работ на стадии «Ввод в действие» является приемка АС в промышленную эксплуатацию.

Содержание работ по стадиям и этапам уточнено в ГОСТ 24.602–86 Автоматизированные системы, стадии и этапы создания АСУ. Они приведены в табл. 3.

3. Содержание работ по стадиям и этапам

Состав работ	Содержание работ
Стадия 1. ИССЛЕДОВАНИЕ И ОБОСНОВАНИЕ СОЗДАНИЯ АС	
Этап 1.1. Обследование автоматизируемого объекта	
1.1.1. Подготовка обследования	Ознакомление с исходными материалами и документами по созданию АСУ. Планирование обследования. Организация рабочих групп. Выбор или разработка инструктивно-методических материалов для проведения обследования
1.1.2. Проведение обследования	Сбор и анализ данных о функционировании объекта. Сбор и анализ данных об организационной и производственной структуре объекта управления

Продолжение табл. 3

Состав работ	Содержание работ
1.1.2. Проведение обследования	Сбор и анализ данных о существующей системе управления, включая документооборот. Формулирование основных целей создания АСУ: производственно-хозяйственных, научно-технических и экономических и т.п. Определение степени готовности объекта управления к созданию АСУ. Определение необходимости проведения пред-проектных научно-исследовательских работ (НИР). Сбор и анализ данных о зарубежных и отечествен-ных аналогах
Этап 1.2. Разработка и оформление требований к системе (технико-экономическое обоснование, тактико-техническое задание, заявка)	
1.2.1. Разработка обоснования на создание АСУ	Выбор и обоснование состава процессов, подлежа-щих автоматизации. Предварительный выбор и обоснование состава функций системы. Оценка затрат и предварительный расчет ожидаемой эффективности АСУ. Принятие решений о целесообразности создания АСУ
1.2.2. Разработка требований к АСУ	Определение требований к системе, ее частям и к качеству выполнения автоматизируемых функций управления (характеристики, параметры, показатели назначения и т.п.)
Стадия 2. ТЕХНИЧЕСКОЕ ЗАДАНИЕ	
Этап 2.1. Научно-исследовательские работы*	
2.1.1. Подготовка НИР	Определение направлений предпроектных НИР. Составление и утверждение технического задания (ТЗ) на НИР
2.1.2. Проведение НИР	Выполнение НИР в соответствии с ТЗ. Разработка рекомендаций по использованию результатов проведенных НИР
2.1.3. Оформление результатов НИР	Составление и оформление отчета о НИР

Продолжение табл. 3

Состав работ	Содержание работ
Этап 2.2. Разработка аванпроекта*	
2.2.1. Предварительная разработка проектных решений	Разработка вариантов функциональной структуры АСУ. Разработка вариантов структур АСУ по видам обеспечения. Сравнительная технико-экономическая оценка рассматриваемых вариантов. Выбор типовых проектных решений по видам обеспечения АСУ
Этап 2.3. Разработка технического задания на АСУ	
2.3.1. Разработка требований к АСУ	Уточнение целей создания АСУ. Укрупненное описание функциональной структуры АСУ. Уточнение состава автоматизируемых функций. Уточнение требований к качеству выполнения автоматизируемых функций управления. Формулирование требований к временному регламенту решения задач (комплексов задач) и их классов. Формулирование требований к частям АСУ и видам обеспечения АСУ. Предварительный выбор состава средств вычислительной техники. Определение перечня задач (комплексов задач), обеспечивающих реализацию автоматизируемых функций управления
2.3.2. Определение (при необходимости) состава НИР, подлежащих выполнению на последующих стадиях создания АСУ	–
2.3.3. Определение порядка проведения работ по созданию АСУ	Определение очередей создания АСУ. Определение состава стадий и этапов создания АСУ. Определение организации-исполнителей. Разработка плана-графика создания АСУ. Разработка плана организационно-технических мероприятий по подготовке объекта управления к вводу в действие АСУ

Продолжение табл. 3

Состав работ	Содержание работ
2.3.4. Разработка частных ТЗ на подсистемы АСУ и виды обеспечений	
Стадия 3. ЭСКИЗНЫЙ ПРОЕКТ	
Этап 3.1. Разработка предварительных решений по выбранному варианту АСУ и отдельным видам обеспечений*	
Состав и содержание работ соответствует составу и содержанию работ стадии 4.	
Стадия 4. ТЕХНИЧЕСКИЙ ПРОЕКТ	
Этап 4.1. Разработка окончательных решений по общесистемным вопросам	
4.1.1. Разработка функциональной структуры АСУ	Уточнение состава задач (комплексов задач), обеспечивающих реализацию автоматизируемых функций управления. Определение состава операций, задач, функций, выполняемых в автоматическом режиме. Разработка общего алгоритма функционирования АСУ
4.1.2. Разработка проектных решений по системе в целом	Выбор и обоснование принципиальных проектных решений по структурам АСУ. Разработка проектных решений по совместимости АСУ со смежными системами. Проведение (при необходимости) патентных исследований. Определение состава требований к техническим и программным средствам системы передачи данных, которые должны использоваться в АСУ. Определение состава средств и элементов системы передачи данных (например центры коммутации сообщений, аппаратура передачи данных, каналы передачи данных, центры коммутации каналов, концентраторы нагрузки и т.п.). Определение структуры системы передачи данных и методов управления (пакетный, диалоговый режим, режим реального времени)

Продолжение табл. 3

Состав работ	Содержание работ
4.1.3. Разработка постановок задач	–
4.1.4. Разработка регламента функционирования	–
4.1.5. Разработка и выдача частных ТЗ на проектирование обеспечений АСУ, частей АСУ	–
4.1.6. Разработка плана организационно-технических мероприятий по подготовке объекта управления к вводу в действие АСУ	–
Этап 4.2. Разработка решений по организационному обеспечению	
4.2.1. Разработка решений по функциям персонала АСУ	Определение функций персонала АСУ. Формирование требований к квалификации персонала АСУ. Разработка предложений по подготовке персонала АСУ
4.2.2. Разработка решений по организационной структуре АСУ	Разработка решений по организационной структуре объекта управления в условиях функционирования АСУ. Разработка (уточнение) штатных расписаний подразделений, обеспечивающих функционирование АСУ
4.2.3. Разработка решений по правовому обеспечению АСУ	Определение статуса АСУ, правового положения структурных подразделений и должностных лиц АСУ. Формулирование правовых положений, определяющих технологический процесс обработки информации в АСУ. Определение порядка получения, накопления в использования информации

Продолжение табл. 3

Состав работ	Содержание работ
Этап 4.3. Разработка решений по техническому обеспечению*	
4.3.1. Разработка решений по составу технического обеспечения	Выбор и обоснование состава и структуры комплекса технических средств АСУ, в том числе комплекса средств автоматизации. Определение перечня серийно-выпускаемых технических средств. Определение технических средств АСУ, подлежащих разработке, технических требований к ним и составление заявок на их разработку. Проектная оценка надежности комплекса технических средств АСУ
4.3.2. Разработка (при необходимости) заданий на проектирование зданий, сооружений, помещений, в том числе смежных частях проекта объекта строительства	–
Этап 4.4. Разработка или выбор алгоритмов автоматизируемой деятельности*	
4.4.1. Разработка (выбор) алгоритмов решения задач	Анализ возможности использования готовых алгоритмов с учетом требований оптимизации. Выбор методов. Синтез моделей. Синтез алгоритмов
Этап 4.5. Разработка решений по информационному обеспечению	
4.5.1. Разработка решений по информационной базе	Определение состава и объема нормативно-справочной информации. Разработка предложений по совершенствованию действующего документооборота. Разработка структуры базы данных. Разработка системы сбора и передачи информации. Разработка решений по организации и ведению базы данных. Определение состава и характеристик входной и выходной информации (сигналов, документов, данных)

Продолжение табл. 3

Состав работ	Содержание работ
4.5.2. Выбор номенклатуры и привязка систем классификации и кодирования информации	Определение перечня типов информационных объектов, подлежащих идентификации в АСУ. Определение перечня необходимых классификаторов и кодовых словарей. Выбор и разработка классификаторов информационных объектов и систем кодирования для кодовых словарей. Определение системы внесения изменений и дополнений в классификаторы. Разработка принципов алгоритмов автоматизированного ведения классификаторов
4.5.3. Разработка решений по обеспечению обмена информацией в системе	Разработка схемы информационного обеспечения
Этап 4.6. Разработка решений по лингвинистическому обеспечению*	
4.6.1. Определение терминологической предметной области	Определение требований (ориентации) пользователей. Определение лексического состава языковых структур. Разработка терминологических словарей
4.6.2. Выбор языков	Установление принципов совместимости используемых языковых средств. Определение структур языков. Определение ограничений на языковые средства
Этап 4.7. Разработка решений по программному обеспечению	
4.7.1. Определение основных решений по программному обеспечению асу	Выбор принципов построения программного обеспечения. Разработка структуры программного обеспечения. Разработка протоколов обмена информацией в системе передачи данных (транспортного, сетевого уровня, уровня канала, физического уровня)

Продолжение табл. 3

Состав работ	Содержание работ
4.7.2. Определение состава программного обеспечения	Определение состава общего программного обеспечения АСУ. Определение состава специального программного обеспечения АСУ, включая выбор базовых средств: операционной системы, системы управления базами данных, терминальной обработки, тестирования, управления учета и разграничения доступа. Выбор пакетов прикладных программ (ППП). Разработка (при необходимости) ТЗ на программные средства, не поставляемые в составе КСА
Этап 4.8. Разработка решений по методическому обеспечению*	
Этап 4.9. Разработка проектно-сметной строительной документации*	
4.9.1. Разработка проектно-сметной документации на АСУ	Содержание работ определяется стандартами СПДС
Этап 4.10. Согласование решений по связям видов обеспечения между собой и разработка общесистемной документации на АСУ в целом	
4.10.1. Окончательная увязка проектных решений	–
4.10.2 Расчет затрат на создание АСУ и уточнение ее технико-экономической эффективности	–
4.10.3. Разработка общесистемной документации	–
Этап 4.11. Составление заказной документации на компоненты и комплексы средств автоматизации или технических заданий на их разработку*	
4.11.1. Подготовка заказной документации на КСА серийного изготовления	–

Продолжение табл. 3

Состав работ	Содержание работ
4.11.2. Подготовка заказной документации на технические и программные средства длительного изготовления	–
4.11.3. Определение технических требований и составление ТЗ на разработку КСА, технических и программных средств, не изготавливаемых серийно	–
Стадия 5. РАБОЧАЯ ДОКУМЕНТАЦИЯ	
Этап 5.1. Разработка рабочей документации по информационному обеспечению	
5.1.1. Разработка технологического процесса обработки данных	Разработка технологического процесса получения данных. Разработка технологического процесса обработки данных на вычислительных и других технических средствах
5.1.2. Разработка эксплуатационной документации по информационному обеспечению	Разработка унифицированных форм документов. Подготовка классификаторов
5.1.3. Проверка информационно-логической структуры базы данных	
Этап 5.2. Разработка рабочей документации по организационному обеспечению	
5.2.1. Уточнение функций и конкретизация состава работ персонала АСУ	Разработка положений и инструкций всех видов, формуляра системы
Этап 5.3. Разработка рабочей документации по методическому обеспечению*	
Этап 5.4. Разработка рабочей документации по лингвистическому обеспечению*	

Продолжение табл. 3

Состав работ	Содержание работ
Этап 5.5. Разработка или адаптация программ и программной документации	
5.5.1. Адаптация ППП и отдельных программ	Освоение и привязка ППП (СУБД, ИПС, пакетов функционального назначения и т.д.)
5.5.2. Разработка программ и программных средств	Разработка программ. Подготовка (при необходимости) стенда для отладки программ. Отладка программ. Разработка программных средств КСА. Разработка контрольных примеров для испытаний программ и программных средств. Разработка программной документации, в том числе эксплуатационной
Этап 5.6. Разработка документации на технические средства разового изготовления*	
5.6.1. Разработка конструкторской документации на технические средства разового изготовления	Разработка документации в соответствии со стандартами ЕСКД
Этап 5.7. Разработка проектно-сметной строительной документации	
5.7.1. Разработка проектно-сметной документации	Разработка документации в соответствии со стандартами СПДС
5.7.2. На основании проектной документации производится разработка заказной документации на технические средства АСУ	–
Стадия 6. ИЗГОТОВЛЕНИЕ НЕСЕРИЙНЫХ КОМПОНЕНТОВ КСА	
Этап 6.1. Изготовление компонентов КСА	
6.1.1. Технологическая подготовка производства	Технологический контроль технической документации на технические и программные средства КСА. Определение состава средств автоматизации программирования. Разработка (при необходимости) технологической документации для изготовления технических средств

Продолжение табл. 3

Состав работ	Содержание работ
6.1.2. Комплектация	–
6.1.3. Изготовление компонентов КСА	Изготовление компонентов программных средств. Изготовление компонентов технических средств
Этап 6.2. Автономная отладка и испытания компонентов КСА	
6.2.1. Автономная отладка компонентов КСА	Отладка компонентов в соответствии с эксплуатационной документацией
6.2.2. Организация испытаний	–
6.2.3. Проведение испытаний	Испытание программных средств. Испытание технических средств. Принятие решения о пригодности компонентов КСА к поставке
Стадия 7. ВВОД В ДЕЙСТВИЕ	
Этап 7.1. Подготовка организации к вводу АСУ в действие, обучение персонала пользователя	
7.1.1. Обучение персонала пользователя	Организация и проведение обучения пользователей и обслуживающего персонала АСУ
7.1.2. Проведение мероприятий по подготовке к вводу АСУ	Реализация проектных решений по организационной структуре АСУ. Обеспечение подразделений объекта управления инструктивно-методическими материалами. Внедрение классификаторов и кодификаторов информации
Этап 7.2. Комплектация АСУ	
7.2.1. Получение комплектующих изделий серийного изготовления	Получение технических средств. Получение программных средств. Получение ЗИП
7.2.2. Получение и входной контроль комплектующих изделий единичного производства	–

Продолжение табл. 3

Состав работ	Содержание работ
7.2.3. Получение материалов и монтажных изделий	Получение материалов от Генподрядчика. Комплектование материалов и изделий поставки монтажных организаций
Этап 7.3. Строительно-монтажные работы*	
7.3.1. Выполнение строительных работ	Строительство (реконструкция) специализированных зданий (помещений) для размещения технических средств и персонала АСУ. Сооружение кабельных каналов. Сдача-приемка помещений
7.3.2. Выполнение монтажных работ	Выполнение работ по монтажу технических средств и линий связи. Испытание смонтированных технических средств. Сдача технических средств для проведения пусконаладочных работ
Этап 7.4. Пуско-наладочные работы (комплексная отладка КСА)	
7.4.1. Наладка технических и программных средств	Наладка технических средств АСУ. Комплексная наладка технических средств АСУ. Постановка и отладка программных средств
7.4.2. Подготовка АСУ к опытной эксплуатации	Завершение выполнения организационно-технических мероприятий по подготовке объекта управления к эксплуатации АСУ. Загрузка информации в базу данных. Проведение опытной эксплуатации систем ведения информационной базы. Разработка программы проведения опытной эксплуатации АСУ
7.4.3. Проведение предварительных испытаний АСУ	Составление программы и методики предварительных испытаний. Проведение испытаний АСУ на работоспособность. Корректировка эксплуатационной документации в соответствии с протоколом испытаний. Устранение неисправностей и внесение изменений в документацию. Выпуск организационно-распорядительной документации о приемке АСУ в опытную эксплуатацию

Окончание табл. 3

Состав работ	Содержание работ
Этап 7.5. Проведение опытной эксплуатации АСУ	
7.5.1. Проведение опытной эксплуатации АСУ	Проведение опытной эксплуатации АСУ. Анализ результатов опытной эксплуатации АСУ
7.5.2. Подготовка АСУ к приемочным испытаниям (государственным, межведомственным, ведомственным)	Доработка (при необходимости) программного обеспечения АСУ. Дополнительная наладка (при необходимости) технических средств АСУ. Корректировка документации по результатам опытной эксплуатации АСУ. Расчет экономической эффективности АСУ по результатам опытной эксплуатации. Подготовка программ и методик приемочных испытаний АСУ и выпуск организационно-распорядительной документации
Этап 7.6. Проведение приемочных испытаний	
7.6.1. Проведение приемочных испытаний	–
Этап 7.7. Устранение замечаний, выявленных при испытаниях	
7.7.1. Анализ результатов испытаний АСУ	–
7.7.2. Устранение замечаний по результатам испытаний АСУ	–
7.8. Приемка АСУ в промышленную эксплуатацию	
7.8.1. Оформление акта о вводе АСУ в действие	–

6. СТАДИИ РАЗРАБОТКИ, ЭТАПЫ И СОДЕРЖАНИЕ РАБОТ ПО ГОСТ 19.102–77

Стандарт устанавливает стадии разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Стадии разработки, этапы и содержание работ должны соответствовать указанным в табл. 4.

4. Стадии разработки, этапы и содержание работ

Стадии разработки	Этапы работ	Содержание работ
1. Техническое задание	Обоснование необходимости разработки программы	Постановка задачи. Сбор исходных материалов. Выбор и обоснование критериев эффективности и качества разрабатываемой программы. Обоснование необходимости проведения научно-исследовательских работ
	Научно-исследовательские работы	Определение структуры входных и выходных данных. Предварительный выбор методов решения задач. Обоснование целесообразности применения ранее разработанных программ. Определение требований к техническим средствам. Обоснование принципиальной возможности решения поставленной задачи
	Разработка и утверждение технического задания	Определение требований к программе. Разработка технико-экономического обоснования разработки программы. Определение стадий, этапов и сроков разработки программы и документации на нее. Выбор языков программирования. Определение необходимости проведения научно-исследовательских работ на последующих стадиях. Согласование и утверждение технического задания

Продолжение табл. 4

Стадии разработки	Этапы работ	Содержание работ
2. Эскизный проект	Разработка эскизного проекта	Предварительная разработка структуры входных и выходных данных. Уточнение методов решения задачи. Разработка общего описания алгоритма решения задачи. Разработка технико-экономического обоснования
	Утверждение эскизного проекта	Разработка пояснительной записки. Согласование и утверждение эскизного проекта
3. Технический проект	Разработка технического проекта	Уточнение структуры входных и выходных данных. Разработка алгоритма решения задачи. Определение формы представления входных и выходных данных. Определение семантики и синтаксиса языка. Разработка структуры программы. Окончательное определение конфигурации технических средств
	Утверждение технического проекта	Разработка плана мероприятий по разработке и внедрению программ. Разработка пояснительной записки. Согласование и утверждение технического проекта
Стадии разработки	Этапы работ	Содержание работ
4. Рабочий проект	Разработка программы	Программирование и отладка программы
	Разработка программной документации	Разработка программных документов в соответствии с требованиями ГОСТ 19.101–77
	Испытания программы	Разработка, согласование и утверждение программы и методики испытаний. Проведение предварительных государственных, межведомственных, приемо-сдаточных и других видов испытаний. Корректировка программы и программной документации по результатам испытаний

Продолжение табл. 4

Стадии разработки	Этапы работ	Содержание работ
5. Внедрение	Подготовка и передача программы	Подготовка и передача программы и программной документации для сопровождения и(или) изготовления. Оформление и утверждение акта о передаче программы на сопровождение и(или) изготовление. Передача программы в фонд алгоритмов и программ

Допускается исключать вторую стадию разработки, а в технически обоснованных случаях – вторую и третью стадии. Необходимость проведения этих стадий указывается в техническом задании.

Допускается объединять, исключать этапы работ и(или) их содержание, а также вводить другие этапы работ по согласованию с заказчиком.

7. ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И СРЕДСТВА ЕГО ПОДДЕРЖКИ

7.1. ОБЩАЯ СХЕМА И ПОНЯТИЕ ЖИЗНЕННОГО ЦИКЛА ПО

Одним из базовых понятий методологии проектирования сложных программных систем (ПС) является понятие жизненного цикла их программного обеспечения (ЖЦ ПО). ЖЦ ПО – это непрерывный процесс, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент его полного изъятия из эксплуатации (табл. 5).

5. Группы процессов ЖЦ ПО по стандарту Р ИСО/МЭК 12207

Основные процессы ЖЦ ПО	Вспомогательные процессы	Организационные процессы
– приобретение	– документирование	– управление проектами
– поставка	– управление конфигурацией	– планирование и организация работ
– разработка	– управление версиями	– создание коллективов разработчиков
– анализ	– конфигурационный учет	– контроль за сроками и качеством выполняемых работ:
– проектирование	– планирование	– верификация
– программирование		– проверка
– оформление документации:		– тестирование
– проектной		– создание инфраструктуры проекта
– эксплуатационной		– определение, оценка и улучшение самого ЖЦ
– подготовка тестов		– обучение
– подготовка материалов для обучения		
– эксплуатация		

Продолжение табл. 5

Основные процессы ЖЦ ПО	Вспомогательные процессы	Организационные процессы
– внедрение компонентов ПО:		
– конфигурирование		
– базы данных		
– рабочих мест пользователей		
– обеспечение эксплуатационной документацией		
– проведение обучения персонала		
– локализация проблем		
– устранение причин возникновения проблем		
– модификация ПО в рамках установленного регламента		
– подготовка предложений по совершенствованию, развитию и модернизации системы		
– сопровождение		

Наряду с понятием ЖЦ широко используется понятие модели жизненного цикла, под которой понимается формализованное описание процессов, составляющих ЖЦ. Если подходить более строго, можно считать, что ЖЦ – это процесс существования конкретного СПО, а модель ЖЦ – обобщенное описание таких процессов относительно либо конкретного СПО, либо абстрактной совокупности различных реализаций СПО. Однако на практике такое различие обычно не делается, данные термины используются как синонимы, причем под ЖЦ в абсолютном большинстве случаев понимается именно его модель.

Относительно ЖЦ СПО необходимо учитывать следующее: в настоящее время имеется достаточно много трактовок содержания ЖЦ СПО, зачастую принципиально отличающихся друг от друга, в том числе, за счет учета специфики рассматриваемого СПО.

В основе жизненного цикла лежит та или иная схема жизненного цикла, фиксирующая последовательность смены стадий и этапов ЖЦ и соответствующих им состояний СПО, жизненный цикл СПО неразрывно связан с ЖЦ АСУ и АС, имеется ряд стандартов, регламентирующих содержание ЖЦ СПО.

Основными стандартами, регламентирующими содержание ЖЦ ПО в части создания ПП, являются:

– международный стандарт Р ИСО/МЭК 12207 (ISO/IEC 12207. ISO – International Organization of Standardization – Международная организация по стандартизации, IEC – International Electrotechnical Commission – Международная комиссия по электротехнике);

– ГОСТ ЕСПД 19.102–77 «Стадии разработки».

Они определяют структуру жизненного цикла, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

Применительно к СПО АСУ необходимо также учитывать стандарты, определяющие стадии создания АСУ и АС:

– ГОСТ 24.602–86 «АСУ. Состав и содержание работ по стадиям создания»;

– ГОСТ 34.601–90 «Информационная технология. Комплекс стандартов на АС. Автоматизированная система. Стадии создания». Кроме того, организациями-производителями СПО могут применяться отраслевые стандарты и стандарты предприятий, уточняющие и дополняющие национальные стандарты применительно к организации создания ПО определенного назначения и(или) в определенных условиях.

В качестве примера стандарта предприятия можно привести методику Oracle CDM (Custom Development Method) по разработке прикладных информационных систем под заказ, разработанную фирмой Oracle.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов (рис. 2):

1) основные процессы ЖЦ ПО:

– приобретение;

– поставка;

– разработка;

– эксплуатация;

– сопровождение;

2) вспомогательные процессы, обеспечивающие выполнение основных процессов;

– документирование;

– управление конфигурацией;

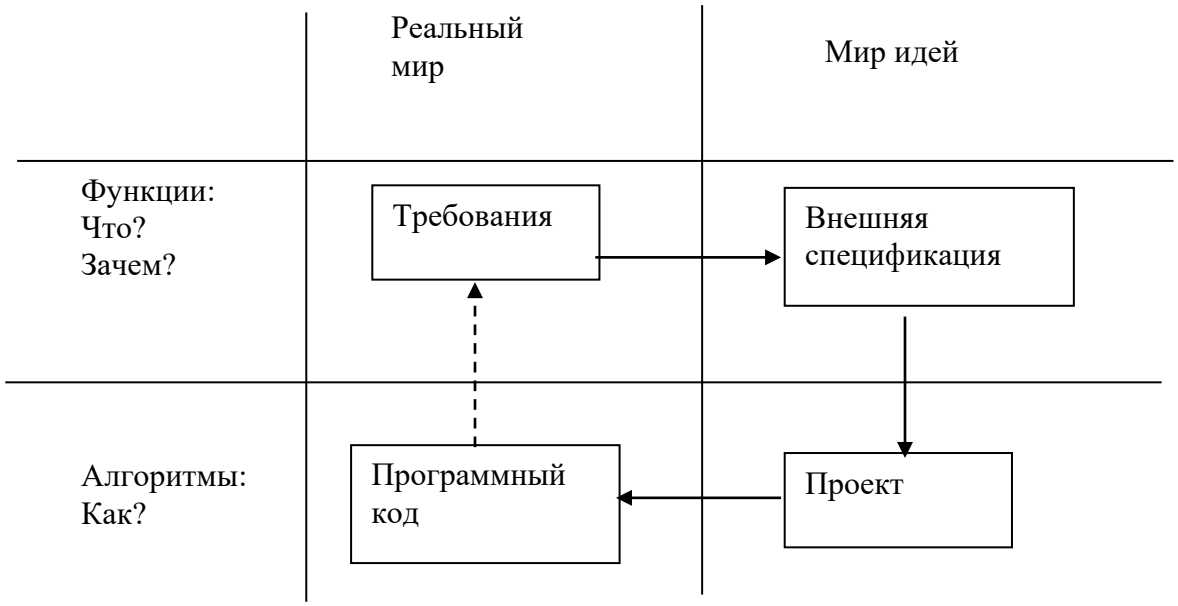


Рис. 2. Жизненный цикл как путь из реального мира в мир абстракций и обратно

3) организационные процессы:

– управление проектами;

– создание инфраструктуры проекта;

– определение, оценка и улучшение самого ЖЦ;

– обучение.

Обобщенный жизненный цикл можно представить в виде следующей последовательности этапов, которые, в свою очередь, можно также разбить на стадии:

1. Планирование разработки.
2. Определение требований к системе:

– выработка требований;

– анализ требований.

3. Проектирование системы:

– проектирование архитектуры системы;

– детальное проектирование компонент системы, в том числе для программного обеспечения;

– общее проектирование программного обеспечения;

– проектирование отдельных программных компонент.

4. Реализация и тестирование системы:

– создание отдельных компонент системы, в том числе, для программного обеспечения;

– создание отдельных программных модулей;

– тестирование отдельных программных модулей;

– тестирование компонент системы, в том числе, программного обеспечения как единого компонента системы.

5. Интегрирование отдельных компонент в систему.
6. Выпуск системы.
7. Эксплуатация системы.
8. Завершение разработки.

Рассмотрим описание жизненного цикла программного обеспечения, которое послужит комментарием к этапам обобщенного жизненного цикла (рис. 1).

В отечественных нормативных документах (например, ГОСТ ЕСПД) принято следующее разграничение на этапы:

– разработка технического задания (этапы 1 и 2);
– технический проект (третий этап до 3.2.1 включительно);
– рабочий проект (3.2.2,4.2.1 и, частично, 4.2,4.3);
– экспериментальное внедрение (4.2 и 4.3);
– сдача в промышленную эксплуатацию (этап 5);
– промышленная эксплуатация (этап 6).

Подобное описание имеет своим прообразом технологию разработки аппаратных средств и поэтому не вполне учитывает все отличительные особенности проектирования программ. Более подходящим выглядит описание жизненного цикла программного обеспечения, состоящее из 12 этапов, которые очень близки этапам обобщенного жизненного цикла (табл. 6). В скобках после имени фазы указывается аналог из обобщенного цикла. Практически все этапы закачиваются проверкой результатов, полученных на соответствующем этапе.

На рисунке 4 показана последовательность этапов разработки программного обеспечения для отдельных компонентов единой программной системы с различными жизненными циклами.

Для компонента W из множества системных требований к единому продукту формируется подмножество требований, относящихся к данному компоненту, используются эти требования при формировании проекта программного компонента, реализовывают этот проект в исходном коде и тогда интегрирует компонент с аппаратурой. Компонент X показывает использование ранее разработанного программного обеспечения. Компонент Y показывает использование простой отдельной функции, которая может быть закодирована прямо на основе требований к программному обеспечению. Компонент Z показывает

использование прототипной стратегии. Обычно целями прототипирования является лучшее понимание требований к программному обеспечению и уменьшение технических рисков и рисков разработки при создании конечного продукта. Исходные требования используются как базис для получения прототипа. Этот прототип преобразуется в окружение, типичное для конкретного использования системы при разработке. Результатом преобразований является уточненные данные, которые используются для создания конечного программного продукта.

6. Пример жизненного цикла программных систем

Основные этапы	Основные действия		Проверки и контроль
Начало проекта и планирование			
Анализ целевых требований	Определение требований		Проверка целевых требований
Анализ системных требований		Инжиринг системы	Проверка системных требовании
Проектирование системы			Проверка проекта системы
Предварительное проектирования ПО	Проектирование		Проверка предварительного проектирования ПО
Детальное проектирование ПО		Разработка компонент	Проверка детального проектирования ПО
Кодирование и тестирование ПО			Проверка аутентичности кода
Интеграция ПО	Создание и тестирование		Проверка функциональной, физической конфигурации Сертификация
Интеграция системы		Тестирование и развитие	Проверка функциональной, физической конфигурации Сертификация
Приемка и поставка системы	Отчуждение		
Эксплуатация и сопровождение			
Завершение проекта			

Рис. 3. Вариант упрощенного жизненного цикла программной системы

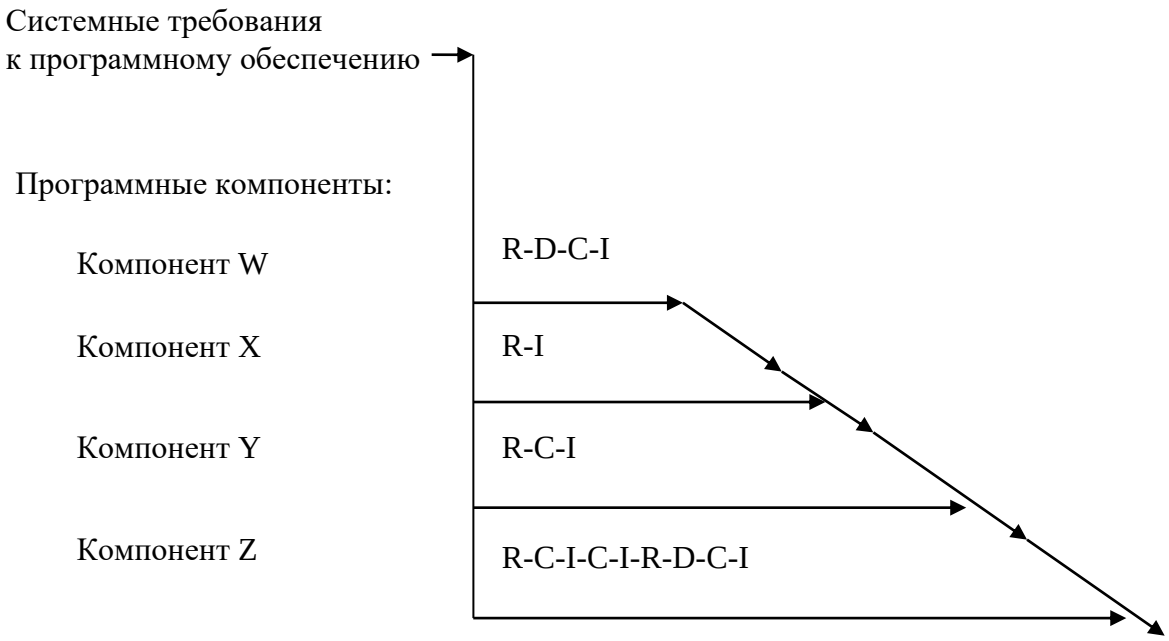


Рис. 4. Последовательность этапов разработки компонент программного обеспечения

Обозначения:

R – постановка задачи;

D – проектирование;

C – кодирование;

I – интеграция.

Примечание: этапы планирования и интегрирования не обозначены

Этапы жизненного цикла по срокам выполнения:

– разработка (development): проектирование (design) и реализация (implementation) – 0,5 – 2,0 года;

– сопровождение (maintenance) – 1 – 10 лет.

7. Этапы разработки по срокам выполнения

Этап	Результат	Трудоемкость
1. Проектирование		40%
1.1. Анализ требований	Внешняя спецификация	10%
1.2. Общее проектирование	Проектные спецификации (Specifications)	10%
1.3. Детальное проектирование		20%
2. Реализация		60%
2.1. Кодирование	Исходные тексты программ	10%
2.2. Автономное тестирование	Журналы ошибок (Bug books)	20%
2.3. Комплексное тестирование		30%

7.2. МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА ПО

Стандарт IS/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО. Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

Модель ЖЦ определяет последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении ЖЦ. Модель ЖЦ зависит от специфики ПС и специфики условий, в которых система создается и функционирует. К настоящему времени наибольшее распространение получили следующие три основные модели ЖЦ:

- каскадная модель (1970 – 1980);
- поэтапная модель с промежуточным контролем (1980 – 1985);
- спиральная модель (с 1986 по н.в.).

Все рассматриваемые модели ЖЦ ПО используются и в настоящее время для соответствующих классов создаваемых систем.

Каскадная модель

В изначально существовавших однородных ПС приложения представляли собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 1.6).

Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Особенности этой канонической модели:

- дискретный (во времени) технологический процесс с остановками по завершению каждого этапа и возможностью контроля и возврата при неудовлетворительных результатах (итерации разработки);
- деление на уровни абстракции позволяет формировать документацию различной степени детализации и формулировать критерии проверки правильности принимаемых решений до реализации проекта, что способствует предупреждению ошибок проектирования;
- дисциплина требует не переходить к очередному этапу до полного завершения предыдущего.

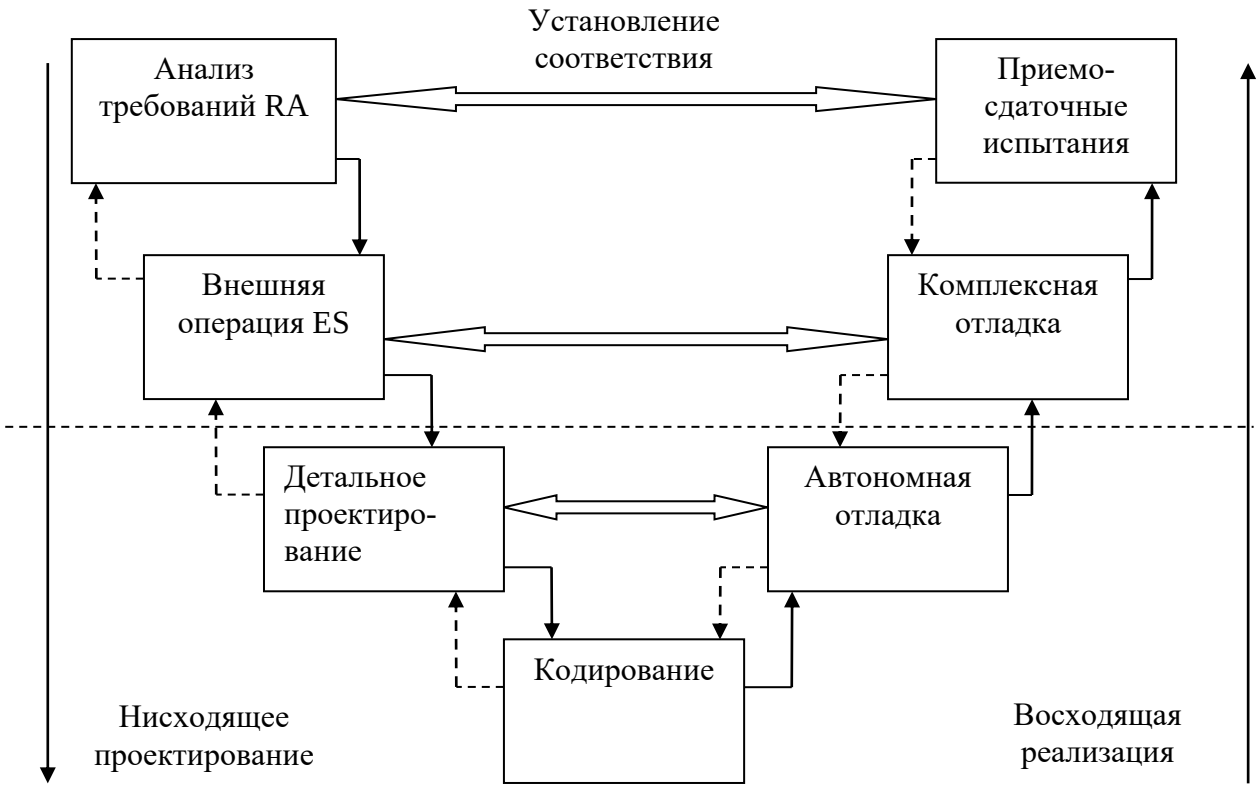


Рис. 5. Каскадная схема разработки ПО

Очевидно:

– чем раньше допущена ошибка, тем позже она может быть обнаружена;

– чем позже обнаружена ошибка, тем дороже ее исправление – больше этапов возврата;

– поэтому цель хорошей технологии – возможно более раннее выявление и предупреждение ошибок, что достигается тщательным проектированием.

Недостатки каскадной модели:

– навязывается стратегия проектирования сверху вниз (top-down), что подходит только к хорошо специфицированным небольшим проектам, в основном, прикладным. Сложные системы либо создаются по частям (как город), либо растут снизу вверх (bottom-up), как природные объекты;

– большая длительность полного цикла;

– заказчики/пользователи могут увидеть первые результаты очень поздно;

– только по завершению всего проекта, – и пожелать радикально изменить требования;

– требования конкурентного рынка заставляют сокращать цикл;

– модель, созданная в начале 1970 гг., абсолютизирует значение стадии проектирования «за столом»: не учитывает ускорения цикла кодирование-отладка в эпоху персональных компьютеров.

Каскадный подход хорошо зарекомендовал себя при построении ПС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их технически как можно лучше. В эту категорию попадают сложные расчетные системы, системы реального времени и др.

Поэтапная модель

В то же время этот подход обладает радом недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему, постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений.

Межэтапные корректировки обеспечивают большую надежность по сравнению с каскадной моделью, хотя и увеличивают весь период разработки.

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ПС «заморожены» в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО пользователи получают систему, не удовлетворяющую их по требностям. Модели автоматизируемого объекта могут устареть одновременно с их утверждением.

Спиральная модель

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ, в которой делается упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации. Каждый виток спирали – цикл разработки очередной версии по каскадной схеме.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Функциональность наращивается постепенно: первые результаты получаются быстро, сложность увеличивается постепенно; удобно для разработки снизу вверх и меньше моделирования на этапе проектирования.

Варианты спиральной модели:

– быстрое прототипирование;

– инкрементальная (наращиваемая) разработка; модели жизненного цикла ASDH.

Схема модели представлена на рис. 6. Прототип – макет, упрощенная версия будущего продукта для подтверждения его осуществимости (feasibility). Упрощения: неполная функциональность и(или) эффективность (например, прототип разрабатывается на неэффективном языке – Smalltalk, Prolog и пр.). Сейчас прототип – почти обязательная принадлежность внешней спецификации небольшого проекта, особенно в области мультимедиа.

Инкрементальная* (*наращиваемая*) *разработка

Инкремент – полностью работоспособная версия, поступающая в эксплуатацию. Удобно для больших систем. Например, ПО бортовой системы управления i Shuttle: 7 инкрементов за 10 лет (IBM). По этому принципу работают программисты Microsoft, поэтому другое ее название – Microsoft Solutions Framework MSF).

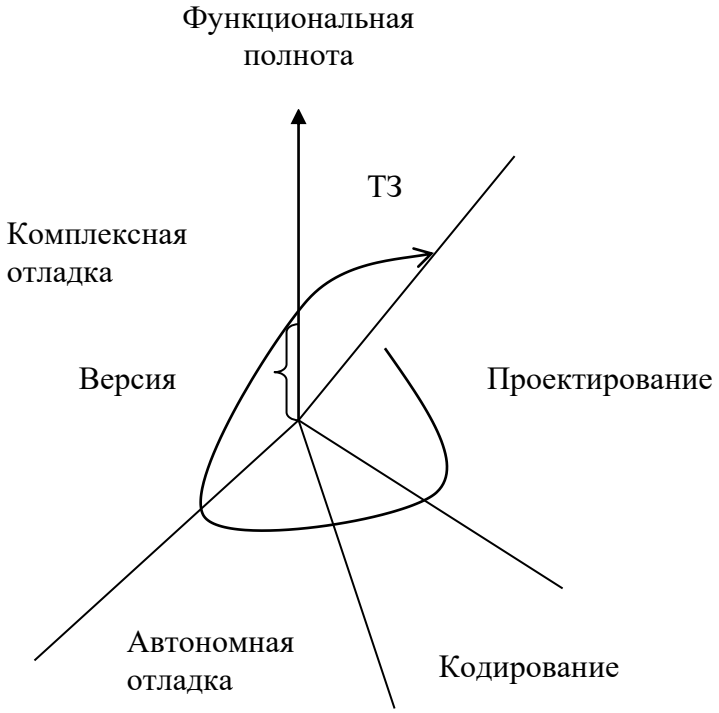


Рис. 6. Классическая спиральная модель ЖЦ

Особенности спиральной модели:

– общая структура действий на каждой итерации – планирование, определение задач, ограничений и вариантов решений, оценка предложенных решений и рисков, выполнение основных работ итерации и оценка их результатов;

– решение о начале новой итерации принимается на основе результатов предыдущей;

– досрочное прекращение проекта в случае обнаружения его нецелесообразности.

Достоинства итерационных моделей:

– полный учет требований заказчика, большее его участие в проекте;

– равномерная нагрузка на группу;

– раннее обнаружение проблем и их разрешение по мере возникновения, уменьшение рисков на каждой итерации.

Недостатки итерационных моделей: сложность планирования; плохая документированность создаваемого ПО.

Проблемой современной программной инженерии являются «тяжелые» процессы. Характеристики «тяжелого» процесса:

1. Необходимость документировать каждое действие разработчиков.

2. Множество рабочих продуктов (в первую очередь – документов), создаваемых в бюрократической атмосфере.

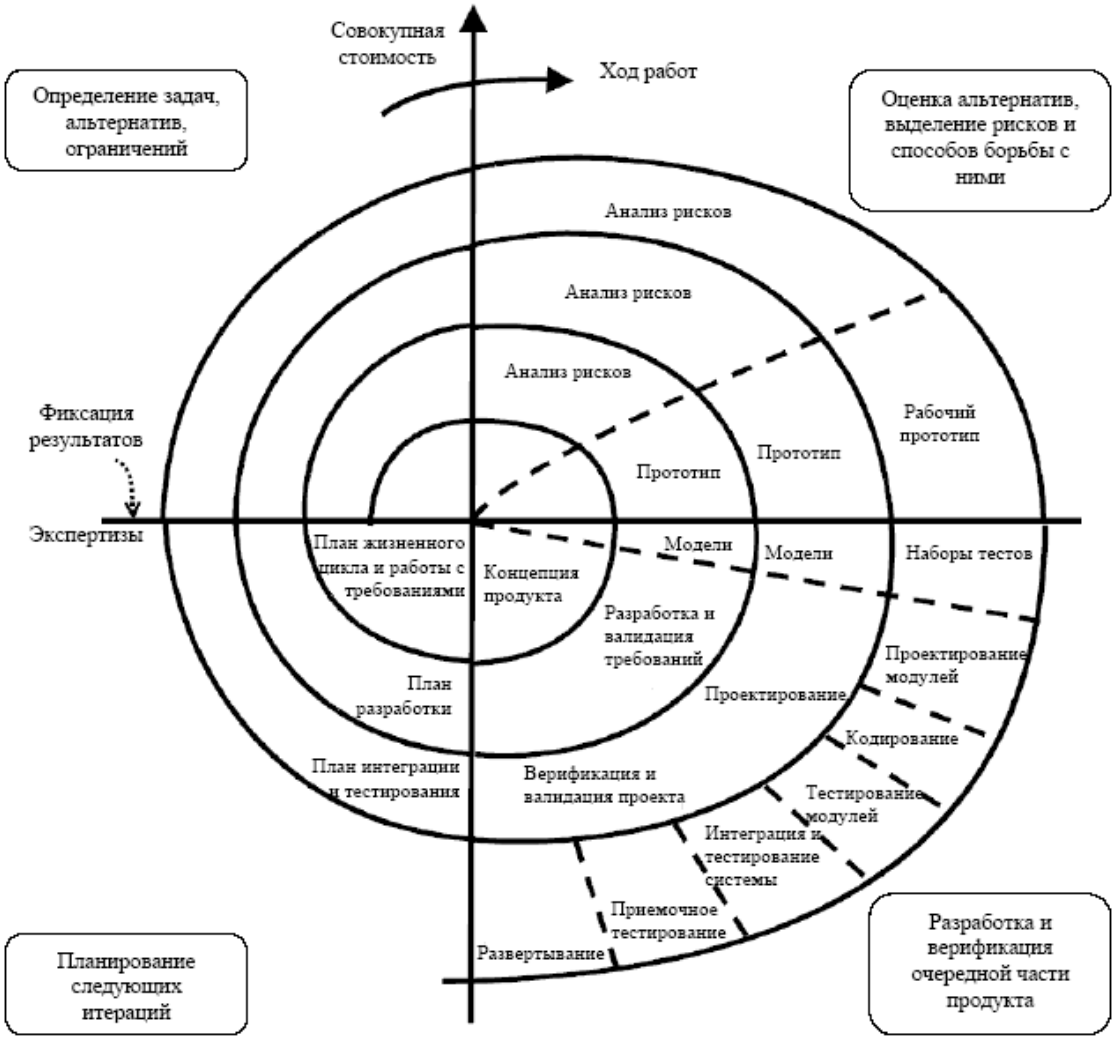


Рис. 7. Схема спиральной модели ЖЦ

3. Отсутствие гибкости.

4. Детерминированность (долгосрочное детальное планирование и предсказуемость всех видов деятельности, распределение человеческих ресурсов на длительный срок, охватывающий большую часть проекта.

Противоположностью «тяжелого» процесса является «легковесный» процесс – основа быстрой разработки ПО (Agile Software Development). Быстрая разработка ориентируется на эффективную коммуникацию между разработчиками, высокую квалификацию разработчиков и другие факторы, позволяющие сократить расходы на «бюрократию».

8. СРЕДСТВА CASE-ТЕХНОЛОГИЙ

Программное средство, предназначенное для полной или частичной поддержки жизненного цикла других программных средств, обеспечивающее автоматизированную поддержку деятельности, выполняемой в рамках технологических операций, называют программным инструментом или CASE-средством (Computer-Aided Software Engineering Tool).

В настоящее время не существует общепринятого определения CASE. Содержание этого понятия обычно определяется перечнем задач, решаемых с помощью CASE, а также совокупностью применяемых методов и средств.

CASE-технология представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем программного обеспечения (ПО), поддержанную комплексом взаимосвязанных средств автоматизации.

CASE – это инструментарий для системных аналитиков, разработчиков и программистов, позволяющий автоматизировать процесс проектирования и разработки ПО.

Инструментальная среда – логически связанная совокупность CASE-средств. Инструментальные среды можно классифицировать по следующим признакам: ориентированность на конкретный язык программирования, специализированность, комплексность, ориентированность на конкретную технологию программирования, ориентированность на коллективную разработку, интегрированность.

Специализированность инструментальной среды показывает: ориентирована ли среда на какую-либо предметную область или нет. Комплексность инструментальной среды показывает: поддерживает ли она все процессы жизненного цикла или нет.

Интегрированность инструментальной среды показывает: является ли она интегрированной или нет. Инструментальная среда считается интегрированной, если взаимодействие пользователя с инструментами подчиняется единообразным правилам, а сами инструменты действуют по заранее заданной информационной схеме, связаны по управлению или имеют общие части. В соответствии с этим различают три вида интегрированности: интегрированность по пользовательскому интерфейсу, интегрированность по данным, интегрированность по действиям.

Интегрированность по пользовательскому интерфейсу означает, что все инструменты объединены единым пользовательским интерфейсом. Интегрированность по данным означает, что инструменты действуют в соответствии с фиксированной информационной схемой системы, определяющей зависимость друг от друга различных используемых в системе фрагментов данных. В этом случае может быть обеспечен контроль полноты и актуальности программных документов и порядка их разработки. Интегрированность по действиям означает, что, во-первых, в системе имеются общие части всех инструментов и, во-вторых, одни инструменты при выполнении своих функций могут обращаться к другим инструментам.

Инструментальную среду, интегрированную хотя бы по данным или по действиям, будем называть инструментальной системой.

При этом интегрированность по данным предполагает наличие в системе специализированной базы данных, называемой репозиторием. Под репозиторием понимают центральное компьютерное хранилище информации, связанное с проектом (разработкой) ПС в течение всего его жизненного цикла.

В настоящее время выделяют три основных класса инструментальных сред:

– инструментальные среды программирования;
– рабочие места case-технологии;
– инструментальные системы технологии программирования.

Большинство CASE-средств основано на парадигме методология/ метод/нотация/средство.

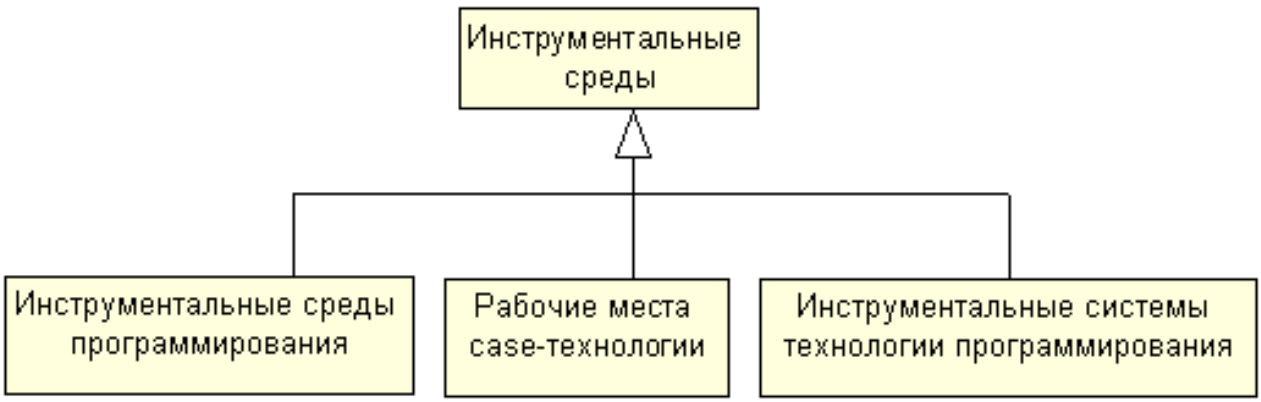


Рис. 8. Основные классы инструментальных сред

Методология определяет руководящие указания для оценки и выбора проекта разрабатываемого ПО, шаги работы и их последовательность, а также правила распределения и назначения методов.

Метод – это систематическая процедура или техника генерации описаний компонентов ПО (например, проектирование потоков и структур данных).

Нотации предназначены для описания структуры системы, элементов данных, этапов обработки и включают графы, диаграммы, таблицы, блок-схемы, формальные и естественные языки.

Средства – инструментарий для поддержки и усиления методов. Эти инструменты поддерживают работу пользователей при создании и редактировании графического проекта в интерактивном режиме, они способствуют организации проекта в виде иерархии уровней абстракции, выполняют проверки соответствия компонентов.

Классификация CASE-методологий

В настоящее время существует достаточно большое количество CASE-методологий, предназначенных для описания различных аспектов системы.

Для описания СУ используются методологии функционального моделирования (являющиеся частью методологий структурного системного анализа (ССА)), к которым относятся диаграммы потоков данных (Data Flow Diagrams DFD) и подгруппа методологий IDEF (Icam DEFinition) или SADT – IDEFO IDEF3. DFD ориентированы на разработчиков ПО, IDEFO – на процесс общения разработчика и заказчика.

Для описания БД используются методологии описания данных: диаграммы сущность-связь (Entity-Relation Diagrams – RD) и IDEFlx, принципиальное отличие которых в формах представления.

Кроме того, в настоящее время успешно развивается объектно-ориентированное представление ИС и соответствующие средства описания – универсальный язык моделирования (Unified Modeling Language – UML).

Вариант классификации CASE-методологий представлен на рис. 9.

IDEF представляет собой группу методологий, состоящую из 14 слабо связанных между собой составляющих и предназначенную для описания различных аспектов системы. Наименование методологий представлено в табл. 7.

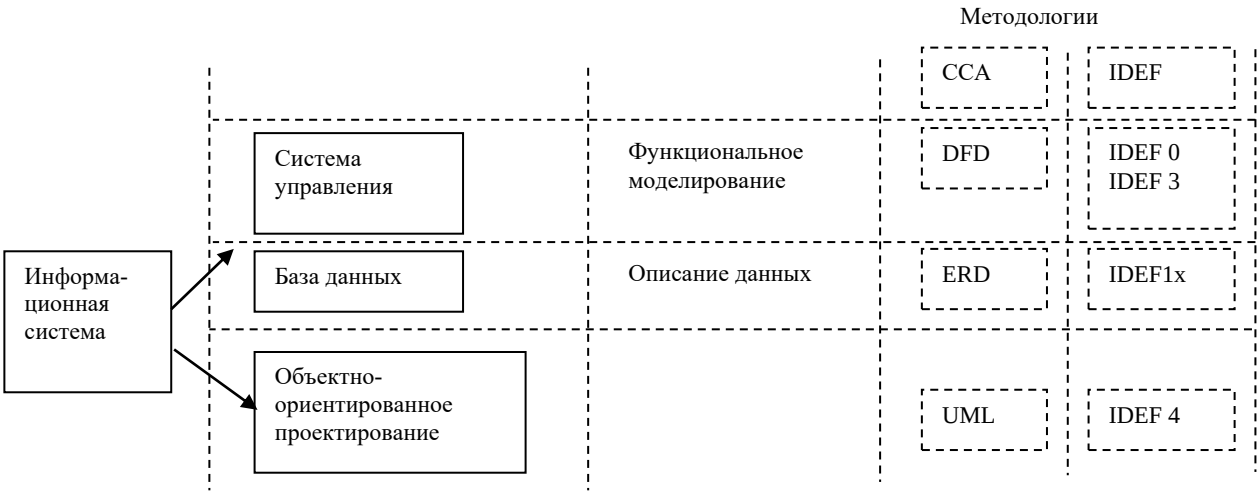


Рис. 9. Классификации CASE-методологий по соответствующим группам

7. Методологии IDEF

	Suite of IDEF Methods	Набор методов IDEF
IDEF O	Function Modeling	Функциональное моделирование
IDEF 1	Information Modeling	Информационное моделирование
IDEF 2	Simulation Modeling	Имитационное моделирование
IDEF lx	Data Modeling	Мооделирование данных
IDEF 3	Process Description Capture	Описание процесса проектирования
IDEF 4	Object-oriented Design	Объектно-ориентированное проектирование
IDEF 5	Ontology Description Capture	Антология описания проекта
IDEF 6	Design Rationale Capture	Разработка логического обоснования проекта
IDEF 7	Information System Audit Method	Метод проверки информационных систем
IDEF 8	User Interface Modeling	Моделирование интерфейса пользователя
IDEF 9	Scenario-driven Info Sys Design Spec	Управляемая сценарием информация системные спецификации проекта
IDEF 10	Implementation Architecture Modeling	Моделирование архитектуры выполнения
IDEF 11	Information Artifact Modeling	Моделирование экспоната информации
IDEF 12	Organization Modeling	Моделирование организации
IDEF 13	Three Schema Mapping Design	Тройная схема отображения проекта
IDEF 14	Network Design	Проектирование сетей

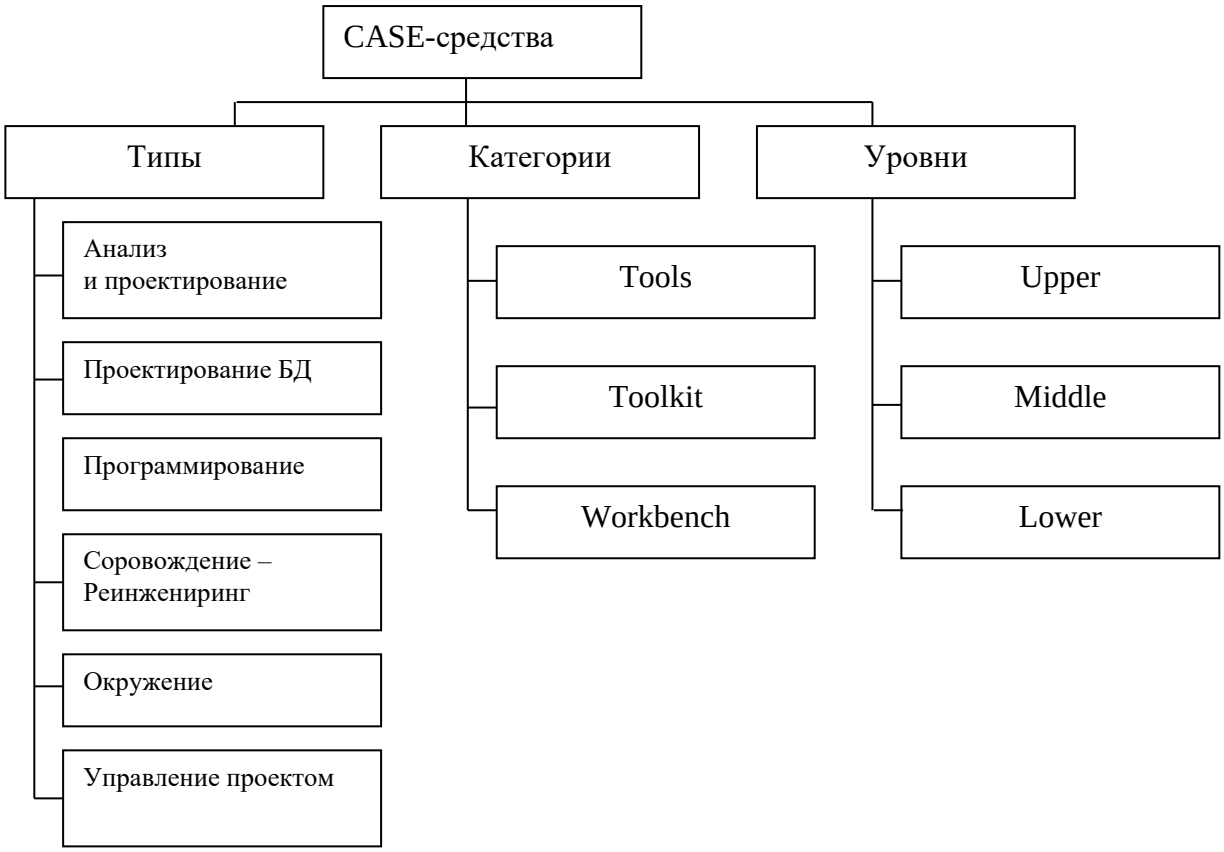


Рис. 10. Все CASE-средства делятся по типам, категориям и уровням

9. ОБОБЩЕННОЕ СОДЕРЖАНИЕ ЭТАПОВ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

1. Анализ и спецификации требований к ИС – анализ функциональных и нефункциональных требований, требований к внешним интерфейсам ИС, к аппаратному обеспечению, по обеспечению информационной безопасности, определение архитектуры аппаратного обеспечения, проектирование ПО, идентификация объектов и их иерархия;

1.1. Сбор и анализ функциональных требований:

1.1.1. Требования, согласованные всеми заинтересованными представителями бизнес-подразделений:

- характеристика ИС;
- основные цели создания и назначение;
- планируемое развитие;
- требования к ИС в целом;
- требования к структуре данных ИС;
- требования к функциональности ИС;
- требования к интерфейсу пользователя и отчетам;
- требования к средствам администрирования;
- требования к средствам разработки;
- требования к взаимодействию с другими программными продуктами;
- требования к документированию;
- требования к программному и аппаратному обеспечению.
- описание спецификации ИС, в которых применяется данное приложение;
- документирование форматов входных и выходных данных;

– методы реализации контролей подготовки, ввода данных и обработки ошибок ввода данных, позволяющих реализовать обнаружение, индикацию и корректировку ошибок;

– методы реализации контролей проверки данных транзакций, введенных для обработки (вручную или системой) на точность, полноту и достоверность;

– методы обнаружения и исключения ошибочных транзакций по ходу их обработки;

– средства, поддерживающие неотказуемость транзакций.

1.1.2. Формализованные требования, согласованные всеми заинтересованными представителями бизнес-подразделений:

– описание детальных спецификаций ПО в части функциональности системы;

– требования к разработке и документированию интерфейсов с внутренними системами;

– требования к разработке и документированию алгоритмов обработки данных;

– требования обеспечения безопасности, целостности и доступности (в том числе при чрезвычайных ситуациях).

1.2. Нефункциональные требования должны быть использованы для определения требований и ограничений ИС. Требования должны быть использованы в качестве основы для ранней системы измерения и оценки жизнеспособности разрабатываемой ИС.

Нефункциональные требования включают:

– ограничения среды и реализации;

– производительность;

– зависимость от платформы;

– расширяемость;

– надежность.

При определении нефункциональных требований должны применяться следующие категории атрибутов качества:

- управляемость;
- рентабельность;
- эффективность;
- возможность взаимодействия с другими приложениями и службами;
- доступность и надежность;
- мощность и производительность;
- возможность внесения исправлений;
- возможность инсталлирования;
- контролируемость;
- поддерживаемость;
- удобство использования;
- безопасность.

Нефункциональные требования должны использоваться для предписания качественных атрибутов разрабатываемого приложения. Данные качественные атрибуты должны применяться для составления планов тестирования приложений в соответствии с нефункциональными требованиями.

1.3. Требования к внешним интерфейсам ИС, в зависимости от необходимости, должны включать:

1. Приоритет, который ИС должна присвоить интерфейсу.
2. Требования к типу интерфейса (функционирование в режиме реального времени, передача данных, хранение и извлечение данных), которые необходимо обеспечить.
3. Требуемые характеристики индивидуальных элементов данных, которые должна предоставлять ИС для обеспечения хранения, передачи, доступа, приема данных:

- имя/идентификаторы;
- тип данных (цифробуквенный, целый);

– размер и формат (длина и пунктуация ряда символов);

– единицы измерения (объема, стоимости, времени);

– интервал или перечисление возможных значений (например, 0 – 99);

– аккуратность (насколько верно) и точность (число значащих разрядов);

– приоритет, хронометраж, частота, объем, порядок и другие ограничения для актуализации элемента данных и применяемые правила бизнеса;

– ограничения по безопасности;

– источники (объекты, которые создают/посылают данные) и получатели (объекты, которые используют/принимают данные).

4. Требуемые характеристики совокупностей элементов данных, которые ИС должна предоставлять, хранить, передавать, принимать и обеспечивать к ним доступ:

– имя/идентификаторы;

– элементы данных, которые входят в их структуру (номер, порядок, группировка);

– среды (например, диски) и структуры данных/элементов в среде;

– аудиовизуальные характеристики сообщений и других выходов;

– отношения между группами, характеристики сортировки/доступа;

– приоритет, хронометраж, частота, объем, порядок и другие ограничения;

– время реагирования на запрос;

– ограничения безопасности;

– источники и получатели.

5. Требуемые характеристики для методов коммуникаций, которые ИС должна использовать для интерфейса:

– уникальные проектные идентификаторы;

– коммуникации, связи/полосы пропускания/частоты/среды и их характеристики;

– формат сообщений;

– контроль потоков;

– скорости передачи, периодичность, интервал между передачами;

– маршрутизация, адресация и пространства имен;

– услуги передачи, включая приоритеты и оценки;

– соображения безопасности/конфиденциальности, такие как шифрование, аутентификация пользователей, проведение аудита.

6. Требования к характеристикам, которыми должны обладать протоколы ИС для взаимодействия:

– уникальный идентификатор проекта;

– приоритет/уровень протокола;

– разбиение на пакеты, включая фрагментацию, повторную сборку, маршрутизацию и адресацию;

– проверки легальности, контроль над ошибками и процедуры восстановления;

– синхронизация, включая установку соединения, поддержание, прекращение;

– состояние, идентификация и любые возможности отчетов;

– другие требуемые характеристики, физическая совместимость взаимодействующих объектов информационных ресурсов.

1.4. Требования по обеспечению информационной безопасности

– идентификация защищаемых информационных ресурсов;

– аутентификация пользователей ИС;

– конфиденциальность информации, циркулирующей в системе;

– аутентифицированный обмен данными;

– целостность данных при формировании, передаче, использовании, обработке и хранении информации;

– авторизированная доступность всех ресурсов системы в условиях нормальной эксплуатации;

– разграничение доступа пользователей к ресурсам ИС;

– администрирование (обозначение прав доступа к ресурсам ИС, обработка информации из регистрационных журналов, установка и снятие системы защиты);

– регистрация действий по входу и выходу пользователей, а также нарушений прав доступа к ресурсам ИС;

– контроль целостности и работоспособности системы обеспечения информационной безопасности;

– безопасность и бесперебойное функционирование системы в чрезвычайных ситуациях.

1.5. На этапе определения архитектуры ИС должны быть разработаны детализированные модели того, что было подготовлено на этапе разработки бизнес-требований.

Разрабатываемые модели включают архитектурные модели, в которых необходимо определить способ преобразования различных функциональных компонентов в физические компоненты.

На данном этапе должны быть разработаны планы тестирования для различных уровней тестирования:

– тестирование блочное;

– тестирование модуля;

– тестирование системы;

– тестирование интерфейсов между системами;

– тестирование файлов загрузки;

– нагрузочное тестирование;

– тестирование безопасности;

– тестирование резервного копирования.

Архитектура ИС включает описание:

– физических сущностей, из которых состоит система;

– информационных потоков между физическими сущностями в системе;

– спецификаций каналов передачи информации в системе;

– выбора платформы и операционной системы;

– выбора архитектуры «файл-сервер» или «клиент-сервер»;

– выбора 3-уровневой архитектуры со следующими слоями: сервер, ПО промежуточного слоя, клиентское ПО;

– выбора базы данных централизованной или распределенной.

Подтверждение требований по мощности, производительности и доступности ИС относятся к основным действиям, производимым на данном этапе.

Результаты этапа включают:

– определение аппаратных и программных требований и ограничений;

– определение функциональных, нефункциональных и требований по безопасности ИС;

– создание программной архитектурной иерархии сегментов и связанных функций;

– создание аппаратной архитектуры;

– определение структурных единиц программного и аппаратного обеспечения.

Структурные единицы включают:

1. Управление ПО – предполагает решение широкого спектра задач:

– отслеживание сбоев в управляемых компьютерах, автоматическое устранение их причин, исправление их последствий и действия по их предотвращению;

– управление производительностью компьютеров и приложений;

– автоматическое конфигурирование компьютеров и сетевых устройств;

– конфигурирование и обновление ПО.

2. Проектирование ИС – определяется как итерационный процесс получения логической модели ИС вместе со строго сформулированными целями, поставленными перед ней, а также написания спецификаций физической системы, удовлетворяющей этим требованиям.

3. Разработка ПО – достаточно подробное уточнение требований заказчика и проектного решения и преобразование их в один или несколько жизнеспособных ПО.

4. Тестирование ПО – один из важнейших этапов проверки качества разработанного ПО.

5. Среда разработки ПО – система программных средств, используемая программистами для разработки ПО. Среда разработки включает:

– текстовый редактор;

– компилятор и(или) интерпретатор;

– средства автоматизации сборки и отладчик;

– систему управления версиями;

– инструменты для упрощения конструирования графического интерфейса пользователя;

– браузер классов, инспектор объектов и диаграмму иерархии классов.

6. Системный уровень поддержки тестирования включает испытание разрабатываемого ПО, испытание системного уровня ПО, поддержку сборки, тестирования и процедуры сдачи ПО в эксплуатацию.

7. Сборка, тестирование и процедура сдачи ПО в эксплуатацию – готовое ПО передается заказчику, производятся приемо-сдаточные испытания, осуществляется обучение пользователей и опытная эксплуатация, после чего ПО ставится на сопровождение и начинается производственная эксплуатация ПО.

8. Система обеспечения качества направлена на обеспечение необходимого уровня объективности и достоверности результатов разработки и внедрения ПО, а также на обеспечение надлежащего качества производимой продукции и(или) предоставляемых услуг.

9. Независимая проверка и подтверждение адекватности ИС.

Эти категории структурного разбиения работ включают в себя процессы жизненного цикла разработки ИС:

– оценка размера программной части ИС – определение размера программной части ИС и составление документации по оценке размера программной части;

– разделение и группировка программных функциональных требований и требований к ИС в категории программного наследования.

Категории программного наследования включают:

– новый проект и новый код;

– аналогичный проект и новый код;

– аналогичный проект и многократно используемый код;

– аналогичный проект и расширенный код, используемый многократно.

Нефункциональные требования и требования к интерфейсам используются для составления планов тестирования ПО и приложений.

Определение характеристик качества ИС – определение количественных, качественных, внутренних и внешних характеристик качества и оценка качества ИС.

10. ОБЪЕКТНАЯ МОДЕЛЬ – ОСНОВА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПОДХОДА К АНАЛИЗУ И ПРОЕКТИРОВАНИЮ

Основным исходным понятием объектно-ориентированного моделирования (ООМ) является объект [36]. В самом общем смысле объект является описанием множества однотипных сущностей, обладающих идентичностью (имя), состоянием (свойства), поведением (с ним можно выполнять определенные операции, или он сам может воздействовать на другие объекты) и семантикой (для объекта могут быть заданы правила, которым должны удовлетворять состояния и операции) [36].

Каждый объект может рассматриваться как экземпляр некоторого класса. Объекты, которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением, по определению, не могут быть отнесены к одному классу.

Объектно-ориентированный подход к описанию сложных систем состоит в том, что структурные элементы систем и их поведенческие свойства описываются в рамках единого понятия.

Объекты могут использоваться для представления как структурированных состояний, так и действий или событий [36].

Абстракции объектов в ООМ могут формулироваться в двух формах: абстракции состояний системы (абстрактные объекты) и абстракции преобразований системы (абстрактные операции) [36].

Абстрактные объекты не находятся в статическом состоянии, с течением времени они изменяют свое состояние. Состояния объекта связывают с его экземпляром. Экземпляры одного объекта различаются значениями свойств, имена которых одинаковы [36].

В объектно-ориентированном подходе не делается различий между свойствами и методами (операциями) объекта. Состояние, в котором может находиться объект, изменяется с помощью методов.

Метод – это определенная внутри объекта процедура или функция, для которой значения свойств объекта доступны без явной передачи их в качестве параметров.

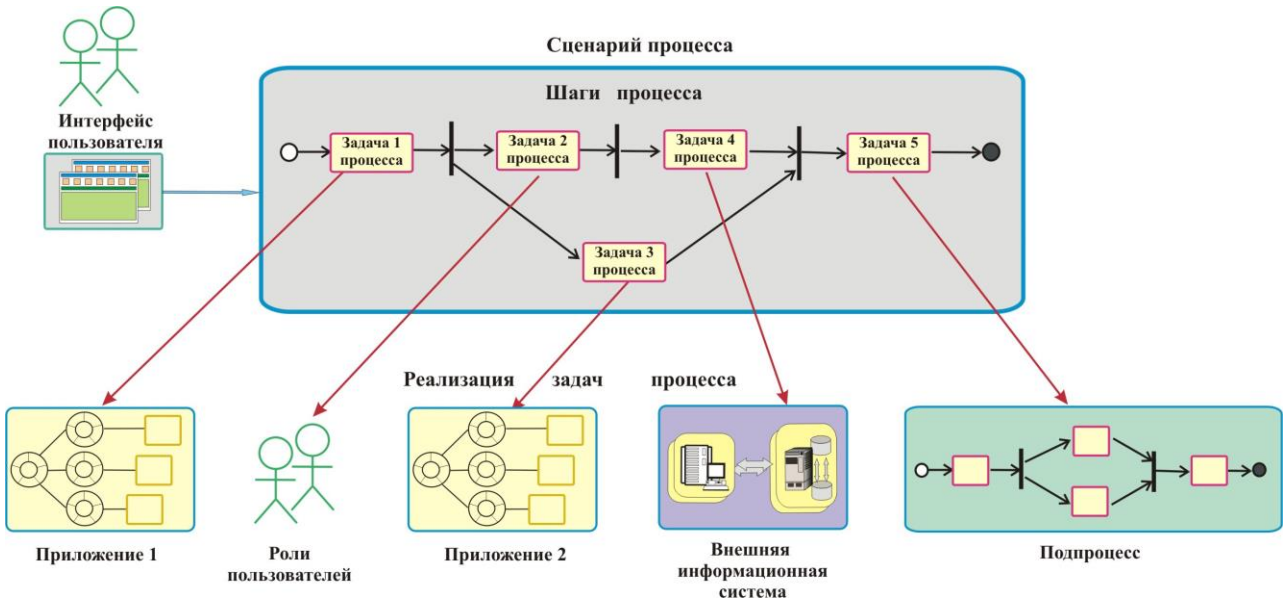


Рис. 11. Пример сценария функционального процесса

Функциональный процесс – это логически завершенный набор этапов работы системы, поддерживающий ее деятельность, направленную на достижение поставленных *целей*.

Модель сценария функционального процесса – это особая форма конечного автомата, которая отображает процесс выполнения работ или действий в системе (рис. 11) [36].

В сценарии процесса моделируются состояния производимых действий или решаемых задач. Состояния задач мгновенны и не допускают переходов во время своего выполнения. При возникновении события задача переходит в активное состояние, в котором она находится до завершения процесса действия. По завершении данной задачи система переходит к следующей задаче. Завершение сценария процесса осуществляется в тот момент, когда закончилась вся предыдущая деятельность системы [36].

Задача определяет работу, которую необходимо выполнить, и она может быть специфицирована различными способами, включая текстовое описание в виде резолюции на документе. Задачи могут быть выполнены вручную или с помощью программных систем. Для определения сценария процесса необходимо описать аспекты составляющих их задач (и обрабатывающих устройств, которые их выполняют).

Процесс описания систем в ООМ можно рассматривать как последовательный переход от разработки наиболее общих *моделей* и представлений концептуального уровня к более частным и детальным представлениям физи-

ческого уровня [36]. При этом на каждом этапе ООМ данные *модели* последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

Общая схема взаимосвязей представлений языка UML представлена на рис. 12 [36].

Наиболее общими представлениями сложной системы считаются статическое и динамическое представления, которые, в свою очередь, могут подразделяться на другие более частные представления.

Моделирование сложной системы средствами UML сводится к ее описанию в различных проекциях или представлениях. Каждая проекция описывает определенный аспект разрабатываемой системы, а все вместе они определяют систему с должной степенью полноты, правильности и адекватности [36].

Интегрированная модель сложной системы в нотации UML представлена на рис. 13 [36].

Рис. 12. Общая схема взаимосвязей моделей и представлений сложной системы в ООМ

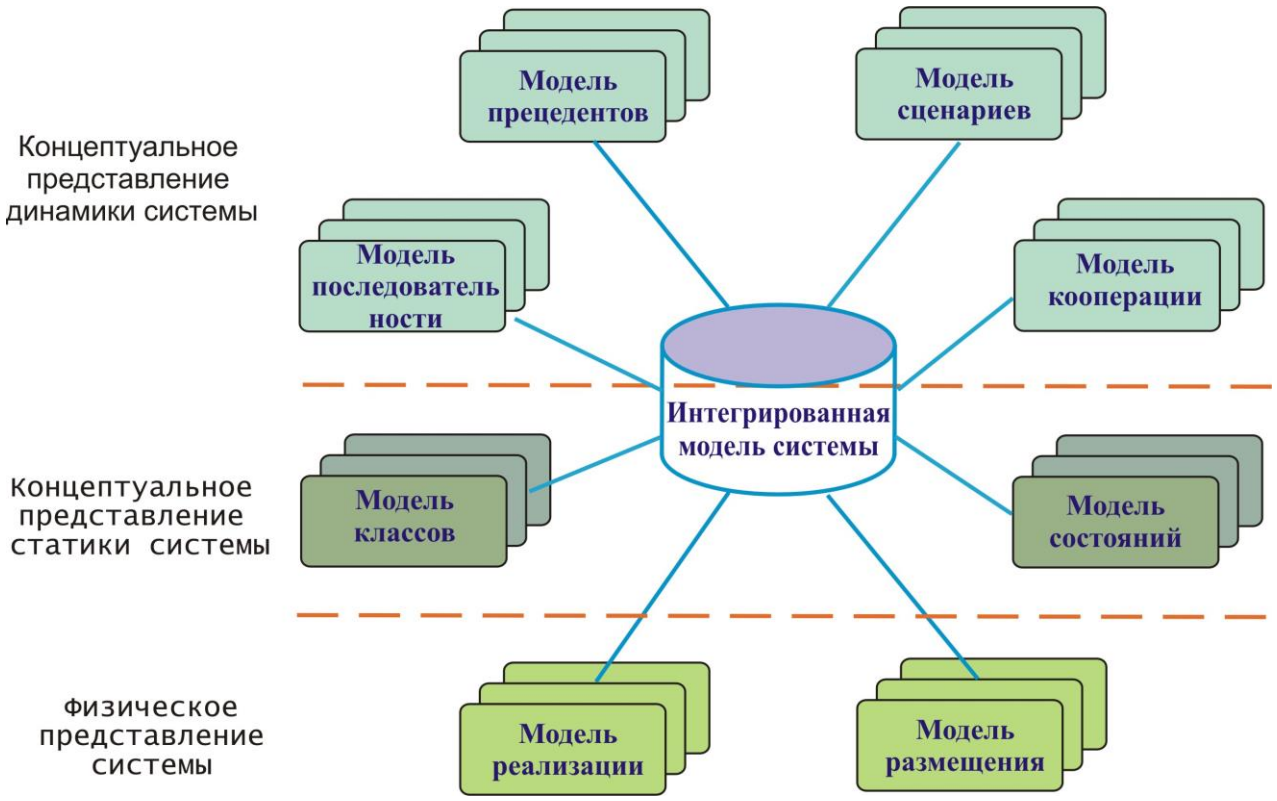


Рис. 13. Совокупность моделей, описывающих различные представления сложной системы на языке UML

На этапе создания концептуального представления поведения сложной системы используются динамические модели, включающие модели прецедентов (вариантов использования), сценариев функциональных процессов и модели взаимодействия объектов [36].

Модель прецедентов (вариантов использования, use case diagrams) – это обобщенная модель функционирования системы в окружающей среде. Прецедент – это реакция системы на внешнее воздействие. Модель прецедентов представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных моделей. На моделях прецедентов представляются варианты использования, действующие лица (актеры), а также отношения между ними. Они особенно важны при организации и моделировании поведения системы.

Модели сценариев функциональных процессов (activity diagrams) – это модели поведения системы в рамках прецедента. На моделях сценариев процессов представляются переходы потока управления от одной деятельности к другой внутри системы. Модели сценариев процессов относятся к динамическому виду системы; они наиболее важны при моделировании ее функционирования и отражают поток управления между объектами.

Модель взаимодействия объектов (interaction diagrams) – модель процесса обмена сообщениями между объектами системы, представляется в виде диаграмм последовательностей (sequence diagrams) или кооперативных диаграмм (collaboration diagrams). На моделях взаимодействия представляются связи между объектами; и, в частности, показываются сообщения, которыми объекты могут обмениваться. Модели взаимодействия относятся к динамическому виду системы. При этом диаграммы последовательности отражают временную упорядоченность сообщений, а диаграммы кооперации – структурную организацию обменивающихся сообщениями объектов. Эти диаграммы являются изоморфными, то есть могут быть преобразованы друг в друга.

На этапе создания статического концептуального описания сложной системы строится логическое представление, использующее модели классов и модели состояний объектов.

Модель классов (class diagrams) – логическая модель базовой структуры системы, отражает статическую структуру системы и связи между ее элементами. Модели классов соответствуют статическому виду системы. На модели классов показывают классы, интерфейсы, объекты и кооперации, а также их отношения.

Для уточнения моделей классов часто используются *модели объектов* (object diagrams). На моделях объектов представляются объекты и отношения между ними. Они являются статическими «фотографиями» экземпляров сущностей, показанных на моделях классов. Модели объектов, как и модели классов, относятся к статическому виду системы, но с расчетом на настоящую или макетную реализацию.

Модель переходов состояний (statechart diagrams) – модель динамического поведения системы и ее компонентов при переходе из одного состояния в другое. На моделях состояний представляется автомат, включающий в себя состояния, переходы, события и виды действий. Модели состояний относятся к динамическому виду системы; особенно они важны при моделировании поведения интерфейса, класса или кооперации.

Отношения между различными видами моделей UML показаны на рис. 14.

На этапе создания физического представления сложной системы используются модели реализации компонентов и модели размещения.

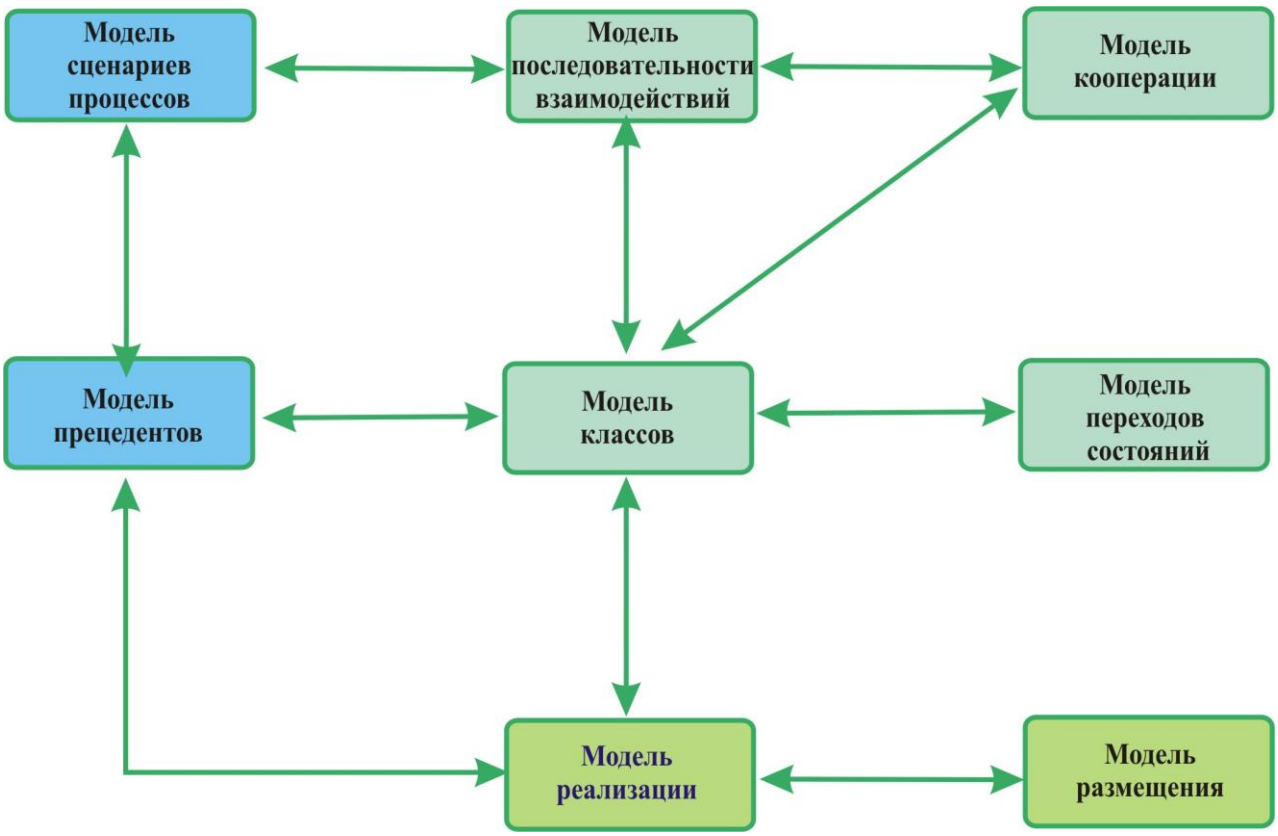


Рис. 14. Взаимосвязи между моделями сложной системы

Модель реализации (компонентов) (Component Diagrams) – модель иерархии подсистем, отражает физическое размещение данных, приложений и интерфейсов системы. На модели реализации представляется организация совокупности компонентов системы и существующие между ними зависимости. Модели реализации относятся к статическому виду системы с точки зрения ее физического представления. Они могут быть соотнесены с моделями классов, так как компонент обычно отображается на один или несколько классов, интерфейсов или взаимодействий.

Модели размещения (Deployment Diagrams) – модель физической архитектуры системы, отображает технологическую конфигурацию системы. На модели размещения обычно представляется конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Модели размещения относятся к динамическому виду архитектуры системы с точки зрения ее развертывания. Они связаны с моделями компонентов, поскольку в узле системы обычно размещаются один или несколько компонентов.

11. ПОСТРОЕНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

Построение концептуальной модели можно формализовать пятеркой:

$$Sвп = <P, O, H, \Re, \Im>,$$

где $P = \{p_i\}$ – множество процессов обработки информации; $O = \{o_i\}$ – множество информационных объектов (данных) ($P \cap O = \varnothing$); $H = \{Hp, Ho\}$ – отношения иерархии: $Hp \subseteq P \times B(P)$ – отношения иерархии процессов; $Ho \subseteq O \times B(O)$ – отношения иерархии объектов; $\Re = \{In, Out\}$ – отношения взаимодействия: $In \subseteq B(O) \times P$ – отношения «входные объекты процесса – процесс»; $Out \subseteq P \times B(O)$ – отношения «процесс – выходные объекты процесса»; $\Im \subseteq P \times B(P)$ – отношения следования процессов.

Отношения иерархии Hp (Ho) устанавливают соответствие между отдельными процессами (объектами) и множествами подчиненных им процессов (объектов). Они представляются в форме функциональных отображений, областями значений которых являются элементы из булеанов B(P) (B(O)):

$$h_p: P \to B(P),\ \{p_j\} = h_p(p_i),\ p_j \in h_p(p_i),$$

$$h_o: O \to B(O),\ \{o_j\} = h_o(o_i),\ o_j \in h_o(o_i),$$

где $\{p_j\}$ – множество процессов, подчиненных в модели процессу p_i; $\{o_j\}$ – множество объектов, подчиненных в модели объекту o_i.

Таким же образом описываются и отношения взаимодействия $\Re$, задающие входные и выходные инфообъекты процессов:

$$in: B(O) \to P,\ \{o_i\} = in(p_i),\ o_i \in in(p_i),$$

$$out: P \to B(O),\ \{o_j\} = out(p_i),\ o_j \in out(p_i),$$

где $\{o_i\}$ – совокупность входных; $\{o_j\}$ – выходных инфообъектов процесса p_i.

Последний компонент Sвп – отношения следования $\Im$, задает частичный порядок выполнения процессов:

$$\Im: P \to B(P),\ \{p_j\} = \Im(p_i),$$

где $\{p_j\}$ – множество процессов, выполнение которых строго предшествует выполнению процесса p_i ($p_j \in \Im(p_i)$).

С элементами модели (процессами и инфообъектами) связываются:

- множество имен процессов N_p и функция $n_p: P \to N_p$;
- множество имен типов процессов T_p и функция $t_p: P \to T_p$;
- множество имен объектов N_o и функция $n_o: O \to N_o$;

– множество имен типов объектов T_o и функция t_o: $O \rightarrow T_o$;

– множество имен типов отношений иерархии процессов и объектов T_h и функции t_{hp}: $P \rightarrow T_h$, t_{ho}: $O \rightarrow T_h$ (T_h = {& – композиция , V- классификация, *- итерация } $\cup$ N, где N – множество натуральных чисел).

Модель имеет строго древовидную иерархическую структуру, в следствии чего при описании модели процессов и объектов, относящихся одновременно к более чем одному элементу верхнего уровня происходит тиражирование процессов и объектов (появление одноименных элементов-синонимов модели). Такой подход позволяет легко проводить анализ корректности модели процессов, который классифицируется на статический и динамический.

В ходе статического анализа проводится проверка правильности следующих синтаксических конструкций модели: имен процессов и объектов, а также их типов; отношений иерархии, взаимодействия и следования, а также групп отношений. Динамический анализ позволяет установить разрешимость некоторого множества задач в рамках модели, посредством задания формальной системы.

Формализуем понятие функциональный модуль.

Пусть $P = \{p_1,...,p_j,...,p_J\}$ – множество процессов обработки данных, с помощью которых реализуется множество информационных и расчетных задач (ИРЗ).

$O = \{o_1,...,o_t,...,o_T\}$ – множество входных, промежуточных и выходных данных (переменных, объектов), обрабатываемых и преобразуемых процессами из P.

На множествах P и O ($P \cap O = \varnothing$) определены отношения иерархии процессов Hp, иерархии данных Ho, взаимодействия $\Re$ и следования процессов $\Im$.

Заданные таким образом множества и отношения образуют концептуальную схему процессов – Sвп.

Этой схеме можно поставить в соответствие граф обработки данных (ГОД) Г=(V,U). Вершинами $V=\{v_i;\ i=1,R\}$ такого графа являются процессы обработки данных из P, а ребрами $U=\{u_{ij}=(v_i,v_j)\}$ – множество данных из O, общих для соответствующих процессов. Характер соответствия элементов графа Г элементам концептуальной модели определяется по правилу:

$\forall p_r \forall p_s\ (p_r,p_s \in P,\ p_s \in h_p(p_r),\ \{o_t\}= in(p_s),\ \{o_w\}=out(p_s),\ \forall p_m: t_p(p_m)= t_p(p_s)) \Rightarrow$

$\Rightarrow \exists v_i \exists v_j\ (v_i,v_j \in V,\ v_i \neq v_j\ ,\ (v_i,v_j)=u_{ij} \in U,\ p_r \leftrightarrow v_i,\ (p_s,\{p_m\}) \leftrightarrow v_j,\ (\{o_t\},\{o_w\}) \leftrightarrow u_{ij})$

(любым смежным вершинам ГОД ставятся в соответствие два разнотипных процесса из P, связанных отношением иерархии, а инцидентному ребру – множество входных и выходных данных из O подчиненного процесса. Причем процессы-синонимы из P стягиваются в одну вершину ГОД.)

Пусть на булеане множества V – (B(V)={M_q; q=1,2^R-1}; M_q={v_i}) задано некоторое подмножество $M_f(V)$, элементы M_q которого удовлетворяют условиям:

$$\cup M_q = V, \text{ где } M_q \in M_f(V), q=1,Q$$

$$M_q \cap M_{q'} = \varnothing, \text{ где } q \neq q', M_q, M_{q'} \in M_f(V).$$

Множество $M_f(V)$ представляет собой агрегированный граф $G=(\Gamma_q,S)$, вершинами которого являются подграфы $\Gamma_q=(M_q,Д_q)$, (M_q={v_i}, $M_q \in M_f(V)$; $Д_q$={$u_{ij}=(v_i,v_j)$, $v_i,v_j \in M_q$} – множество ребер M_q), а S – множество ребер, связывающие подграфы Γ_q между собой:

$$S = \cup_{q \neq q'} \{u_{ij}=(v_i,v_j),\ v_i \in M_q,\ v_j \in M_{q'}\}.$$

Подграфы $\Gamma_q=(M_q,Д_q)$ называются функциональными модулями графа $\Gamma=(V,U)$ обработки данных.

Множество ребер S графа G образуют межмодульный информационный интерфейс системы модулей ГОД Г.

Построенная таким образом графовая модель полезна при постановке и решении задачи синтеза оптимальной концептуальной схемы модульной структуры.

Концептуальная схема модульного программного обеспечения формально представляется в виде тройки:

$$S_{к\_м} = <Апо, KSм, SVм>,$$

где Апо – совокупность описаний всех элементов модели (имена данных и процессов, их типов, а также типов иерархических отношений); KSм – концептуальная схема модульной структуры; SVм – схема связи модульной структуры.

Концептуальной схемой модульной структуры называется пара:

$$KSм = <NM,SM>,$$

где NM – имя модульной структуры; SM – схема модулей, которая представляется парой:

$$SM = <PSM,PFV>,$$

PSM = {SMq; q=1,Q} – множество схем модулей, где SMq – схема q-го модуля:

$$SMq = <P, O, H, \mathfrak{R}, \mathfrak{I}>$$

PFV={FVqq'; q,q'=$\overline{1,Q}$} – множество отношений, существующих между модулями (Q – количество модулей), где: FVqq' – множество входных и выходных данных процессов, принадлежащих модулю q', и вызываемых из модуля q ($q \neq q'$):

$$FVqq'=\{o_i \mid o_i \in in(p_j) V o_i \in out(p_j),\ p_j \in h_p(p_i); p_i \in SMq,\ p_j \in SMq'\}.$$

Основной задачей, возникающей при разработке модульногопрограммного обеспечения, является получение не произвольного проектного решения, а такого, которое удовлетворяло бы предъявляемым критериям качества.

Основным критерием оценки качества концептуальной схемы модульной структуры считают сложность межмодульного интерфейса.

Межмодульный информационный интерфейс необходимо минимизировать.

Оптимальной концептуальной схемой модульной структуры называется такая KS_M, которая определяет минимальный по сложности межмодульный информационный интерфейс.

Пусть SM_f – некоторая схема модулей для KSм. Множество допустимых схем модулей обозначим через σ ($SM_f \subset \sigma$). Тогда задача синтеза оптимальной концептуальной схемой модульной структуры в общем виде может быть представлена как:

$$SM^o \rightarrow \min_{SMf \subset \sigma} \Omega(SM_f),$$

где $\Omega(SM_f)$ – функция, определенная на множестве допустимых схем модулей σ и отражающая сложность межмодульных информационных связей.

Класс функций $\Omega(SM_f)$ характеризуется нелинейной аналитической зависимостью, включающей большое число переменных, поэтому методы решения, ориентированные на поиск экстремума $\Omega(SM_f)$, представляют существенные алгоритмические и вычислительные трудности. Это обуславливает переход к вспомогательной модели – графу обработки данных (ГОД) Г с последующей постановкой и решением на нем задачи синтеза оптимальной модульной структуры.

Для представления концептуальной схемы модульной структуры KS_M в виде ГОД Г необходимо выполнить взаимооднозначное соответствие между множеством схем модулей SM_f и множеством разбиений $M_f(V)$.

Такое преобразование позволяет представить общий вид задачи синтеза оптимальной модульной структуры в следующей форме:

$$M^o(V) \rightarrow \min_{Mf(V) \subset \sigma} \Psi(M_f(V)),$$

где $\Psi(M_f(V))$ – функция, определенная на множестве допустимых разбиений σ ГОД Г.

Сформулируем задачу синтеза оптимальной концептуальной схемы модульной структуры. Пусть задан взвешенный граф Г=(V,U) обработки данных ($|V| = R$), вершины v_i и ребра u_{ij} которого имеют веса c_i и c_{ij} соответственно, и пусть задан набор чисел$\{m_q\}$, $m_q > 0$, q=1,Q. Требуется найти разбиение $M_f(V) = (M_1,...,M_q,..,M_Q)$ графа на Q модулей – подграфов $<\Gamma_q=(M_q,Д_q)>$, для которых выполнены условия, а также:

$$\sum c_i \le m_q \quad q=1,Q$$

$$v_i \in M_q$$

$$\sum_{q=1}^{Q-1} \sum_{q'=q+1} \sum c_{ij} \rightarrow \min$$

$$v_i \in M_q,\ v_j \in M_q$$

или

$$\sum_{q=1}^{Q} \sum_{v_i, v_j \in M_q} c_{ij} \rightarrow \max$$

Задача аналогична известной в теории графов задаче о разрезании графов.

12. ПРАВИЛА СИНТЕЗА ПРОЦЕДУРНЫХ СПЕЦИФИКАЦИЙ

Под спецификацией программы следует понимать исходное или промежуточное (на пути к программе) описание задачи, которую должна решать проектируемая программа. Таким образом, спецификация – задание для программиста, написанное спецификатором, постановщиком задачи или системным аналитиком, т.е. лицом, профессиональный статус которого явно отличается от статуса программиста. А программа – задание для вычислительной машины, написанное программистом.

Любой программный продукт несет отпечаток той предметной области, для которой он разрабатывается, но в тоже время может быть условно разделен на две части:

– часть, для которой сохраняется соответствие объектам предметной области;

– часть, для которой данное соответствие нарушено вследствие роста степени детализации действий.

Следовательно, задача генерации процедурных спецификаций может рассматриваться как синтез в некотором процедурном языке описания программы в той ее части, которая охватывается моделью вычислительного процесса.

Процедурная спецификация (спецификация алгоритма) строится на основе множеств F – имен операций и U – имен переменных. Элементами процедурной спецификации являются операции, множество которых образуется объединением двух подмножеств A и B, где A – элементарные, а B – составные (структурные) операции.

Для каждой операции a_i, $a_i \in A$ считаются заданными:

– $in_d(a_i)=x_1, x_2, \ldots, x_n$ – упорядоченная совокупность входных переменных ($x_i \in U$);

– $out_d(a_i)=y_1, y_2, \ldots, y_m$ – упорядоченная совокупность выходных переменных ($y_i \in U$);

– $f(a_i) \in F$ – имя выполняемой операции.

Над множеством операций определяются частичные отношения, описывающие возможные передачи управления (частичный порядок выполнения операций):

– $in_c(a_i) \in A$ – множество операций, от которых a_i может получить управление;

– $out_c(a_i) \in A$ – множество операций, которым возможна передача управления от a_i. Для указания конкретного способа передачи или получения управления вводятся признаки $t_{in}(a_i)$, $t_{out}(a_i)$, принимающие значения из {&, V}.

Если для некоторого a_i $t_{in}(a_i)$=&, то выполнение a_i начнется только тогда, когда получено управление от всех a_j, $a_j \in in_c(a_i)$ (синхронизация параллельных вычислительных процессов). При $t_{out}(a_i)$=&, после выполнения a_i управление будет передано всем a_k, $a_k \in out_c(a_i)$ (инициализация параллельного исполнения вычислительных процессов).

Если установлен признак типа передачи (получения) управления $t_{in}(a_i)$=V или $t_{out}(a_i)$=V, то управление от одного из a_j, $a_j \in in_c(a_i)$ передается одному из a_k, $a_k \in out_c(a_i)$.

С составными операциями b_i, $b_i \in B$ связываются некоторые A_i, $A_i \subseteq A$, и U_i, $U_i \subseteq U$ – множества операций и имен переменных. При этом считается, что выполнение образующих ее операций a_j, $a_j \in A_i$ осуществляется в соответствии с задаваемой отношениями InC_i, $OutC_i$, InD_i, $OutD_i$ (которые определены над множествами A_i и U_i) схемой их выполнения.

Для составных операций определены:

– $in_d(b_i)=x_1, x_2, \ldots, x_n$ – упорядоченная совокупность имен переменных, играющих роль входных формальных параметров b_i;

– $out_d(b_i)=y_1, y_2, \ldots, y_m$ – упорядоченнная совокупность имен выходных формальных параметров b_i;

– $f(b_i) \in F$ – имя выполняемой операции.

При этом пара a_i, b_i, для которых справедливо $f(a_i)=f(b_i)$, рассматривается как вызов и определение составной операции.

Для операции $a_i \in A_i$, входящей в некоторую b_i, справедливо

$$\forall u_j, \forall a_j, u_j \in in_c(a_j) \vee u_j \in out_c(a_j) \Rightarrow u_j \in U_i \cup \{x_i\} \cup \{y_i\}$$

(входными и выходными переменными локальных операций могут быть только локальные переменные, указанные в качестве параметров составной операции b_i), а также

$$\forall a_k, \forall a_j, a_k \in in_c(a_j) \vee a_k \in out_c(a_j) \Rightarrow a_k, a_k \in A_i$$

(передачи управления могут осуществляться только между операциями, принадлежащими к A_i – локальным операциям составной b_i).

Таким образом, составная операция представляет собой конструкцию вида:

$b_i = <f(b_i), in_d(b_i), out_d(b_i), A_i, U_i, T_{ui}, H_{ui}, InD_i, OutD_i, InC_i, OutC_i>$

где T_{ui} – множество типов переменных из U_i ($u_x \in U_i$, $t_u(u_x) \in T_{ui}$); H_{ui} – отношения иерархии переменных ($H_{ui} \subseteq U_i \times B(U_i)$); InD_i, $OutD_i$, InC_i, $OutC_i$ – множество отношений передач, получения и информационной зависимости, определенные над множествами A_i и U_i.

В соответствии с введенными конструкциями, задача генерации процедурной спецификации может быть подготовлена как построение множества B (порождение для каждого шаблона составной операции-элемента B).

Рассмотрим правила синтеза спецификации.

Для каждого информационного объекта, входящего в рассматриваемый относительно некоторого процесса p_i шаблон, вводится переменная:

$$\forall o_k,\ o_k \in in(p_j) \vee o_k \in out(p_j),\ p_j \in h_p(p_i) \Rightarrow$$
$$\Rightarrow U_i = U_i \bigcup \{u_k\} \bigcup t_u(u_k) = t_o(o_k).$$

Тип вводимой переменной совпадает с типом объекта, которому она ставится в соответствие. Для любого процесса p_i, входящего в шаблон на нижнем уровне, на алгоритмическом уровне в множество A_i вводится операция a_j:

$$\forall p_j,\ p_j \in h_p(p_i),\ t_{hp}(p_i) = \& \Rightarrow A_i = A_i \bigcup \{a_j\},\ f(a_j) = t_p(p_j),$$
$$in_d(a_j) = \{u_x : o_x \to u_x,\ o_x \in in(p_j),\ t_u(u_x) = t_o(o_x)\},\ U_i = U_i \bigcup \{u_x\},$$
$$out_d(a_j) = \{u_y : o_y \to u_y,\ o_y \in out(p_j),\ t_u(u_y) = t_o(o_y)\},\ U_i = U_i \bigcup \{u_y\}.$$

При этом имя операции a_j совпадает с именем типа процесса p_j, а в качестве входных и выходных переменных операции указываются переменные, поставленные в соответствие входным и выходным объектам процесса p_j.

Для процесса-владельца шаблона, который имеет иерархическое отношение со своими подпроцессами типа композиция ($t_{hp}(p_i)$=&), порождается составная операция b_i, имя которой совпадает с именем типа процесса $f(b_i)=t_{hp}(p_i)$, а входные и выходные переменные которой могут быть определены двояким образом. Генерация может осуществляться в предположении, что некоторый o_i, $o_i \in in(p_i)$ или $o_i \in out(p_i)$, является концептуальным объектом и в реализации представляется просто совокупностью своих элементов:

$$\forall o_i,\ o_i \in in(p_i),\ o_i \notin O_f \Rightarrow \{u_k\} \in in_d(b_i),$$

$$\forall o_j,\ o_j \in out(p_i),\ o_j \notin O_f \Rightarrow \{u_y\} \in out_d(b_y),$$

где $\{u_k\colon o_k \to u_k,\ o_k \in h_o(o_i)\}$; $\{u_y\colon o_y \to u_y,\ o_y \in h_o(o_j)\}$.

Альтернативным вариантом является физическая реализация o_i, состоящая во введении некоторой структурной переменной u_i:

$$\forall o_i,\ o_i \in in(p_i),\ o_i \in O_f \Rightarrow U_i = U_i \bigcup \{u_i\},\ u_i \in in_d(b_i)$$

$$\forall o_j,\ o_j \in out(p_i),\ o_j \in O_f \Rightarrow U_i = U_i \bigcup \{u_j\},\ u_j \in out_d(b_i),$$

а на множестве имен переменных устанавливается отношение иерархии:

$$H_{ui} = H_{ui} \bigcup h_u(u_i;\ u_{i1},\ u_{i2},\ \dots,\ u_{ik}),,$$

где $o_i \to u_i$, $o_{i1} \to u_{i1}$, ... , $o_{ik} \to u_{ik}$; o_{i1}, ... , $o_{ik} \in h_o(o_i)$ (вводится структурная переменная, состоящая из компонент переменных, поставленных в соответствие объектам, подчиненным реализуемому o_i).

Процессы, для которых иерархическое отношение, связывающее их с множеством подчиненных, имеет тип классификация $t_{hp}(p_i)=V$ могут рассматриваться как процедуры выбора альтернативной ветви вычислений. Им могут быть поставлены в соответствие, помимо составной b_i, операция, производящая такой анализ и передающая управление в одну из ветвей, а также операция, производящая слияние ветвей после их выполнения:

$$\forall p_i,\ t_{hp}(p_i) = V \Rightarrow A_i = A_i \bigcup \{a_{ib},\ a_{ie}\},$$

$$out_c(a_{ib}) = \{a_x\colon p_x \to a_x,\ p_x \in h_p(p_i)\},\ t_{out}(a_{ib}) = V,$$

$$in_c(a_{ie}) = \{a_y\colon p_y \to a_y,\ p_y \in h_p(p_i)\},\ t_{in}(a_{ie}) = V.$$

Входные и выходные переменные операции b_i, соответствующей p_i, $t_{hp}(p_i)=V$, определяются также, как и ранее, для них вводятся иерархические отношения, но типом иерархии в данном случае будет V, иначе говоря, иерархия описывает не компоненты переменной верхнего уровня, а ее вариантные части (по аналогии со структурой записи с вариантами в языках программирования).

Аналогично, для p_i, $t_{hp}(p_i) = *$, могут быть введены операции:

$$\forall p_i,\ t_{hp}(p_i) = *\ \Rightarrow\ A_i = A_i \bigcup \{a_{ib_1},\ a_{ib_2},\ a_{ie}\};$$

$$out_c(a_{ib_1}) = \{a_{ib_2}\};$$

$$in_c(a_{ib_2}) = \{a_{ib_1}, a_{ie}\},\ t_{in}(a_{ib_2}) = V;$$

$$out_c(a_{ib_2}) = \{a_x: p_x \to a_x,\ p_x \in h_p(p_i)\},\ t_{out}(a_{ib_2}) = \&;$$

$$in_c(a_{ie}) = \{a_y: p_y \to a_y,\ p_y \in h_p(p_i)\},\ t_{in}(a_{ie}) = \&;$$

$$a_{ib_2} \in out(a_{ie}),\ a_{ie} \in in(a_{ib_2}).$$

Отличие здесь состоит в типах передач управления &: в общем случае должны быть выполнены все операции, образующие цикл, при этом после выполнения завершающей a_{ie}, управление может быть передано на начальную a_{ib_2}, которая получает его либо при входе, либо при выходе из цикла ($t_{in}(a_{ib_2}) = V$).

13. ПРИМЕР МОДЕЛИРОВАНИЯ КЛАССОВ ДЛЯ ОЦЕНИВАНИЯ КАЧЕСТВА ПРОГРАММНЫХ ПРОДУКТОВ

Диаграммы классов (class diagram), которые являются составляющими модели классов, позволяют создавать логическое представление системы, на основе которого создается исходный код описанных классов. На диаграммах классов изображаются также атрибуты классов, операции классов. Каждый класс на диаграмме выглядит в виде прямоугольника, разделенного на три части. В первой содержится имя класса, во второй – его атрибуты. В последней части содержатся операции класса, отражающие его поведение (действия, выполняемые классом) [7].

Диаграмма классов информационной системы оценки качества программных продуктов изображена на рис. 16.

Диаграмма классов включает в себя следующие классы:

- класс CountingUnit (вычислительный элемент);
- класс Criteria (критерий);
- класс DoubleSlider (ползунок с вещественным числом);
- класс Factor (фактор);
- класс Level (уровень);
- класс LoginDialog (окно авторизации);
- класс Metric (метрика);
- класс NewProjectDialog (окно создания нового проекта);
- класс Output (вывод);
- класс Phase (фаза);
- класс QualityRating (оценка качества);
- класс RatingSet (набор оценок);
- класс RatingUnit (оценочный элемент);
- класс Setting (настройка);
- класс SettingGroup (группа настроек);
- класс WeightSetting (окно весовых настроек).

Класс CountingUnit имеет следующие атрибуты:

- «typeOfUnit» – тип вычислительного элемента;

– «paramsAmount» – количество параметров;

– «numbersVector» – массив чисел, которые используются для расчета значения элемента;

– «resultSlider» – объект ползунка, привязанного к вычислительному элементу.

Так же класс CountingUnit имеет следующие операции:

– «CountingUnit()» – конструктор класса, инициализирующий объект;

– «slider()» – выдача доступа к графическому объекту ползунка, привязанного к вычислительному элементу;

– «saveValue()» – считывает данные с полей и сохраняет в объект для последующих расчетов;

– «calculate()» – рассчитывает значение показателя и устанавливает его на ползунок.

Класс Output имеет следующие атрибуты:

– «content» – графический объект для отображения отчета;

– «activeRS» – ссылка на активный набор оценок;

– «activePhase» – номер активной фазы жизненного цикла;

– «activeFactor» – номер активного фактора;

– «type» – тип отчета;

– «fileNumber» – количество файлов для сохранения.

Также класс Output имеет следующие операции:

– «Output()» – конструктор класса, инициализирующий объект;

– «setFullContent()» – генерирует полный отчет и устанавливает его в объект отображения;

– «setFactorContent()» – генерирует отчет по фактору и устанавливает его в объект отображения;

– «setPhaseContent()» – генерирует отчет по фазе жизненного цикла и устанавливает его в объект отображения;

– «saveAsHTML()» – сохраняет отчет в формате HTML;

– «saveAsDoc()» – сохраняет отчет в формате DOC;

– «saveAsODT()» – сохраняет отчет в формате ODT;

– «changeFilesNumber()» – изменяет количество файлов для сохранения отчета;

– «makeMimetypeFile()» – генерирует файл типов данных для сборки отчета в формате ODT;

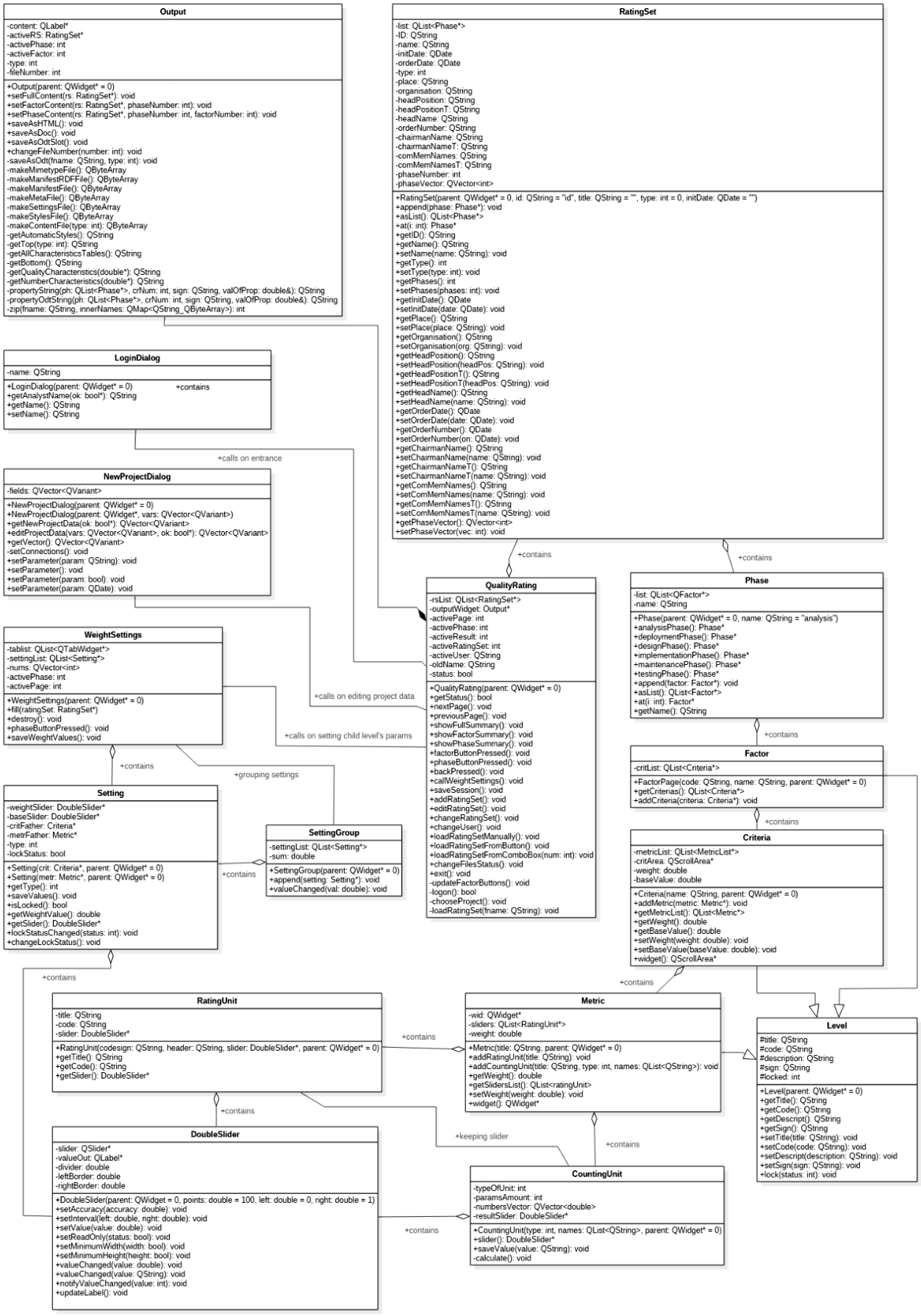


Рис. 16. Диаграмма классов информационной системы оценки качества программных продуктов

– «makeManifestRDFFile()», «makeManifestFile()» – генерируют файлы-манифесты для сборки отчета в формате ODT;

– «makeMetaFile()» – генерирует файл мета-данных для сборки отчета в формате ODT;

– «makeSettingsFile()» – генерирует файл настроек для сборки отчета в формате ODT;

– «makeStylesFile()» – генерирует файл стилей для сборки отчета в формате ODT;

– «makeContentFile()» – генерирует файл контента (содержимого документа) для сборки отчета в формате ODT;

– «getAutomaticStyles()» – генерирует автоматические стили для включения в файл контента;

– «getTop()», «getBottom()» – генерируют титульные листы и страницы заключения для включения в файл контента;

– «getAllCharacteristicsTables()», «getQualityCharacteristics()», «getNumberCharacteristics()» – генерируют сводные таблицы значений показателей для включения в файл контента;

– «propertyString()», «propertyOdtString()» – функции, генерирующие отдельные строки в сводных таблицах;

– «zip()» – собирает отчет в формате ODT.

– Класс RatingSet имеет следующие атрибуты:

– «list» – список, содержащий объекты фаз;

– «ID» – номер набора оценок;

– «name» – название программного изделия;

– «initDate» – дата начала процедуры оценивания качества программного изделия;

– «orderDate» – дата приказа об организации оценивания качества программного изделия;

– «type» – тип программного изделия, подлежащего оценке;

– «place» – место (организация), где проходит оценивания качества программного изделия;

– «organisation» – организация-заказчик процедуры оценивания качества программного изделия;

– «headPosition» – должность руководителя организации-заказчика;

– «headName» – имя руководителя организации-заказчика;

– «orderNumber» – номер приказа об организации оценивания качества программного изделия;

– «chairmanName» – имя и должность председателя комиссии, проводящей оценку качества программного изделия;

– «comMemNames» – имена и должности членов комиссии, проводящей оценку качества программного изделия;

– «phaseNumber» – фазы жизненного цикла, для которых производится оценка качества, в виде одного числа;

– «phaseVector» – фазы жизненного цикла, для которых производится оценка качества, в виде массива чисел.

Так же класс RatingSet имеет следующие операции:

– «RatingSet()» – конструктор класса, инициализирующий объект;

– «append()» – добавляет в объект набора оценок объект фазы жизненного цикла;

– «asList()» – предоставляет список объектов фаз жизненного цикла, содержащихся в наборе оценок;

– «at()» – предоставляет объект фазы жизненного цикла под указанным номером;

– «getX()» – возвращает значение выбранного параметра X (имя параметра совпадает с именем соответствующего атрибута);

– «setX()» – устанавливает значение выбранного параметра X (имя параметра совпадает с именем соответствующего атрибута).

Класс LoginDialog имеет следующие атрибуты:

– «name» – имя аналитика.

– Так же класс LoginDialog имеет следующие операции:

– «LoginDialog()» – конструктор класса, инициализирующий объект;

– «getAnalystName()» – запрашивает у пользователя имя аналитика для авторизации;

– «getName()» – возвращает введенное пользователем имя;

– «setName()» – устанавливает значение имени.

– Класс NewProjectDialog имеет следующие атрибуты:

– «fields» – список значений параметров, полученных из полей ввода.

Так же класс NewProjectDialog имеет следующие операции:

– «NewProjectDialog()» – конструктор класса, инициализирующий объект;

– «getNewProjectData()» – запрашивает у пользователя различные параметры для создания нового проекта;

– «editProjectData()» – запрашивает у пользователя различные параметры для редактирования текущего проекта;

– «getVector()» – возвращает список значений параметров, полученных из полей ввода;

– «setConnections()» – связывает поля ввода с соответствующими параметрами во внутреннем списке параметров;

– «setParameter()» – установка значения определенного параметра из списка параметров.

Класс QualityRating имеет следующие атрибуты:

– «rsList» – список наборов оценок, загруженных в данный момент в программу;

– «outputWidget» – графический объект для вывода отчета;

– «activePage» – номер активного фактора;

– «activePhase» – номер активной фазы жизненного цикла;

– «activeResult» – флаг вывода отчета на экран;

– «activeRatingSet» – номер активного набора оценок;

– «activeUser» – имя авторизовавшегося пользователя;

– «oldName» – сохраненное имя проекта;

– «status» – флаг статуса программы.

Так же класс QualityRating имеет следующие операции:

– «QualityRating()» – конструктор класса, инициализирующий объект;

– «getStatus()» – возвращает значение статуса программы;

– «nextPage()», «previousPage()» – управляют переключениями страниц в окне программы;

– «showFullSummary()», «showFactorSummary()», «showPhaseSummary()» – выводят отчет определенного типа в окно программы;

– «factorButtonPressed()», «phaseButtonPressed()», «backPressed()» – обновляют окно программы после выбора фактора, фазы жизненного цикла или нажатия кнопки «Назад»;

– «callWeightSettings()» – вызывает окно редактирования значений весовых коэффициентов и базовых значений показателей;

– «saveSession()» – сохраняет состояние текущего проекта в файл;

– «addRatingSet()» – вызывает окно создания нового проекта и инициализирует новый проект с полученными значениями параметров;

– «editRatingSet()» – вызывает окно редактирования проекта и сохраняет полученные значения параметров в проект;

– «changeRatingSet()» – очищает окно программы и начинает процедуру выбора проекта;

– «changeUser()» – очищает окно программы и начинает процедуру авторизации;

– «loadRatingSetManually()» – загружает набор оценок из выбранного пользователем файла;

– «loadRatingSetFromButton()», «loadRatingSetFromComboBox()» – загружает набор оценок по нажатию на кнопку или выбору элемента из выпадающего списка;

– «changeFilesStatus()» – фиксирует значения галочек количества файлов для генерации отчета;

– «exit()» – производит завершение работы программы;

– «updateFactorButtons()» – обновляет внешний вид кнопок выбора факторов;

– «logon()» – производит процедуру авторизации, вызывая при этом окно выбора пользователя;

– «chooseProject()» – производит процедуру выбора проекта, вызывая окно управления проектами;

– «loadRatingSet()» – управляет загрузкой проекта в окно программы.

Класс WeightSettings имеет следующие атрибуты:

– «tablist» – список вкладок, в которых расположены ползунки настроек базовых значений и значений весовых коэффициентов;

– «settingList» – список настроек (объектов настройки базовых значений и значений весовых коэффициентов);

– «nums» – список номеров фаз, для которых ведется оценка качества программного изделия;

– «activePhase» – номер активной фазы жизненного цикла;

– «activeFactor» – номер активного фактора.

Так же класс WeightsSettings имеет следующие операции:

– «WeightSettings()» – конструктор класса, инициализирующий объект;

– «fill()» – загружает в окно настройки базовых значений и значений весовых коэффициентов содержимое;

– «destroy()» – уничтожает окно после его закрытия;

– «phaseButtonPressed()» – обновляет окно программы после выбора фазы жизненного цикла;

– «saveWeightValues()» – сохраняет в объекты уровней оценивания новые значения базовых значений и весовых коэффициентов.

Класс Setting имеет следующие атрибуты:

– «weightSlider» – указатель на объект ползунка настройки значения весового коэффициента;

– «baseSlider» – указатель на объект ползунка настройки базового значения;

– «critFather» – указатель на объект критерия, который связан с данным объектом настройки;

– «metrFather» – указатель на объект метрики, которая связан с данным объектом настройки;

– «type» – тип настройки (для метрики или критерия);

– «lockStatus» – статус блокировки изменения значения настройки.

Также класс Setting имеет следующие операции:

– «Setting()» – конструктор класса, инициализирующий объект;

– «getType()» – сообщает о типа данной настройки;

– «saveValues()» – сохраняет значения настроек в связанный с ним объект уровня оценивания (метрики или критерия);

– «isLocked()» – сообщает о статусе блокировки настройки;

– «getWeightValue()» – возвращает текущее значение весового коэффициента;

– «getSlider()» – предоставляет доступ к ползунку настройки значения;

– «lockStatusChanged()» – отправка сообщения об изменении статуса состояния галочки блокировки настройки;

– «changeLockStatus()» – управляет блокировкой настройки при изменении состояния галочки блокировки настройки.

Класс SettingGroup имеет следующие атрибуты:

– «settingList» – список объектов настроек, которым управляет объект группы;

– «sum» – текущая сумма значений настроек внутри группы.

Также класс SettingGroup имеет следующие операции:

– «SettingGroup()» – конструктор класса, инициализирующий объект;

– «append()» – добавление объекта настройки в группу;

– «valueChanged()» – управляет проверкой условий соответствия значений настроек внутри группы.

Класс RatingUnit имеет следующие атрибуты:

– «title» – название оценочного элемента;

– «code» – код оценочного элемента;

– «slider» – ссылка на объект ползунка, связанного с данным оценочным элементом.

Также класс RatingUnit имеет следующие операции:

– «RatingUnit()» – конструктор класса, инициализирующий объект;

– «getTitle()» – возвращает название оценочного элемента;

– «getCode()» – возвращает код оценочного элемента;

– «getSlider()» – возвращает ссылку на объект ползунка, связанного с данным оценочным элементом.

Класс Double Slider имеет следующие атрибуты:

– «divider» – делитель, используемый для перевода целых чисел в дробные и наоборот;

– «valueOut» – графический объект вывода текущего значения ползунка;

– «leftBorder», «rightBorder» – левая и правая границы интервала возможных значений ползунка;

– «slider» – ссылка на объект графический объект ползунка.

Также класс DoubleSlider имеет следующие операции:

– «DoubleSlider()» – конструктор класса, инициализирующий объект;

– «setAccuracy()» – устанавливает точность (количество знаков после запятой) значения ползунка;

– «setInterval()» – устанавливает границы интервала возможных значений ползунка;

– «getValue()» – устанавливает значение ползунка;

– «setReadOnly()» – устанавливает значение статуса «Только для чтения»;

– «setMinimumWidth()», «setMinimumHeight()» – устанавливают минимальные размеры графического объекта ползунка;

– «valueChanged()» – сообщает об изменении значения ползунка;

– «notifyValueChanged()» – переадресовывает изменение внутреннего объекта ползунка в родительский объект;

– «updateLabel()» – обновляет графический объект вывода текущего значения ползунка.

Класс Phase имеет следующие атрибуты:

– «list» – содержит список объектов факторов, включенных в данную фазу жизненного цикла;

– «name» – название фазы жизненного цикла.

Также класс Phase имеет следующие операции:

– «Phase()» – конструктор класса, инициализирующий объект;

– «append()» – добавляет в фазу жизненного цикла объект фактора;

– «asList()» – предоставляет доступ к списку объектов факторов включенных в данную фазу жизненного цикла;

– «at()» – предоставляет объект фактора под указанным номером;

– «XPhase()» – возвращает заполненный объект фазы определенного типа, где X – тип (X может принимать значения analysis, design, implementation, testing, deployment, maintenance);

– «getName()» – возвращает имя фазы жизненного цикла.

Класс Level имеет следующие атрибуты:

– «title» – название показателя;

– «code» – код показателя;

– «description» – описание показателя;

– «sign» – знаковое обозначение показателя;

– «locked» – статус блокировки изменения показателя.

Также класс Level имеет следующие операции:

– «Level()» – конструктор класса, инициализирующий объект;

– «getX()» – возвращает значение выбранного параметра X (имя параметра совпадает с именем соответствующего атрибута);

– «setX()» – устанавливает значение выбранного параметра X (имя параметра совпадает с именем соответствующего атрибута).

– «lock()» – изменяет состояние блокировки изменения показателя.

Класс Factor наследует все атрибуты и методы от класса Level, а также содержит следующие атрибуты:

– «critList» – список критериев, которые содержит данный фактор.

Также класс Factor имеет следующие операции:

– «Factor()» – конструктор класса, инициализирующий объект;

– «getCriterias()» – предоставляет доступ к списку критериев, содержащихся в факторе;

– «addCriteria()» – добавляет объект критерия в фактор.

Класс Criteria наследует все атрибуты и методы от класса Level, а также содержит следующие атрибуты:

– «metricList» – список метрик, которые содержит данный критерий;

– «critArea» – графический объект отображения состояния объекта критерия;

– «weight» – значение весового коэффициента критерия;

– «baseValue» – базовое значение критерия.

Так же класс Criteria имеет следующие операции:

– «Criteria()» – конструктор класса, инициализирующий объект;

– «getMetricList()» – предоставляет доступ к списку метрик, содержащихся в критерии;

– «addMetric()» – добавляет объект метрики в критерий;

– «getWeight()» – сообщает текущее значение весового коэффициента;

– «getBaseValue()» – сообщает текущее базовое значение;

– «setWeight()» – устанавливает новое значение весового коэффициента;

– «setBaseValue()» – устанавливает новое базовое значение;

– «widget()» – предоставляет доступ к графическому объекту отображения состояния объекта критерия.

Класс Metric наследует все атрибуты и методы от класса Level, а также содержит следующие атрибуты:

– «sliders» – список указателей на объекты ползунков оценочных элементов, привязанных к данной метрике;

– «widget» – графический объект отображения представления метрики;

– «weight» – значение весового коэффициента метрики.

Также класс Metric имеет следующие операции:

– «Metric()» – конструктор класса, инициализирующий объект;

– «getSlidersList()» – предоставляет доступ к списку указателей на объекты ползунков оценочных элементов, привязанных к данной метрике;

– «addRatingUnit()» – добавляет оценочный элемент в данную метрику;

– «addCountingUnit()» – добавляет вычислительный элемент в данную метрику;

– «getWeight()» – сообщает текущее значение весового коэффициента;

– «setWeight()» – устанавливает новое значение весового коэффициента;

– «widget()» – предоставляет доступ к графическому объекту отображения состояния объекта критерия.

Связи между классами:

– класс QualityRating связан посредством ассоциативной связи с классами LoginDialog, NewProjectDialog, WeightSettings, а посредством связи типа «агрегация» связан с классами Output и RatingSet;

– класс RatingSet связан посредством связи типа «агрегация» с классом Phase;

– класс Phase связан посредством связи типа «агрегация» с классом Factor;

– класс Factor связан посредством связи типа «агрегация» с классом Criteria, а также с классом Level посредством связи типа «обобщение»;

– класс Criteria связан посредством связи типа «агрегация» с классом Metric, а также с классом Level посредством связи типа «обобщение»;

– класс Metric связан посредством связи типа «агрегация» с классами RatingUnit и CountingUnit, а также с классом Level посредством связи типа «обобщение»;

– класс RatingUnit связан посредством связи типа «агрегация» с классом DoubleSlider, а также с классом CountingUnit посредством связи типа «ассоциация»;

– класс CountingUnit связан посредством связи типа «агрегация» с классом DoubleSlider;

– класс WeightSettings связан посредством связи типа «агрегация» с классом Setting, а также с классом SettingGroup посредством связи типа «ассоциация»;

– класс SettingGroup связан посредством связи типа «агрегация» с классом Setting;

– класс Setting связан посредством связи типа «агрегация» с классом DoubleSlider.

14. ПРИМЕР ФРАГМЕНТОВ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ ПРОИЗВОДСТВЕННОГО ПРЕДПРИЯТИЯ

Произведен объектно-ориентированный анализ процесса функционирования производственного предприятия. На рисунке 17 приведена диаграмма, характеризующая деятельность производственного предприятия в форме *Business Use Case*.

На рисунке 18 приведена диаграмма, характеризующая деятельность ИС производственного предприятия в форме *Use Case Model*.

На рисунке 19 приведена диаграмма классов, подсистемы оперативно-календарного планирования ИС производственного предприятия.

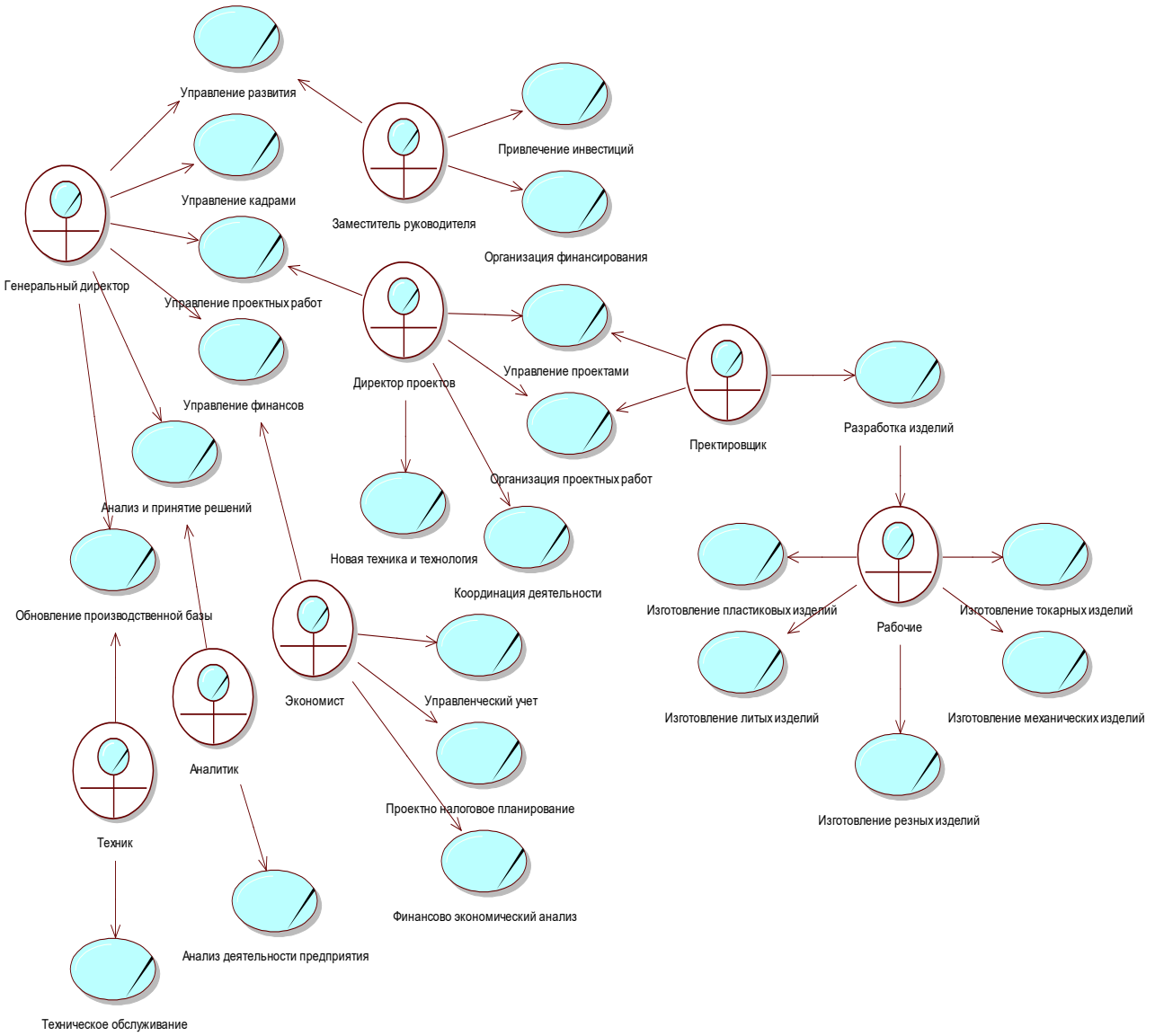


Рис. 17. Диаграмма *Business Use Case*

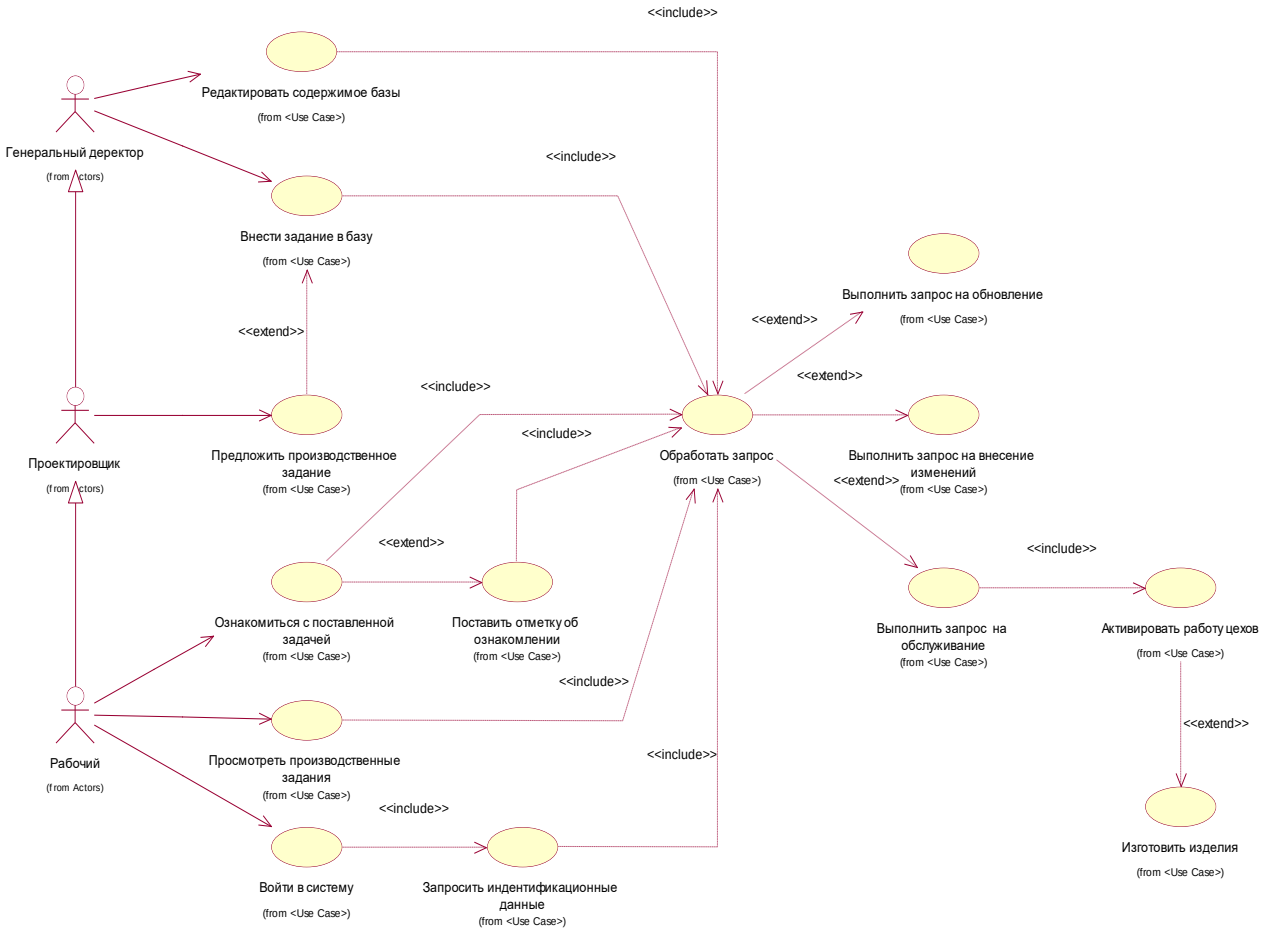


Рис. 18. Диаграмма *Use Case Model*

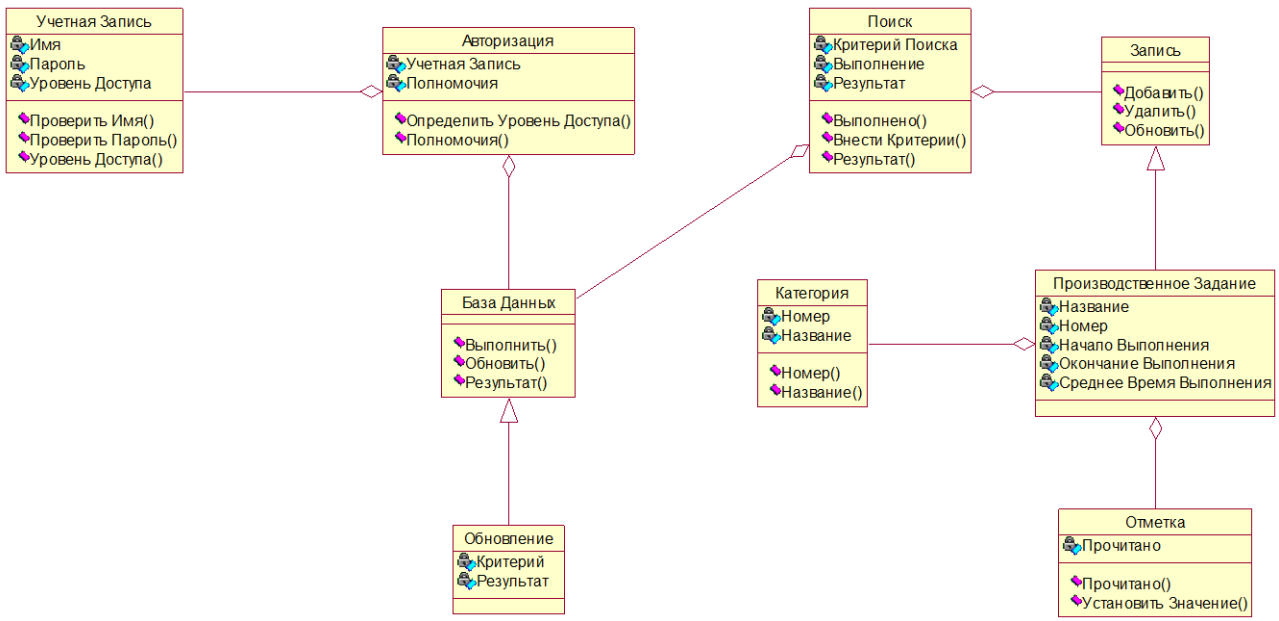


Рис. 19. Диаграмма классов

Учетная Запись – отвечает за хранение текущей учетной записи рабочего, списка возможных его действий и обеспечивает необходимые операции над данными или производственными заданиями.

Авторизация – выполняет включение/выключение возможностей системы для различных уровней доступа.

База Данных – отвечает за работу с базой.

Обновление – специализированный класс для запросов на обновление в базу данных.

Поиск – хранит список записей из базы данных, критерии для выбора и методы для внесения и изъятия данных, запуска поиска и определения статуса.

Запись – хранит конкретную запись из базы.

Производственное Задание – расширяет класс запись для работы с производственными заданиями, позволяющий просмотреть начало работы, окончание и продолжительность производственного процесса.

Категория – номер и название категории, с которой осуществляется работа.

Отметка – установка отметки о прочтении.

На рисунке 20 приведена диаграмма классов пользовательского интерфейса.

На рисунке 21 приведена диаграмма деятельности, которая раскрывает детали алгоритмической реализации операций, выполняемых системой. Диаграммы деятельности применяются для визуализации алгоритмов, по которым работают операции классов.

На рисунке 22 приведена диаграмма состояний, применяемая для пояснения работы сложных объектов, а точнее переходов их из одного состояния в другое.

Начальное состояние системы: открыто главное меню, ожидается действие рабочего.

Далее возможно несколько вариантов действий:

– вызов окна авторизации: система ожидает ввода данных, проверяет введенную информацию и либо открывает доступ, либо запрашивает повторное введение;

– если осуществляется вызов окна поиска по производственным заданиям, то система ожидает ввода критериев, после чего осуществляет поиск и отображает найденный результат;

– если осуществляется вызов окна внесения производственного задания на рассмотрение, то после ввода данных ожидается подтверждение правильности;

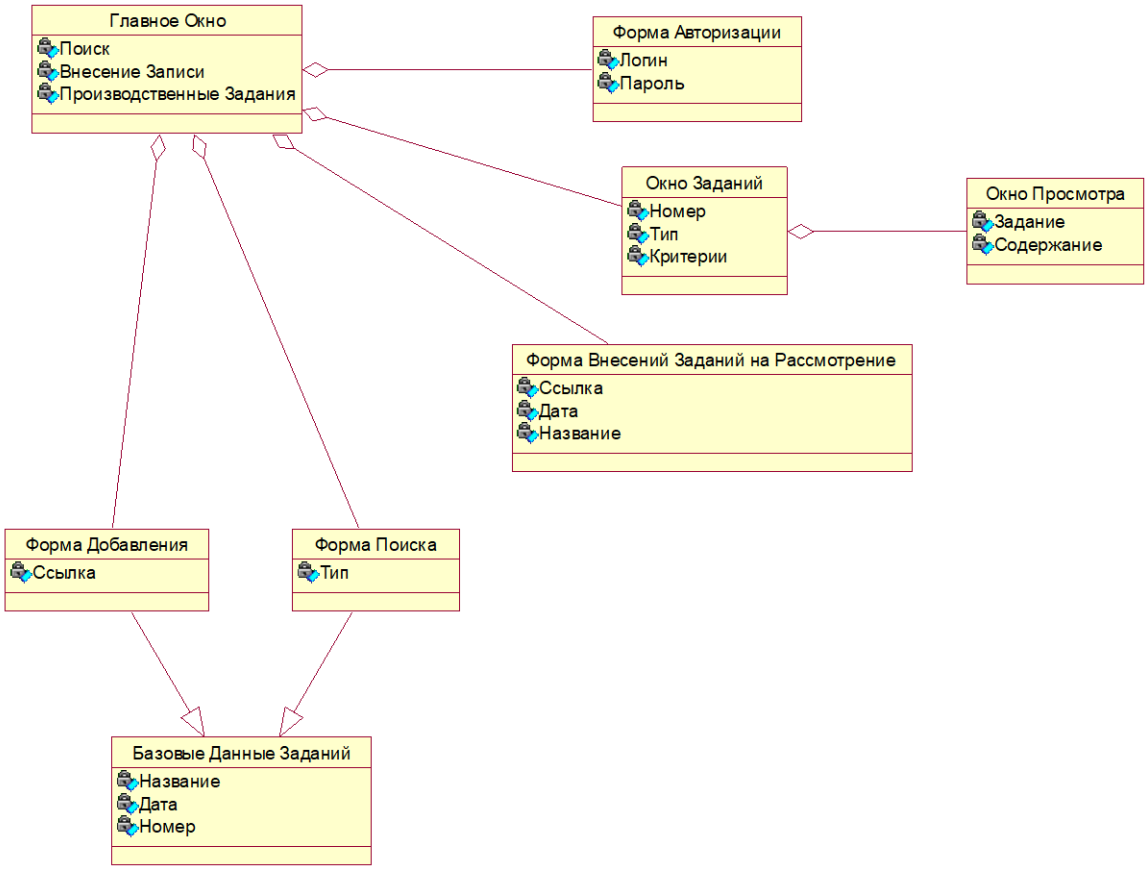


Рис. 20. Диаграмма классов пользовательского интерфейса

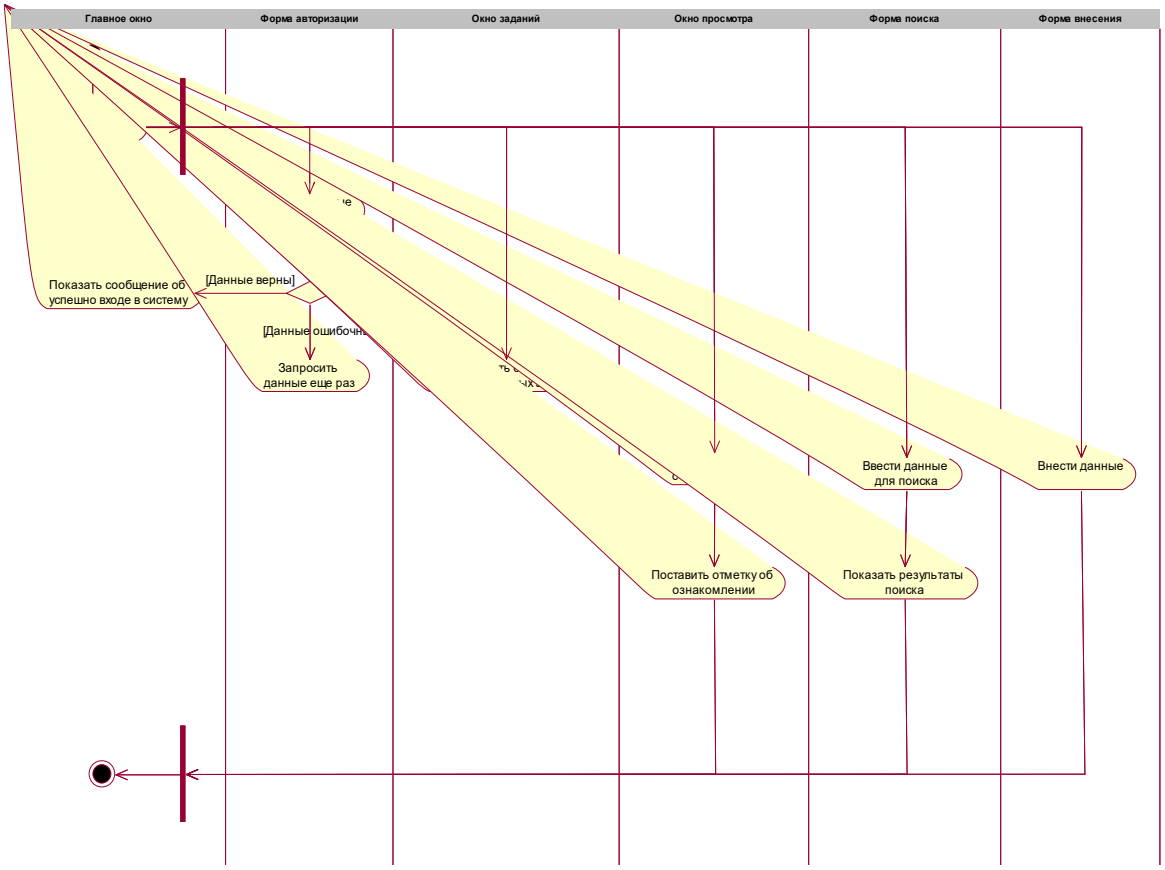


Рис. 21. Диаграмма деятельности

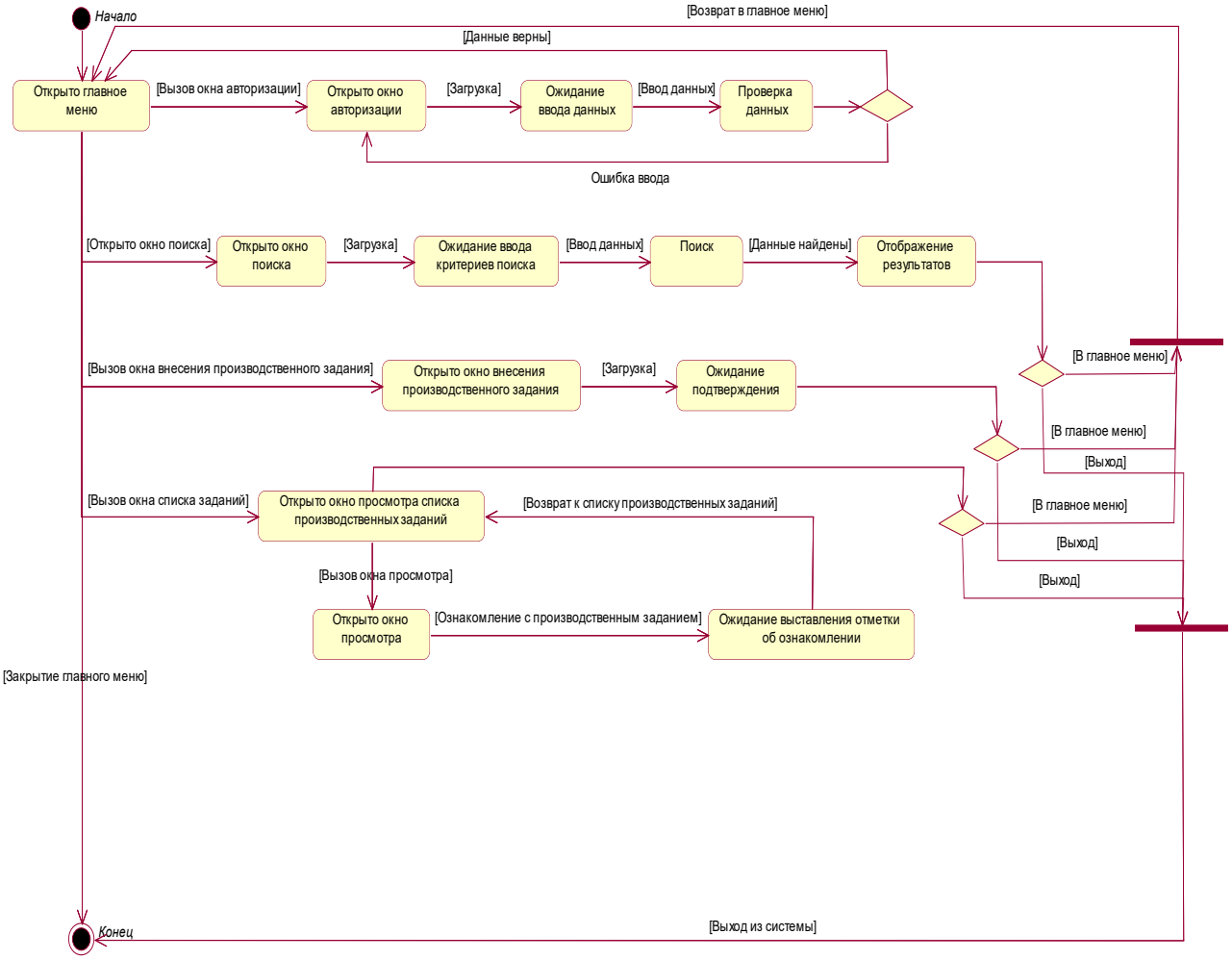


Рис. 22. Диаграмма состояний

– если осуществляется вызов списка производственных заданий, то после система ожидает выбора интересующего задания и выставление отметки об ознакомление, после непосредственного прочтения инструкций.

В любом из вариантов после выполнения действий система может перейти либо к главному окну, либо выполнить выход, в зависимости от действий рабочего.

На рисунке 23 приведена диаграмма размещения, которая отображает технологическую конфигурацию системы.

Сервер баз данных – содержит данные о сотрудниках, производственных заданиях, различной номенклатуры изделий и т.д.

Технологический сервер – содержит данные о видах производства.

Коммутатор – распределяет производственные задания по рабочим станциям.

Диаграммы последовательности, кооперации и компонентов приведены на рис. 24 – 36.

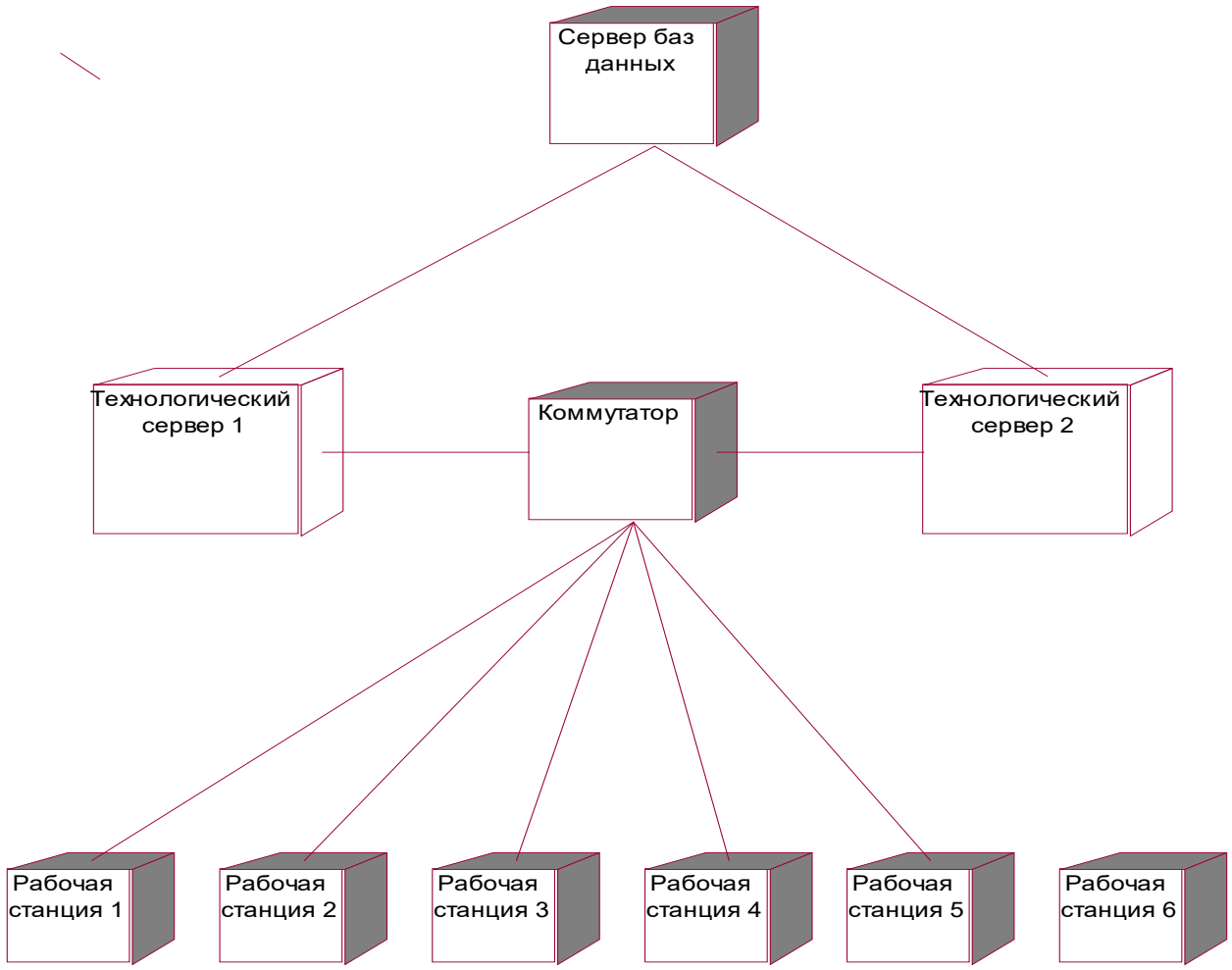


Рис. 23. Диаграмма размещения

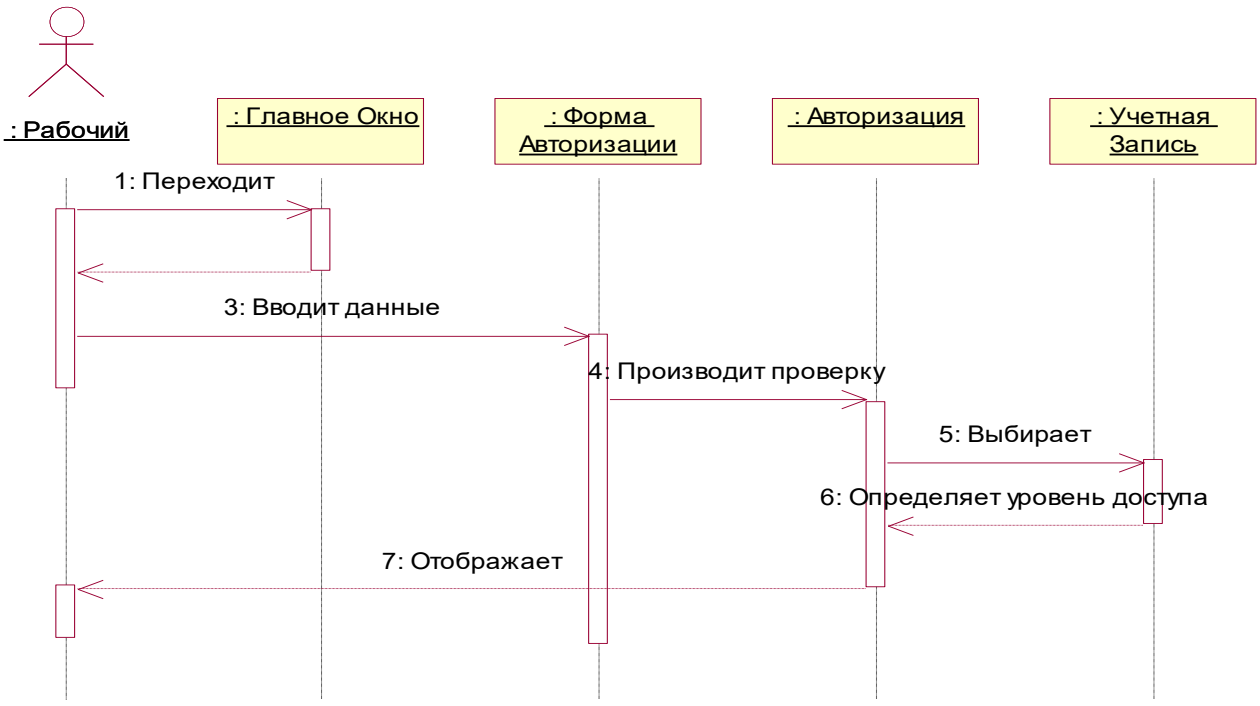


Рис. 24. Диаграмма последовательности входа в систему

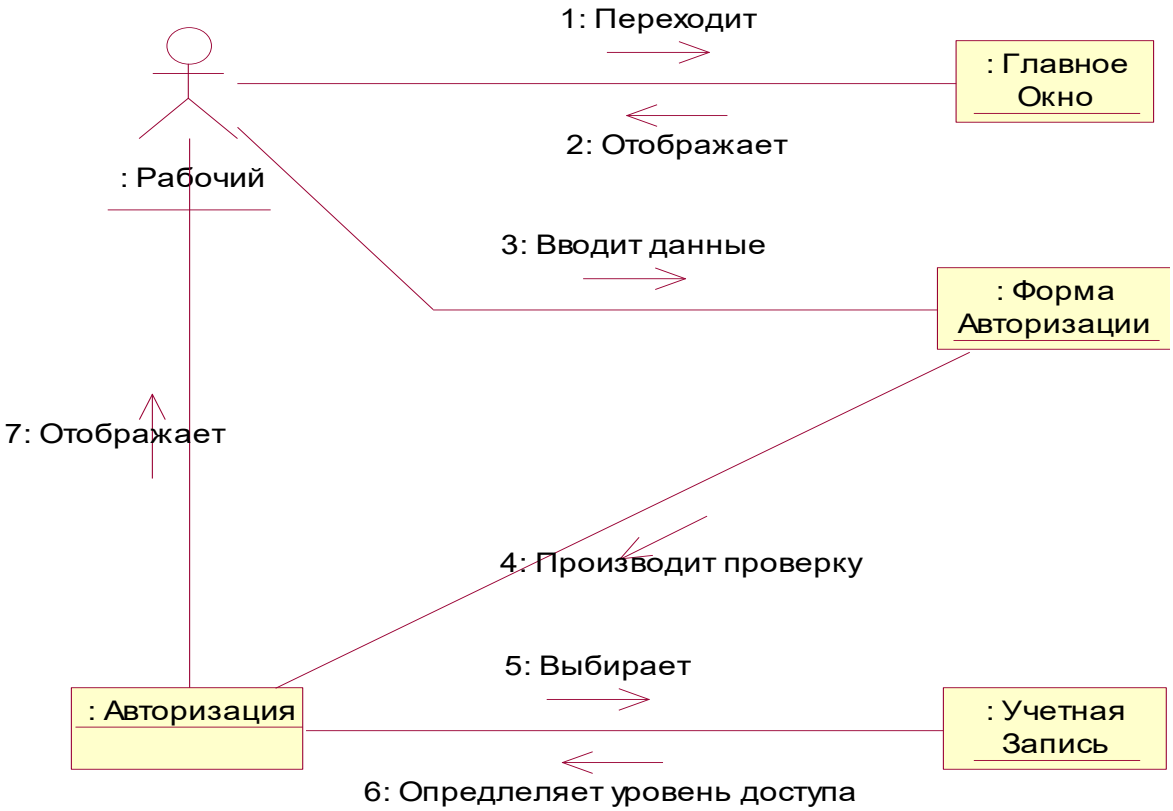


Рис. 25. Диаграмма кооперации входа в систему

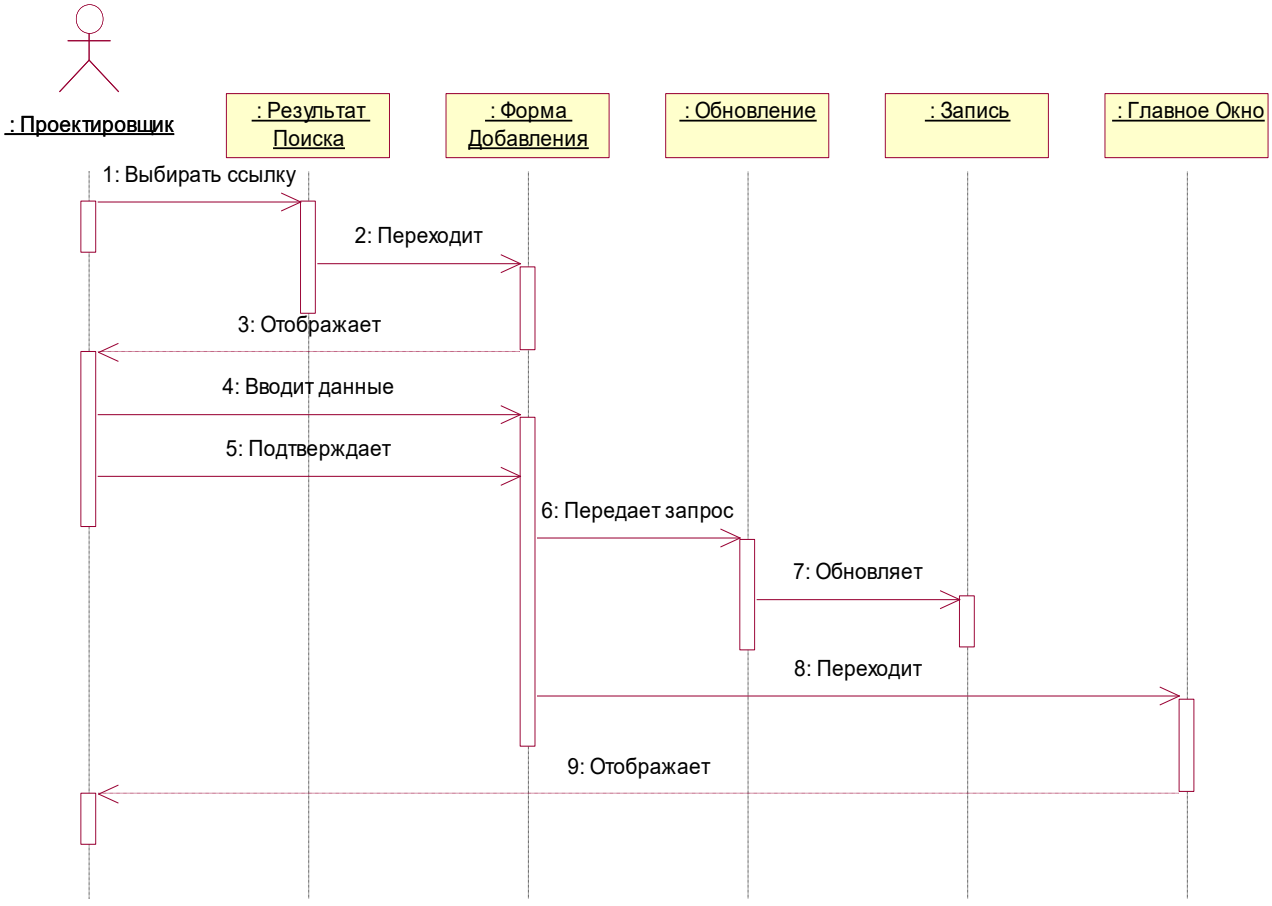


Рис. 26. Диаграмма последовательности добавления производственного задания

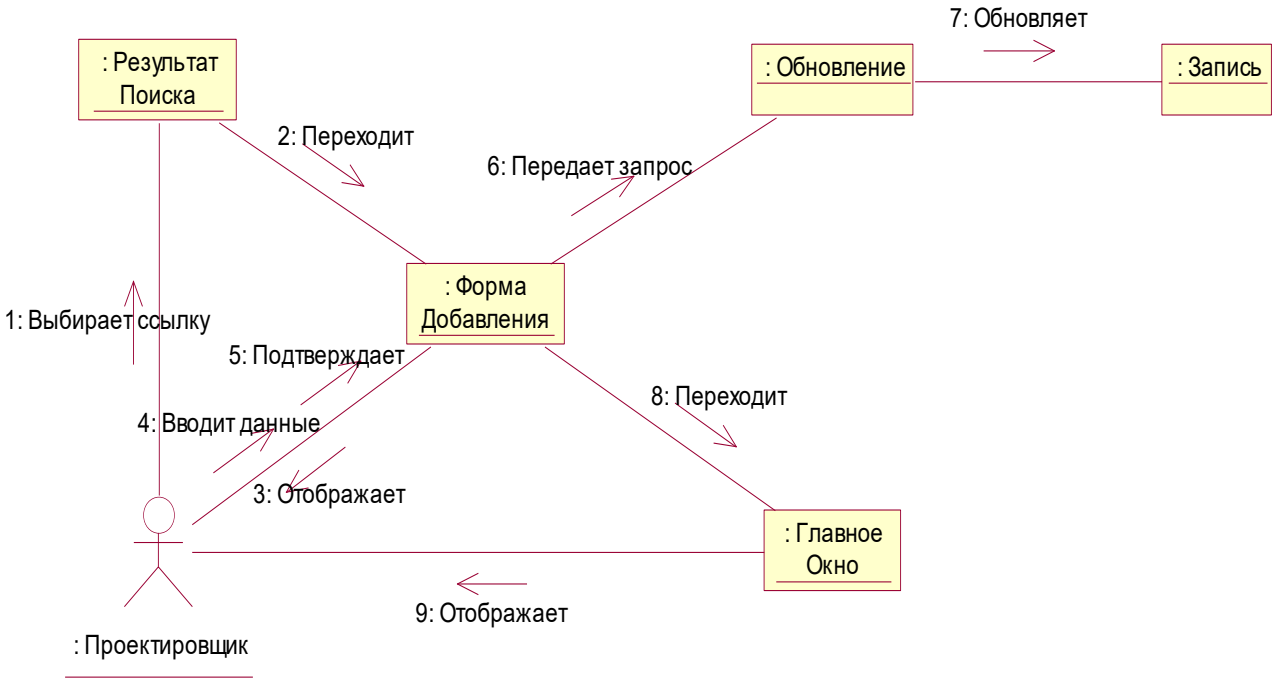


Рис. 27. Диаграмма кооперации добавления производственного задания

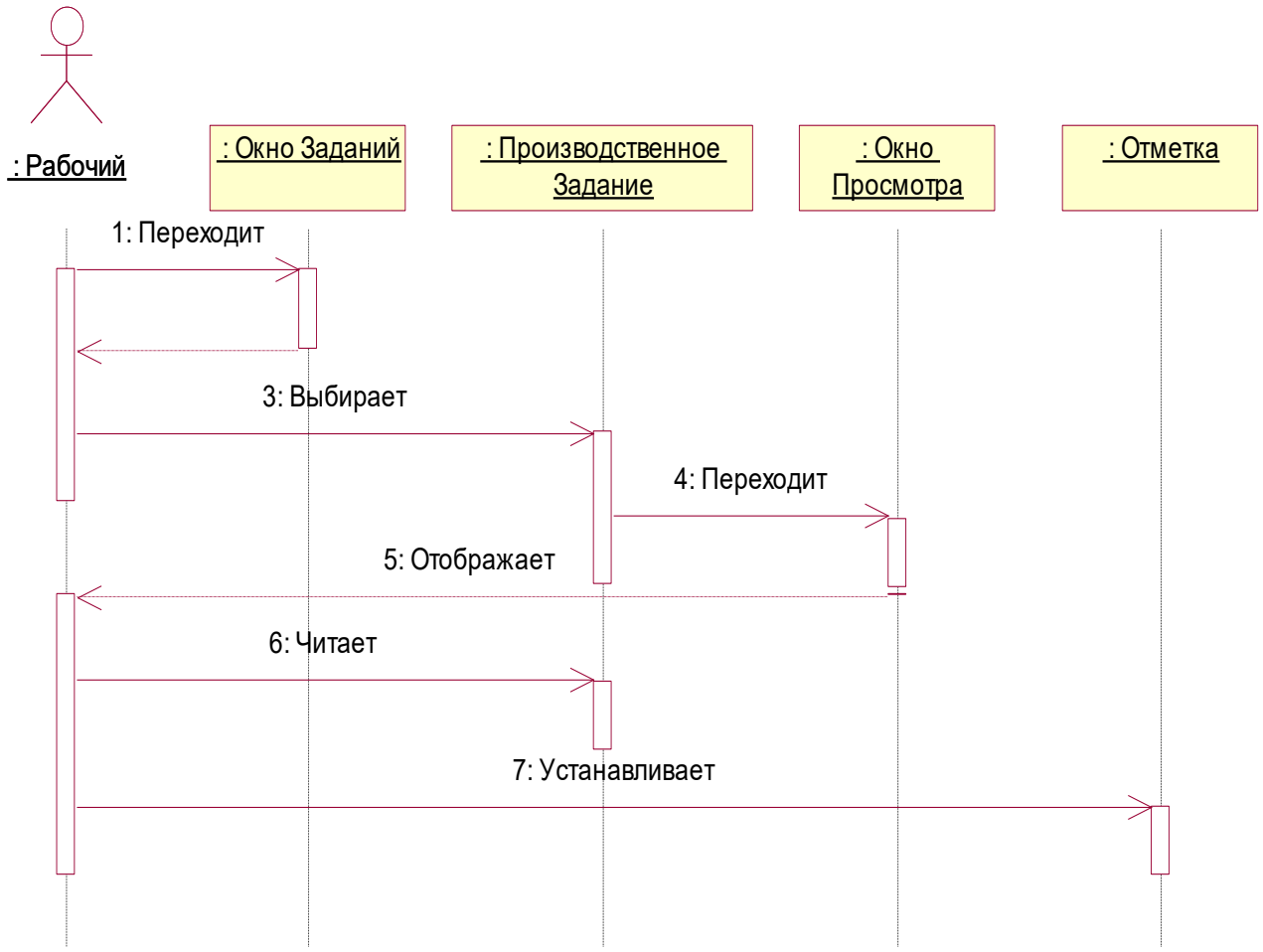


Рис. 28. Диаграмма последовательности ознакомления с производственными заданиями

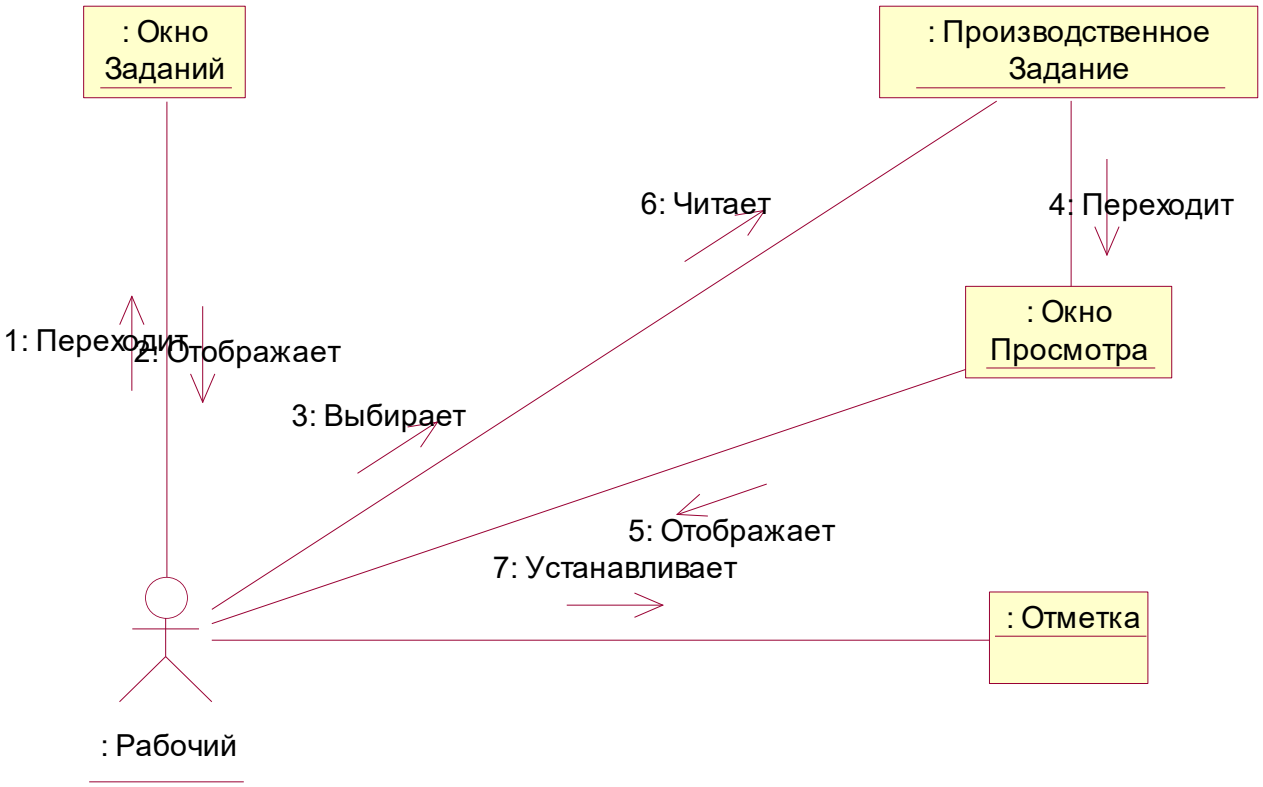


Рис. 29. Диаграмма кооперации ознакомления с производственными заданиями

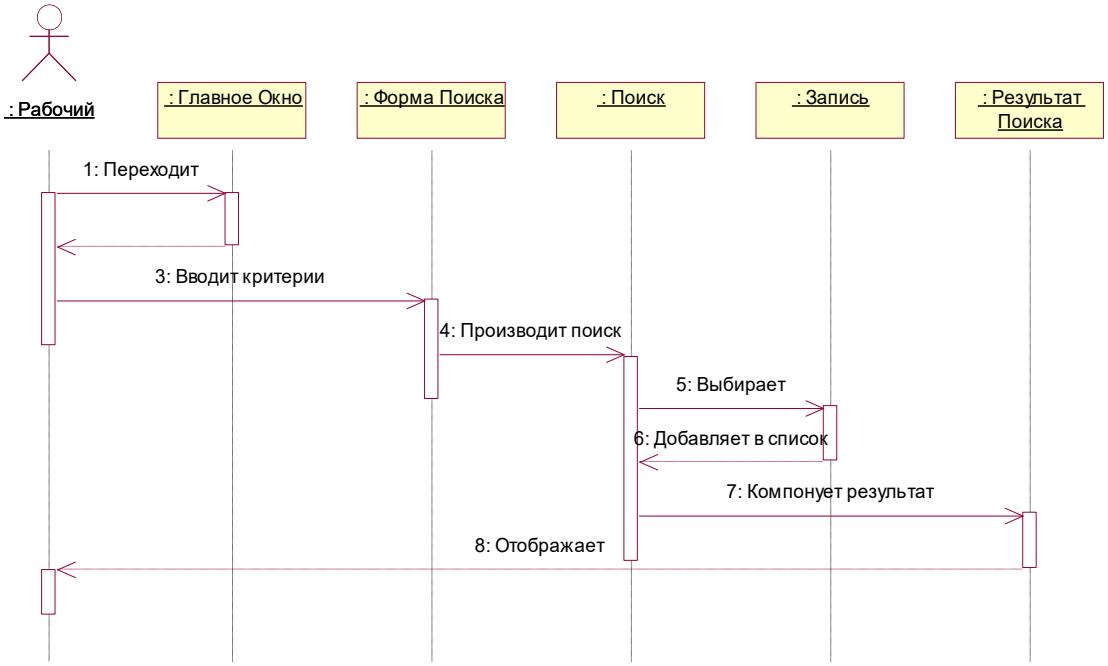


Рис. 30. Диаграмма последовательности поиска

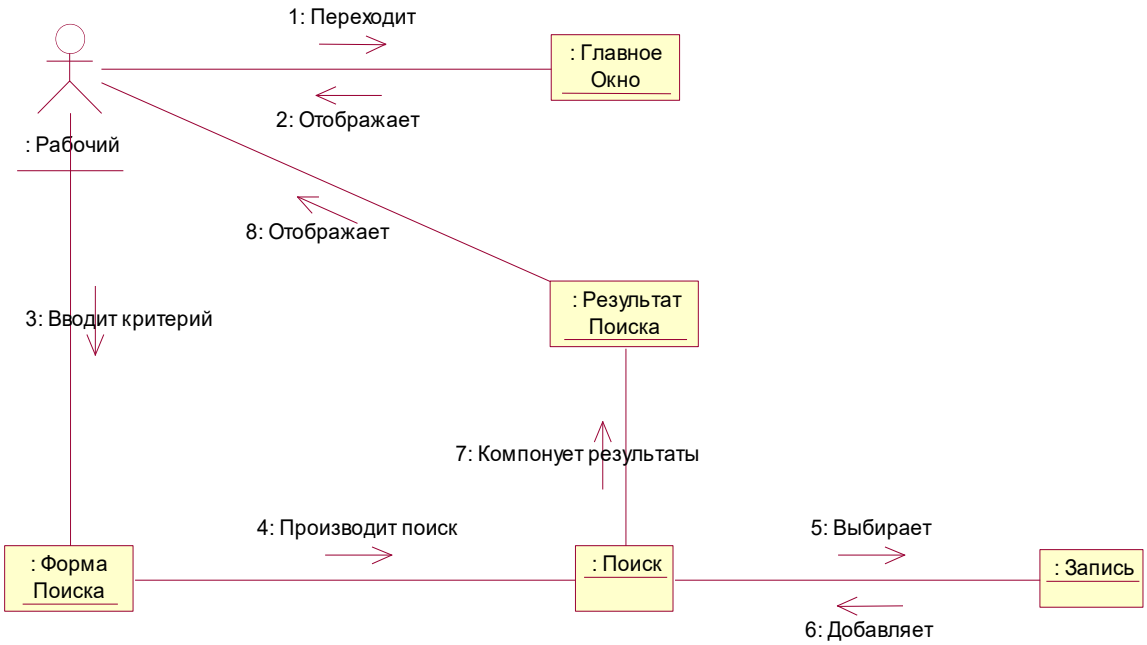


Рис. 31. Диаграмма кооперации поиска

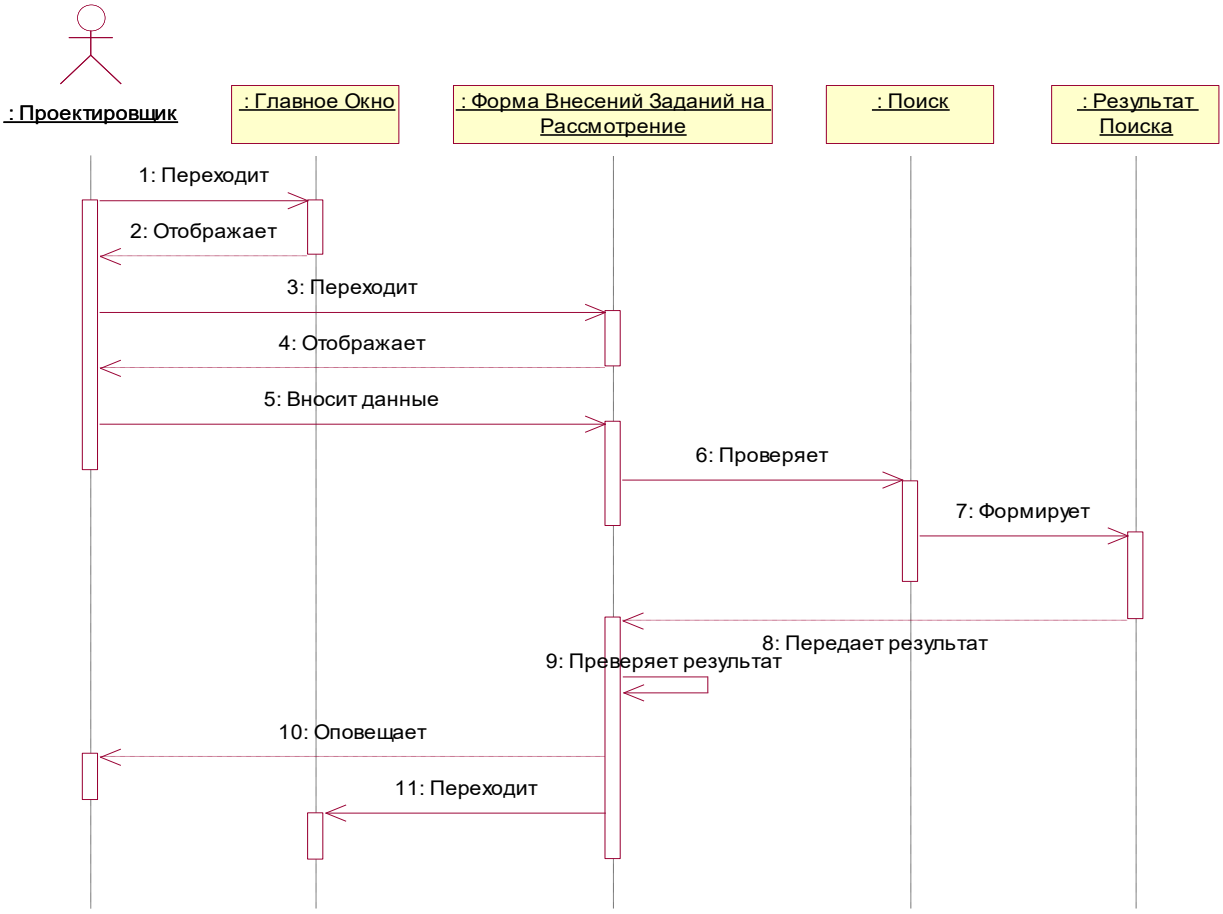


Рис. 32. Диаграмма последовательности предложения производственного задания

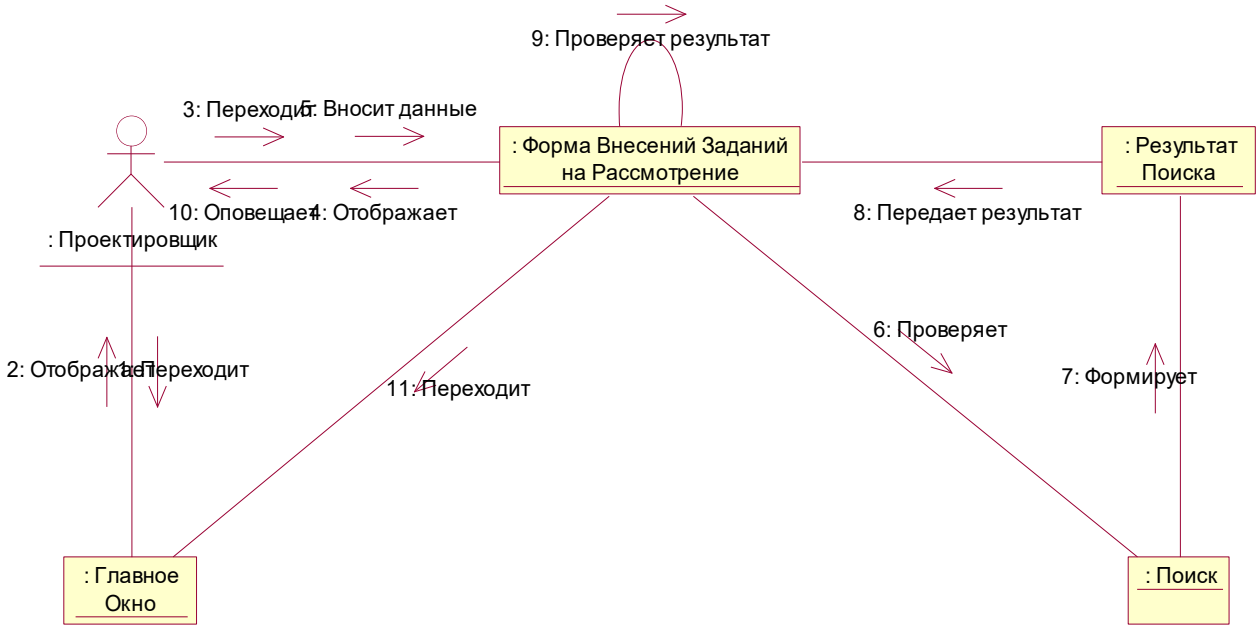


Рис. 33. Диаграмма кооперации предложения производственного задания

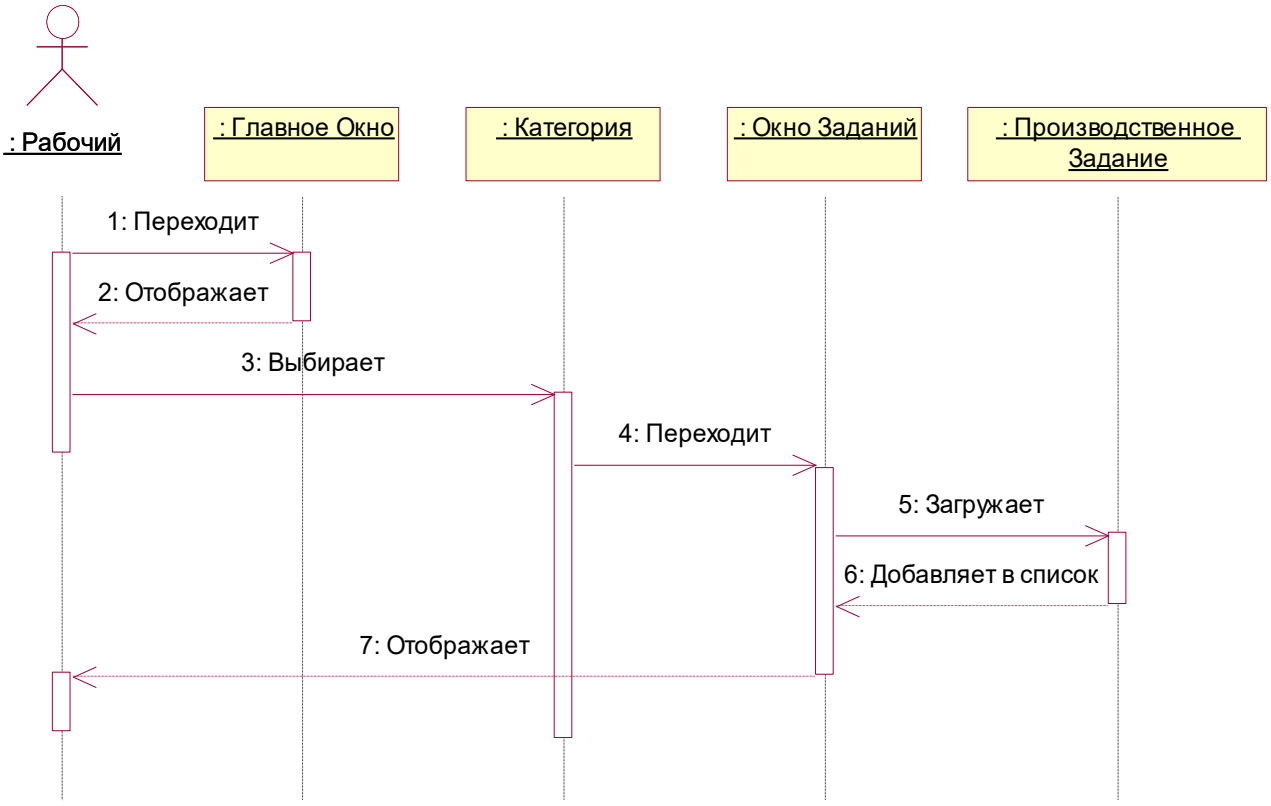


Рис. 34. Диаграмма последовательности просмотра производственного задания

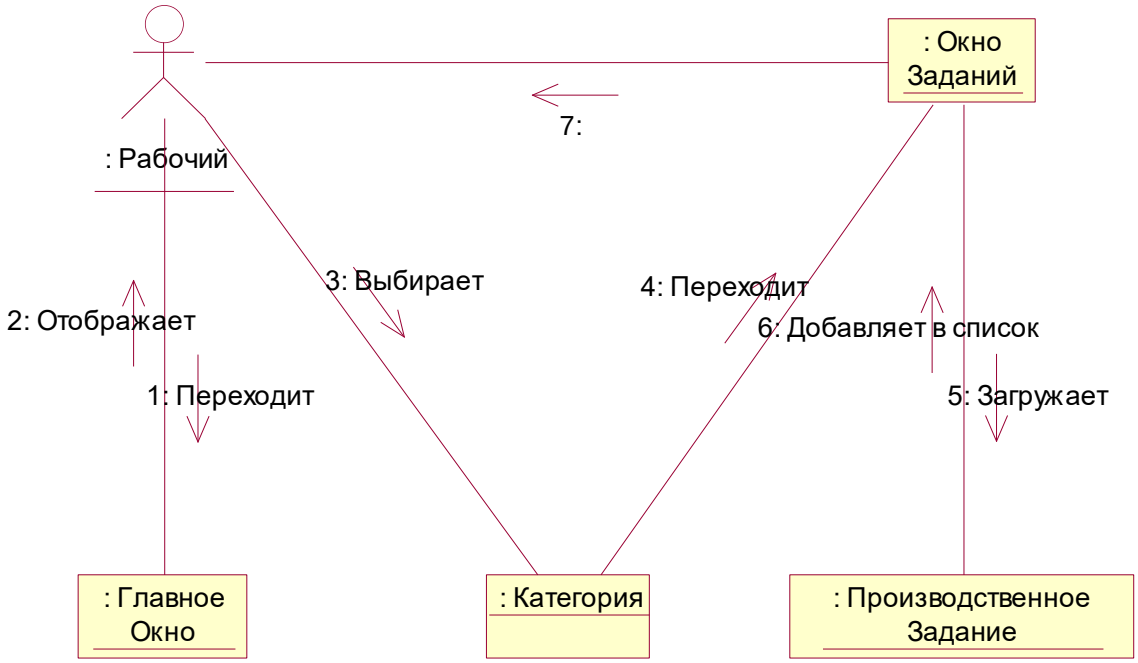


Рис. 35. Диаграмма кооперации просмотра производственного задания

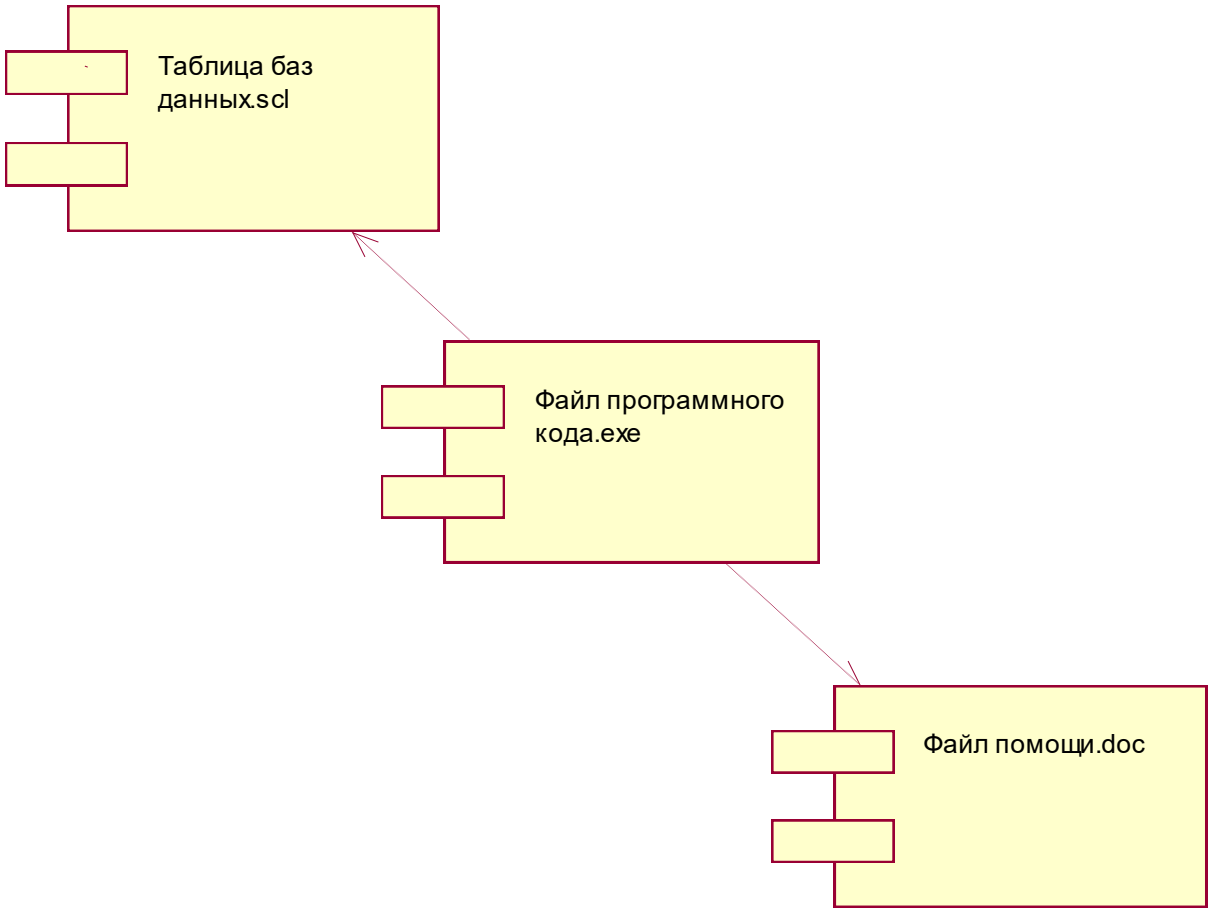


Рис. 36. Диаграмма компонентов

15. ФРАГМЕНТЫ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ МОБИЛЬНОГО АВТОСЕРВИСА

Сформируем требования к информационной системе.

Информационная система (ИС) «Мобильный автосервис» – это информационно-советующая система, предоставляющая возможность клиентам оформить заявку на ремонт автомобиля в удобное для них время и практически в любом месте, также данная информационная система упрощает работу механика с данными клиентов.

Ее основной целью является автоматизировать работу. В программу будут заноситься все данные о клиенте, например, имя, фамилия, номер телефона, и т.д., а также на основе этой информации будут проводиться статистические исследования.

Клиент сможет оформить заявку, где указывает марку и модель автомобиля, описывает произошедшую поломку и указывает свое местоположение, для получения услуг диагностики и ремонта. Механик же непосредственно может просматривать каждую заявку и выбирать необходимую на свое усмотрение.

Доступ к ИС будет предоставляться при помощи веб-интерфейса или мобильного приложения. На сервере будет отведено место для создания и хранения необходимых документов.

Все объекты, взаимодействующие с данной информационной системы можно разделить на две части – пользователи (класс «Клиент») и операторы (класс «Механик»).

На диаграмме (рис. 37) показаны все варианты использования данной ИС.

Клиенты могут зарегистрироваться в системе, а после создать заявку. В заявке клиентом указывается марка и модель своего автомобиля, поломка и местоположение. Механики могут получать заявки, отправлять свободные эвакуаторы к клиенту, проводить диагностику для выявления поломки с последующим ремонтом, находить в базе данных инструкции по ремонту конкретного автомобиля, а также имеют возможность закупать необходимые запчасти. После закупки запчастей и проведения ремонта, механик отправляет клиенту чек, составленный ИС.

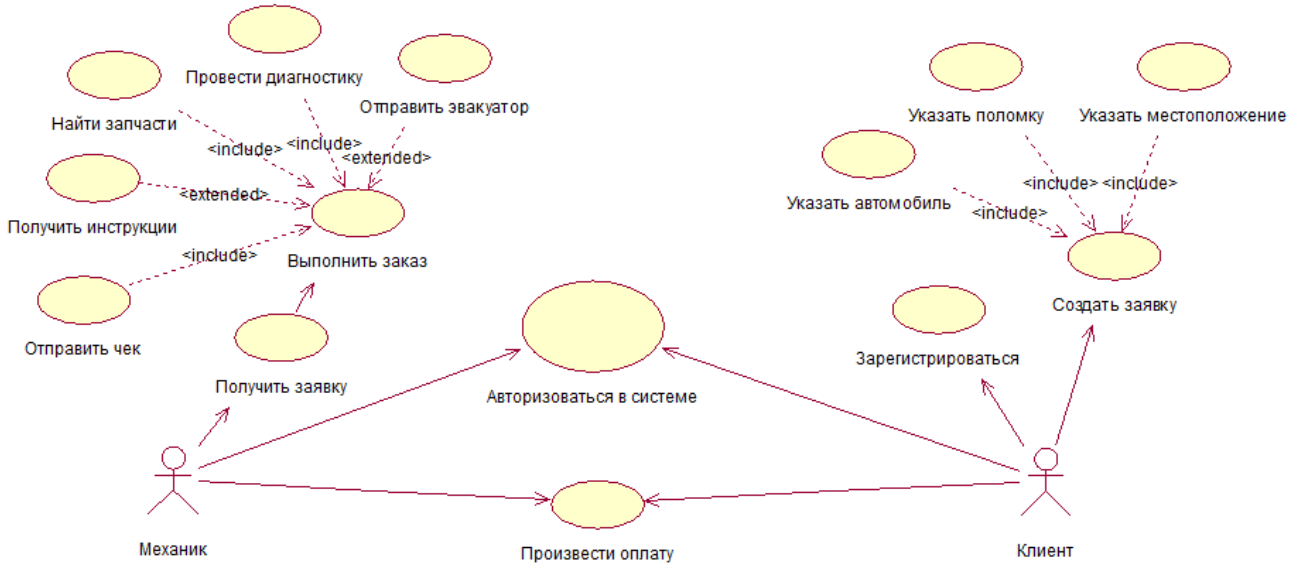


Рис. 37. Диаграмма вариантов использования

Диаграммы деятельности показывают основную логику действий в рамках того или иного варианта использования. Диаграмма деятельности – «Регистрация клиента», подразумевает под собой регистрацию клиента в ИС. Клиент вносит в форму регистрации свой логин, пароль и e-mail. Если все введенные данные соответствуют синтаксису, то регистрация завершается успешно (рис. 38).

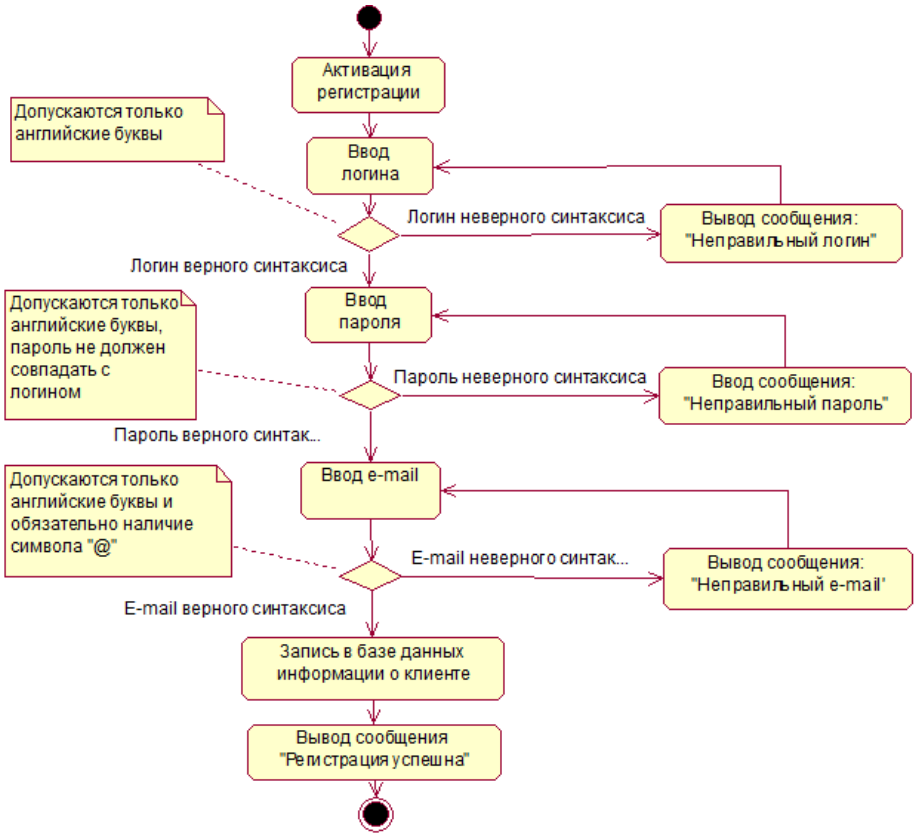


Рис. 38. Диаграмма деятельности «Регистрация клиента»

Диаграмма деятельности – «Авторизация клиента», подразумевает под собой авторизацию клиента в ИС. Клиент вносит в форму авторизации свой логин и пароль. Если все введенные данные верны, то авторизация завершается успешно. Если клиент забывает логин или пароль, он может отправить их на свой *e-mai*l для последующей авторизации.

Результат представлен на рис. 39.

Диаграмма деятельности – «Авторизация механика», подразумевает под собой авторизацию механика в ИС.

Механик вносит в форму авторизации свой код и пароль. Если все введенные данные верны, то авторизация завершается успешно и механик переходит на страницу списка внесенных заявок. Если механик забывает свой код или пароль, то он имеет возможность отправить их на свой мобильный телефон для последующей авторизации.

Диаграмма приведена на рис. 40.

Диаграмма деятельности – «Создать заявку», подразумевает под собой указание автомобиля, информации о возникшей поломке и местоположения клиента, для дальнейшей отправки данной информации в БД и составления заявки.

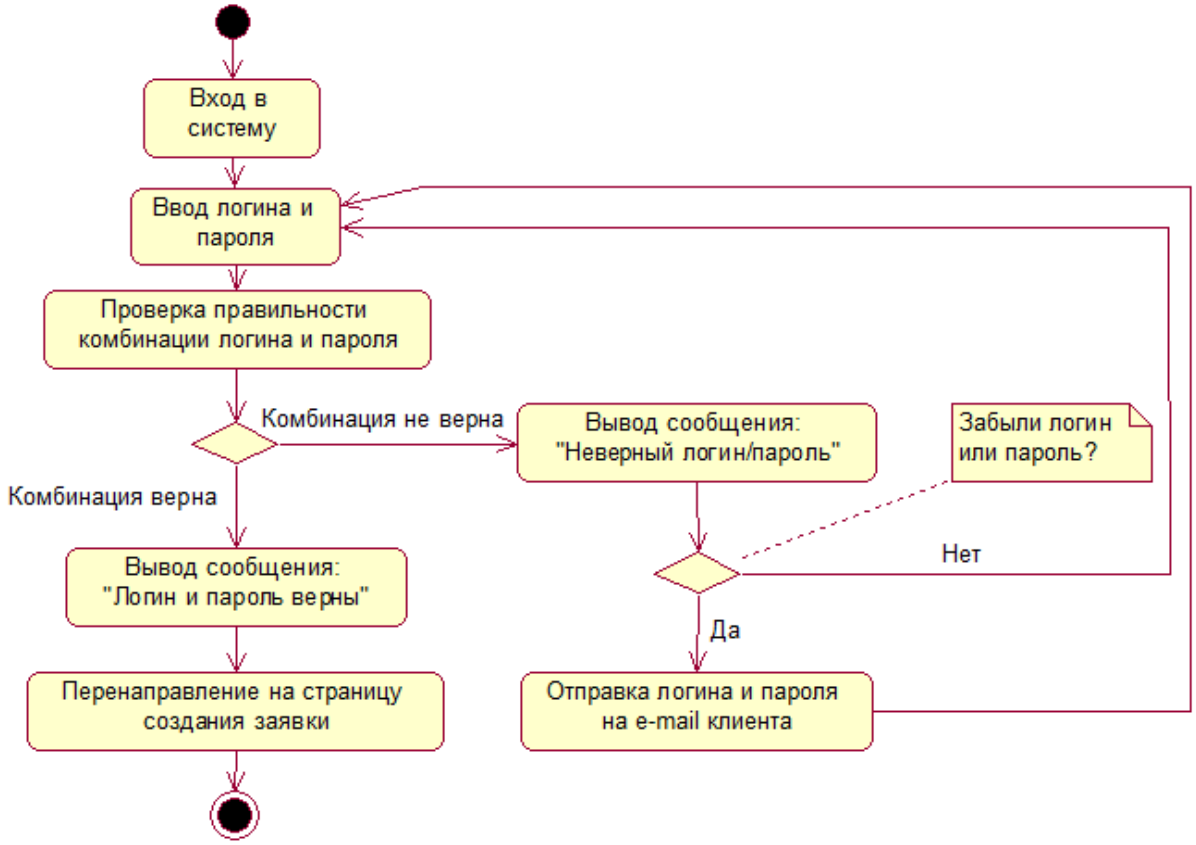


Рис. 39. Диаграмма деятельности «Авторизация клиента»

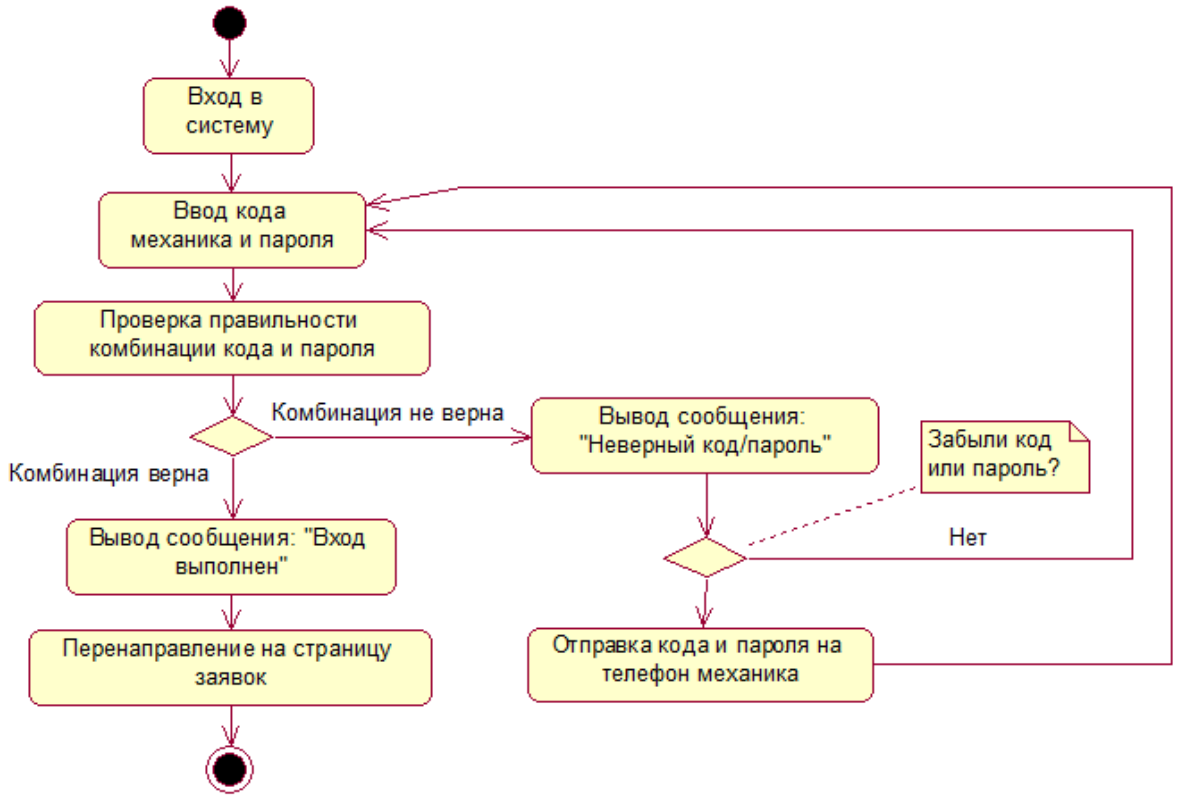


Рис. 40. Диаграмма деятельности «Авторизация механика»

Клиент выбирает из предложенного списка марку своего автомобиля, указывает модель, год выпуска и *VIN*-код. После этого клиент переходит на страницу указания поломки где он указывает поломку своего автомобиля при помощи выбора определенных пунктов из выпадающих списков, либо описав поломку вручную. Указав поломку, клиенту остается лишь указать свое местоположение после чего данные вводятся в БД, и клиент переходит на заключительную страницу.

Диаграмма деятельности «Создать заявку» приведена на рис. 41.

Диаграмма деятельности – «Провести диагностику», подразумевает под собой проведение диагностики автомобиля механиком для выявления поломки и если она присутствует, механик переходит в базу запчастей для составления списка необходимых деталей для ремонта.

Если поломка не была выявлена, статус заявки изменяется на отмененный и автомобиль возвращается клиенту (рис. 42).

Диаграмма деятельности – «Найти запчасти», подразумевает под собой проведение поиска необходимых запчастей для ремонта автомобиля клиента механиком. Данная информационная система работает с базой данных запчастей. Списки запчастей в базу данных поступают от магазинов, с которыми сотрудничает автосервис. Список запчастей в базе данных ежедневно обновляется.

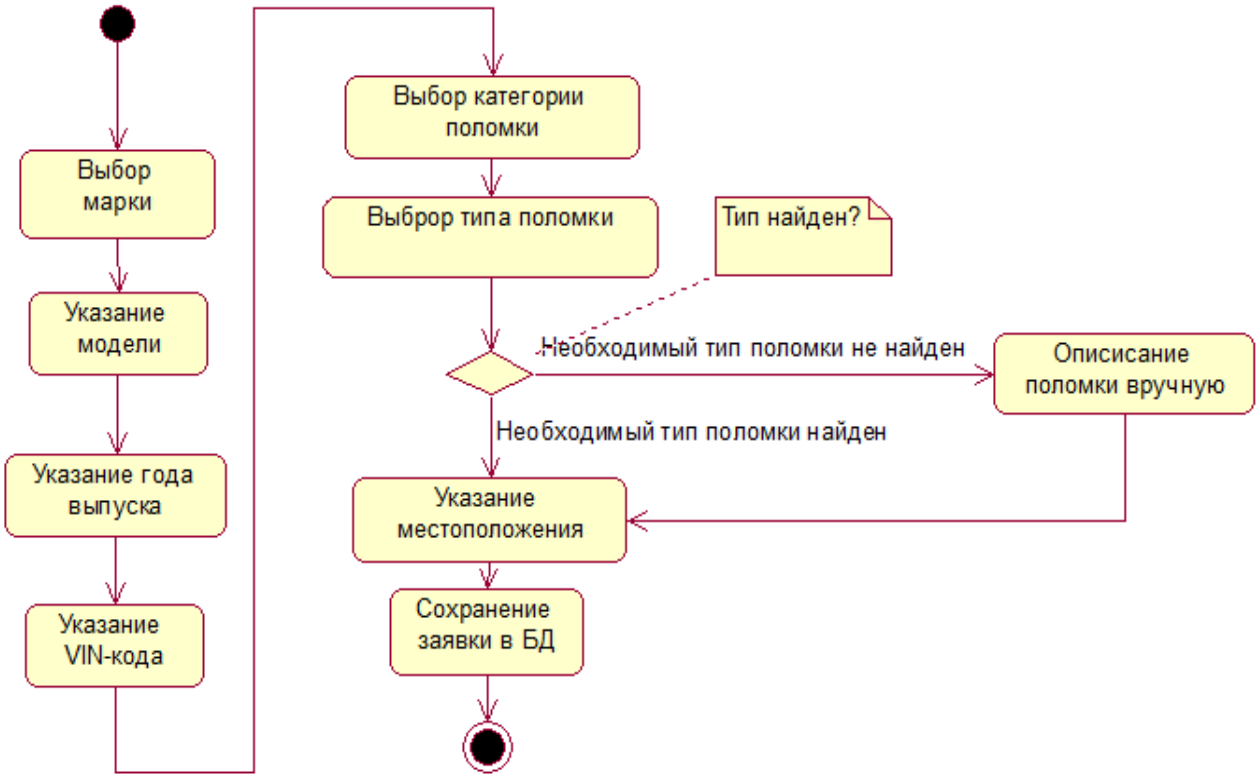


Рис. 41. Диаграмма деятельности «Создать заявку»

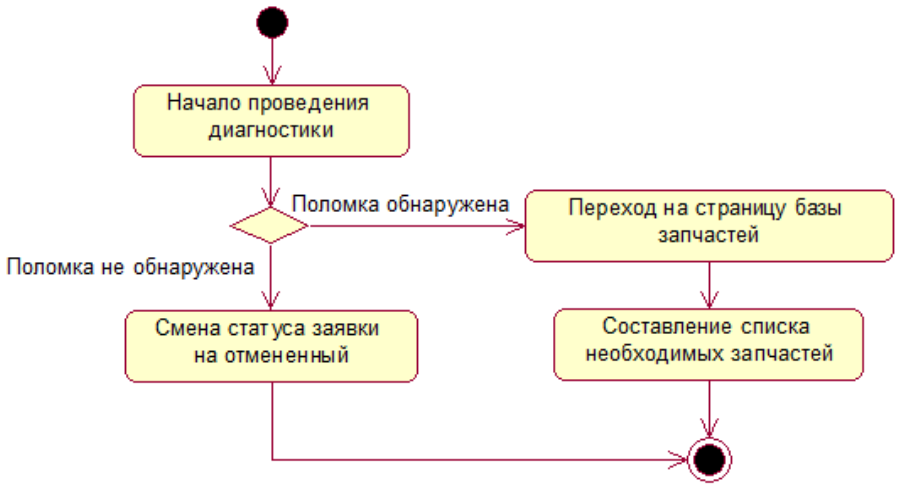


Рис. 42. Диаграмма деятельности «Провести диагностику»

Для упрощения поиска необходимых запчастей система имеет поисковой модуль, который дает возможность механику искать определенную запчасть по ее индивидуальному коду в базе данных. Также модуль имеет фильтр, благодаря которому механик может сортировать список в необходимом для себя виде.

После нахождения и покупки необходимых запчастей ИС оформляет чек, который далее отправляется механиком клиенту.

Если же необходимые запчасти не найдены, об этом механик сообщает клиенту, после чего узнает у клиента желает ли он продолжить ремонт. Если клиент не желает ожидать поставки новых запчастей, оформляется отказ от обслуживания, производится смена статуса заявки на отмененный и автомобиль возвращается клиенту. Если клиент готов ожидать поставки запчастей, обслуживание продолжается после обновления базы запчастей и нахождения необходимых деталей для ремонта.

Диаграмма деятельности «Найти запчасти» представлена на рис. 43.

Диаграмма деятельности – «Получить инструкции», подразумевает собой поиск информации по ремонту определенного автомобиля механиком в БД, для последующего ее вывода. Информация по ремонту поделена на категории и типы в которых могли бы возникнуть поломки, для упрощения поиска. Данная диаграмма обусловливает то, что данная система является информационно-советующей, и помогает механику выполнить ремонт предоставляя необходимую информацию.

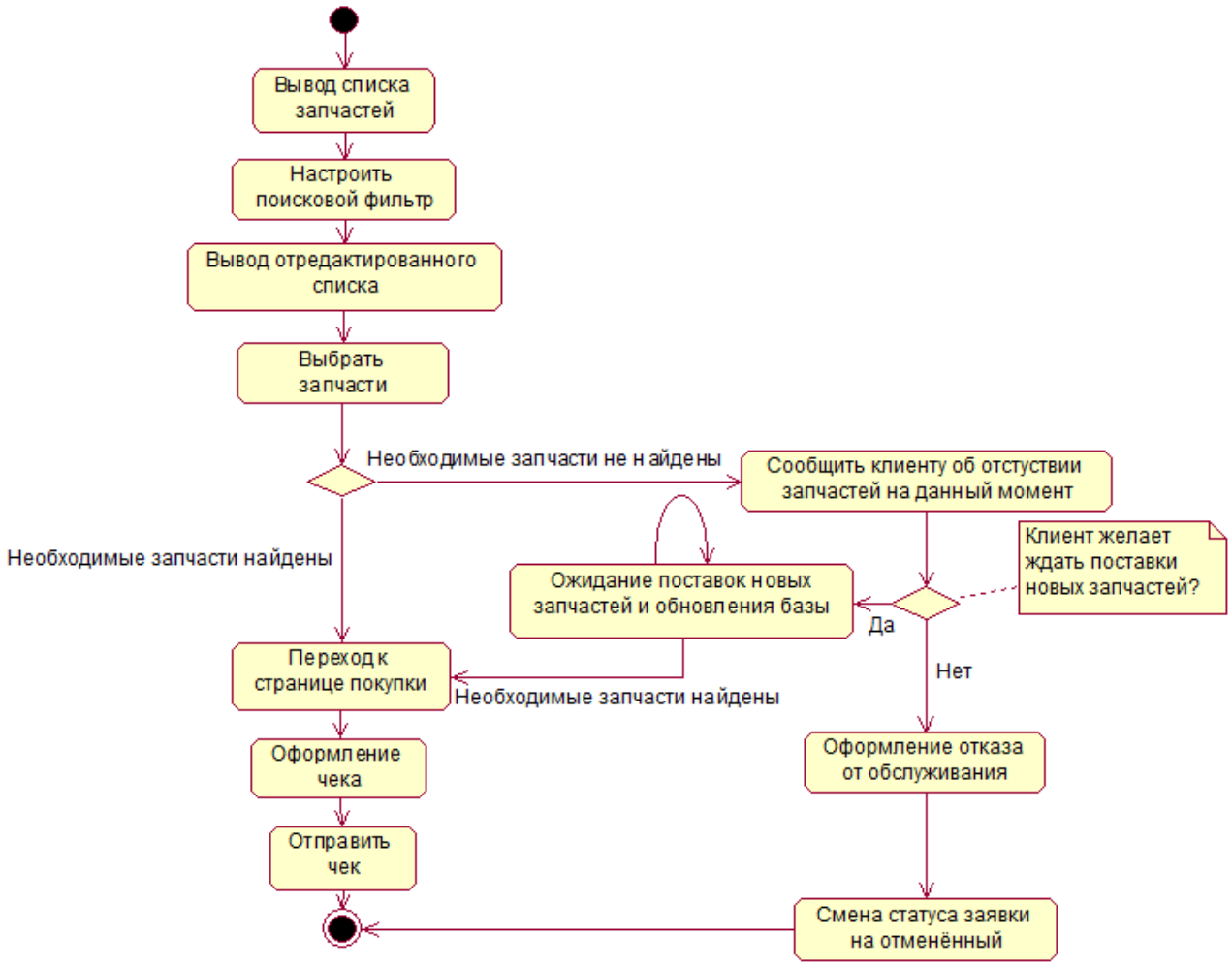


Рис. 43. Диаграмма деятельности «Найти запчасти»

Если же механик не находит информации по ремонту данного автомобиля, он в праве продолжить ремонт без инструкций при условии, что справится самостоятельно с обслуживанием автомобиля. Если же механик не справляется с работой, он имеет возможность поменяться с другим механиком. Если механик, который способен обслужить текущий автомобиль не найден, оформляется отказ от обслуживания, производится смена статуса заявки на отмененный и автомобиль возвращается клиенту.

Диаграмма деятельности «Получить инструкции» – рис. 44.

Диаграмма деятельности – «Отправить эвакуатор», подразумевает собой проверку наличия свободных эвакуаторов механиком, для последующей отправки к клиенту.

Если свободные эвакуаторы имеются, один из них сразу же отправляется на место поломки, если же нет, то необходимо ожидать возвращения одного из эвакуаторов.

Диаграмма деятельности «Отправить эвакуатор» приведена на рис. 45.

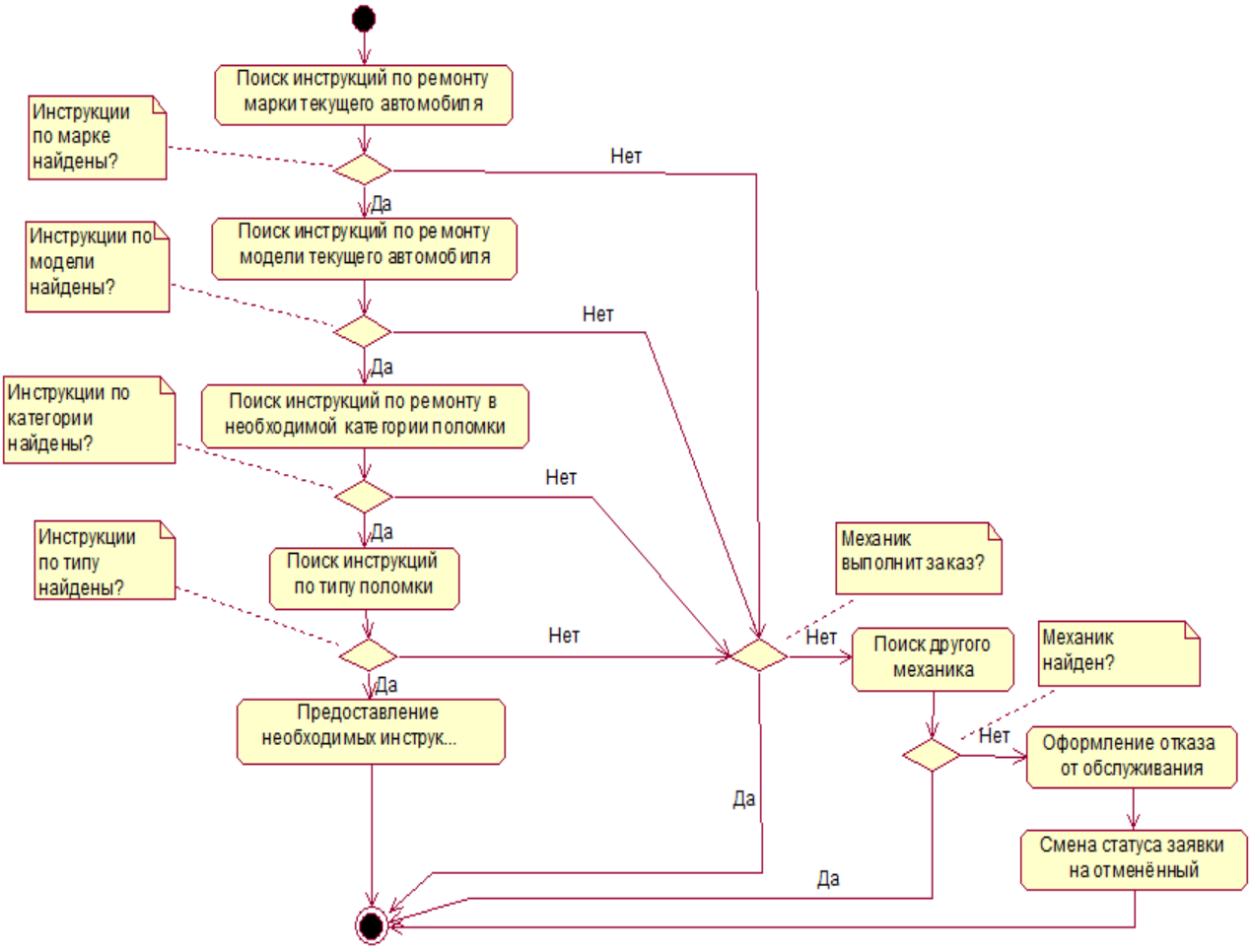


Рис. 44. Диаграмма деятельности «Получить инструкции»

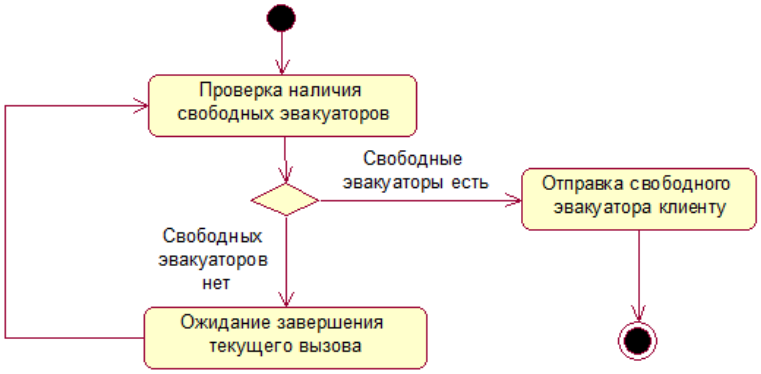


Рис. 45. Диаграмма деятельности «Отправить эвакуатор»

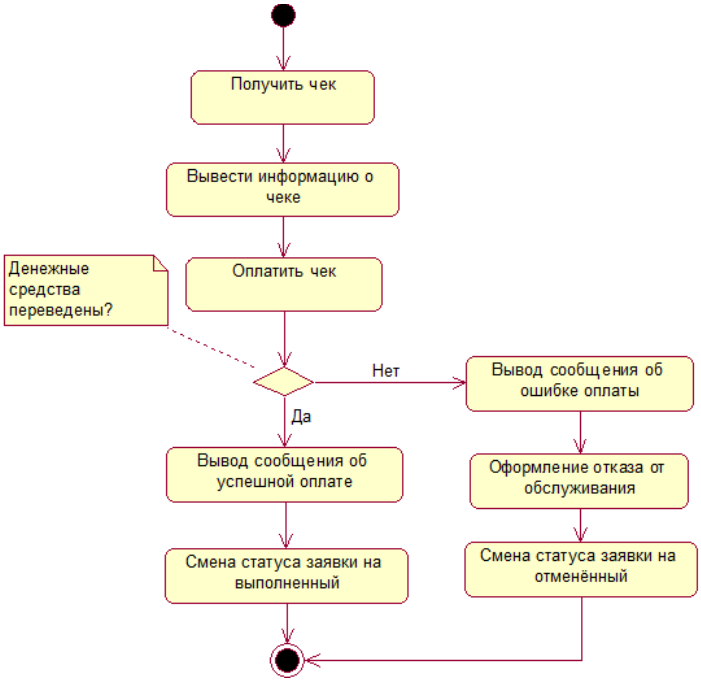


Рис. 46. Диаграмма деятельности «Оплатить»

Диаграмма деятельности – «Оплатить» (рис. 46), подразумевает под собой получение и оплату чека клиентом. Клиент получает чек, после чего переводит средства на кассу. Если средства переведены, клиенту выводится сообщение об успешной оплате, производится смена статуса заявки на выполненный и клиент забирает отремонтированный автомобиль. Если же средства не переведены, выводится сообщение об ошибке оплаты. В дальнейшем оформляется отказ от обслуживания и производится смена статуса заявки на отмененный.

Диаграмма последовательности – диаграмма, на которой показано взаимодействие объектов (обмен между ними сигналами и сообщениями), упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления.

Диаграмма последовательности «Создание заявки» (рис. 47) показывает, как происходит оформление заявки клиентом.

Клиент отправляет данные для создания заявки, после обработки данных заявка оформляется и сохраняется в БД, после чего клиент получает сообщение о составлении заявки.

Диаграмма последовательности «Поиск запчастей» показывает, как происходит поиск необходимых запчастей механиком.

Механик посылает запрос базе запчастей для получения данных содержащихся в ней, далее механик получает запрошенные данные, после чего совершает покупку (рис. 48).

Диаграмма последовательности «Оплата» показывает, как происходит оформление чека, отправка чека механиком клиенту, оплата клиентом, и смена статуса заявки на выполненный (рис. 49).

Механик запрашивает состояние чека, после чего происходит обращение к заявке для получения данных о клиенте и составляется чек. После этого механик отправляет чек клиенту и ожидает оплаты, после перевода денежных средств клиентом происходит смена статуса заявки на выполненный.

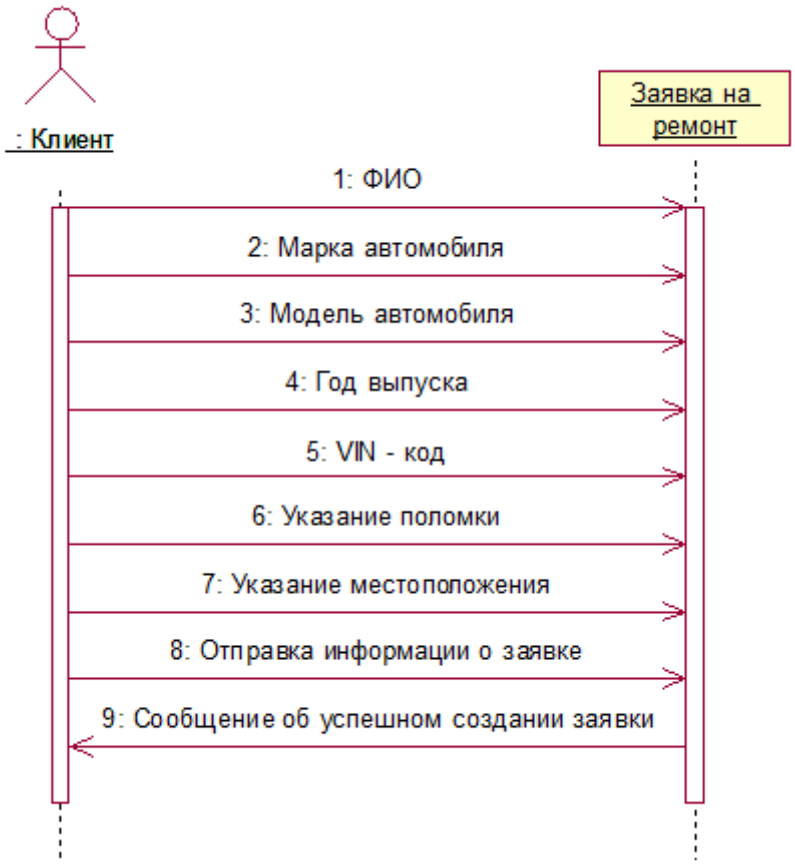


Рис. 47. Диаграмма последовательности «Создание заявки»

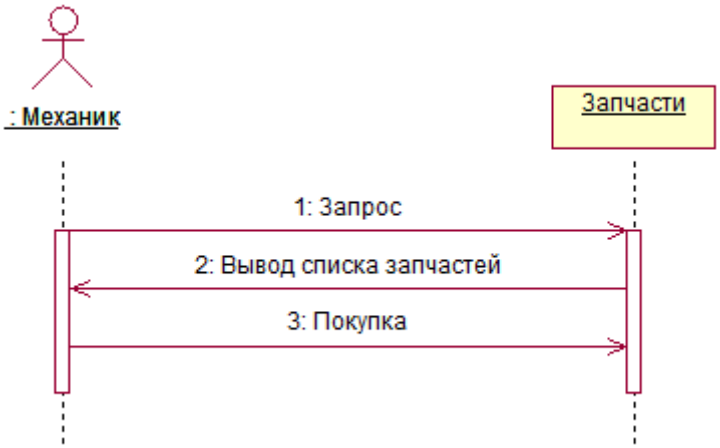


Рис. 48. Диаграмма последовательности «Поиск запчастей»

Рис. 49. Диаграмма последовательности «Оплата»

Модель состояний показывает изменение состояния объектов, важных для ИС во времени в ответ на воздействия. Модель состояния состоит из диаграмм состояния, которые важны для ИС и изменяют свое состояние во времени.

Диаграмма состояния «Регистрация клиента» подразумевает, что форма регистрации находится в состоянии ожидания до тех пор, пока не получит запрос на регистрацию, после получения запроса происходит переход в состояние выведения формы регистрации и последующего ввода данных, после введения всех данных и их успешной обработки форма регистрации возвращается в состояние ожидания (рис. 50).

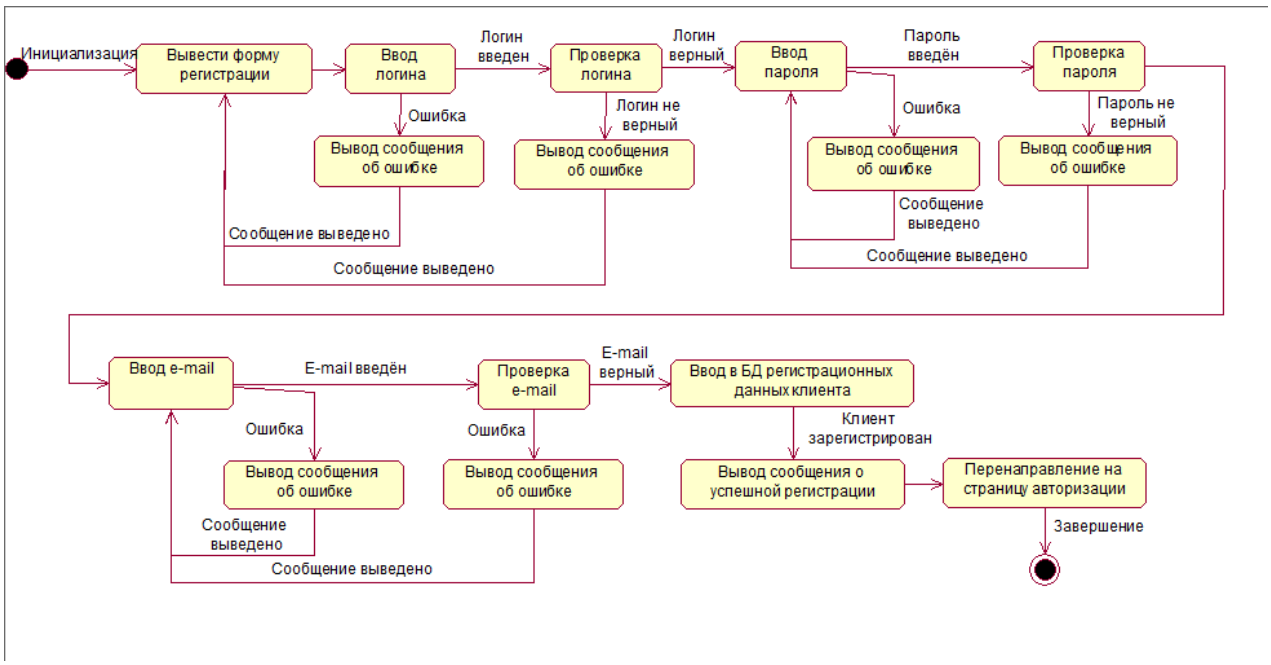


Рис. 50. Диаграмма состояний «Регистрация клиента»

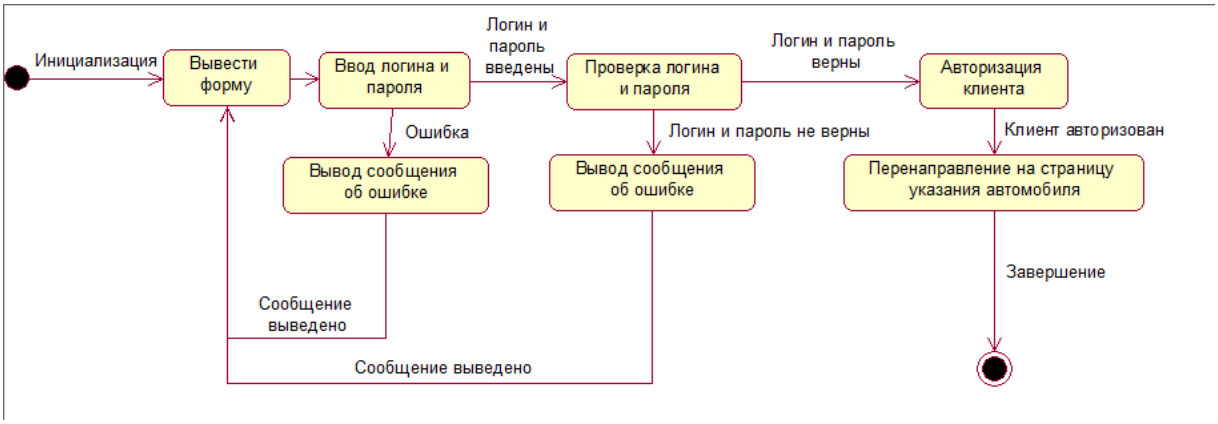


Рис. 51. Диаграмма состояний «Вход в систему клиента»

Диаграмма состояния «Вход в систему клиента» подразумевает что, форма авторизации находится в состоянии ожидания до тех пор, пока не получит запрос на авторизацию, после получения запроса происходит переход в состояние выведения формы авторизации и последующего ввода данных, после введения всех данных и их успешной проверки на правильность форма авторизации возвращается в состояние ожидания (рис. 51).

Диаграмма состояния «Вход в систему механика» подразумевает, что форма авторизации находится в состоянии ожидания до тех пор, пока не получит запрос на авторизацию, после получения запроса происходит переход в состояние выведения формы авторизации и последующего ввода данных, после введения всех данных и их успешной проверки на правильность, форма авторизации возвращается в состояние ожидания (рис. 52).

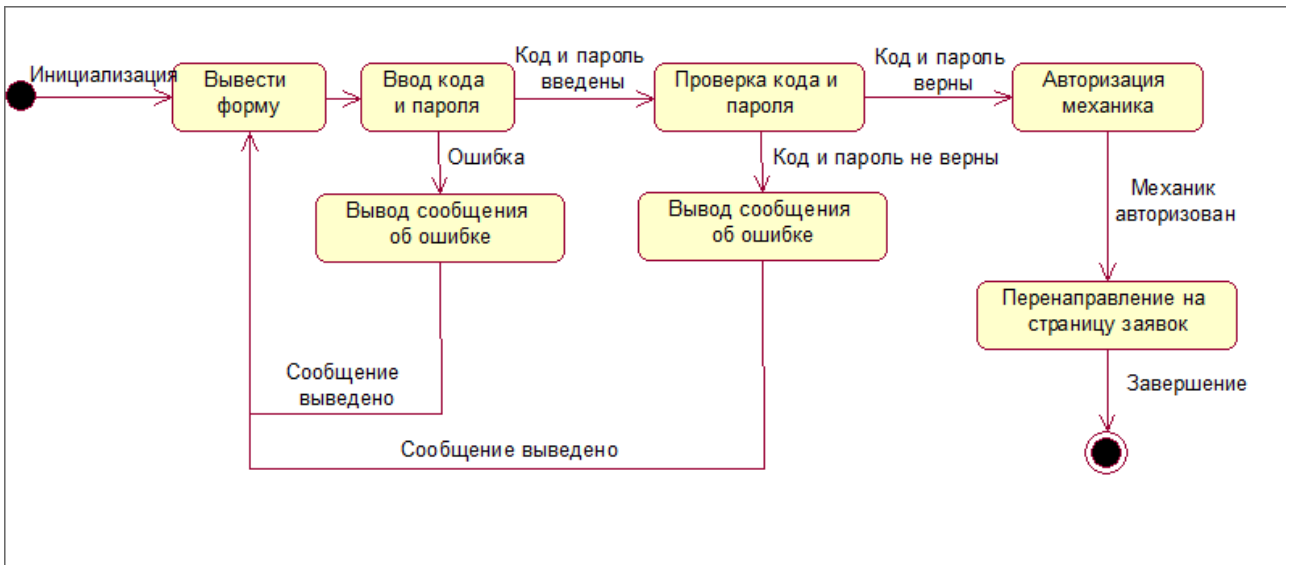


Рис. 52. Диаграмма состояний «Вход в систему механика»

Для разработки ИС созданы следующие классы:

– клиент – класс, содержащий в себе всю информацию о клиентах, такую как: код клиента, ФИО и номер телефона;

– механик – класс, содержащий информацию о механиках – код механика, ФИО и номер телефона;

– заявка на ремонт – класс, содержащий в себе информацию о заявках, а именно: код заявки, код клиента, код механика, дата создания заявки, статус;

– эвакуатор – класс, содержащий в себе информацию о наличии эвакуаторов – код эвакуатора;

– запчасти – класс, содержащий информацию о имеющихся запчастях, а именно: код списка запчастей и код запчасти;

– чек – класс, который содержит в себе информацию о чеках, а именно: код чека, общая стоимость услуг.

На рисунке 53 предоставлена диаграмма классов.

На данной диаграмме показаны все отношения между классами. Так видно, что класс «Заявка на ремонт» зависит от классов «Клиент» и «Механик». Также от класса «Механик» зависит класс «База запчастей» и «Эвакуатор».

Диаграмма компонентов – статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т.п.

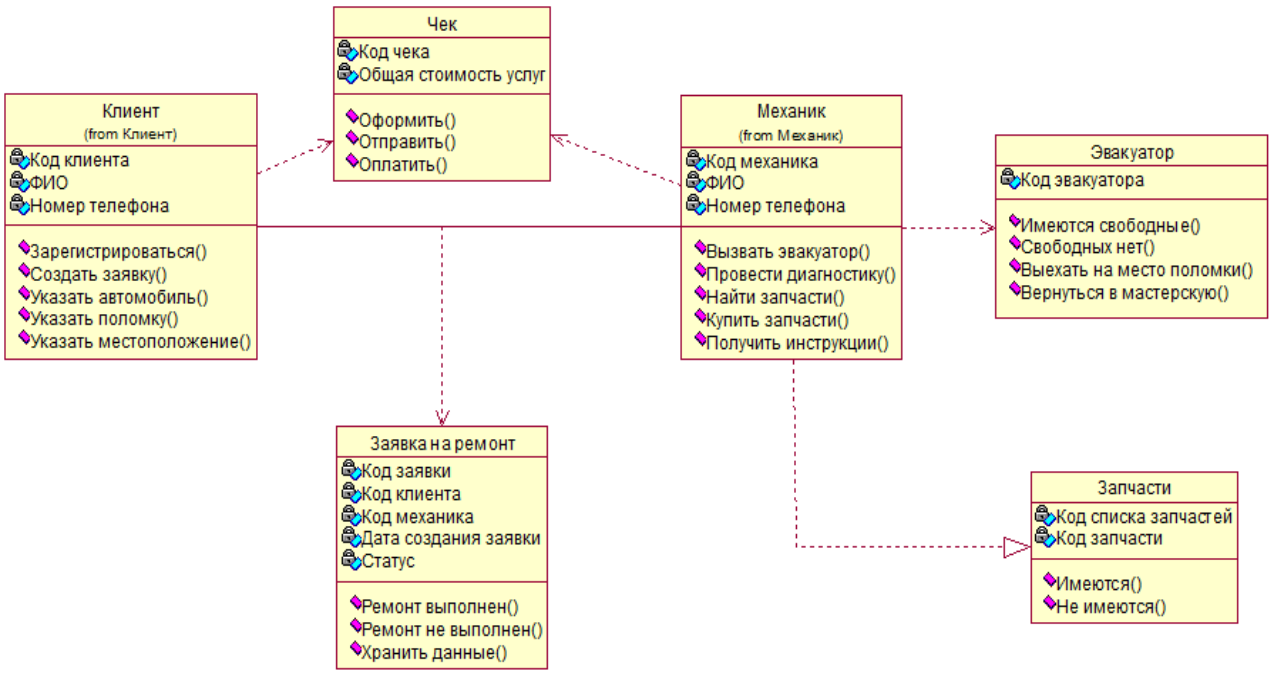


Рис. 53. Диаграмма классов

Основным компонентом будет являться приложение – «*main.exe*». Именно на нем будет происходить большинство действий. Также будут несколько компонентов. Первый из них – «*registration.dll*», он содержит в себе форму регистрации и будет ее выводить каждому незарегистрированному клиенту для последующей регистрации, второй компонент «*zayavka.dll*», в свою очередь, отвечает за хранение составленных заявок клиентов. Третий компонент – «*parts.dll*» хранит и выводит механику базу запчастей. Все компоненты привязаны к приложению «main.exe» (рис. 54).

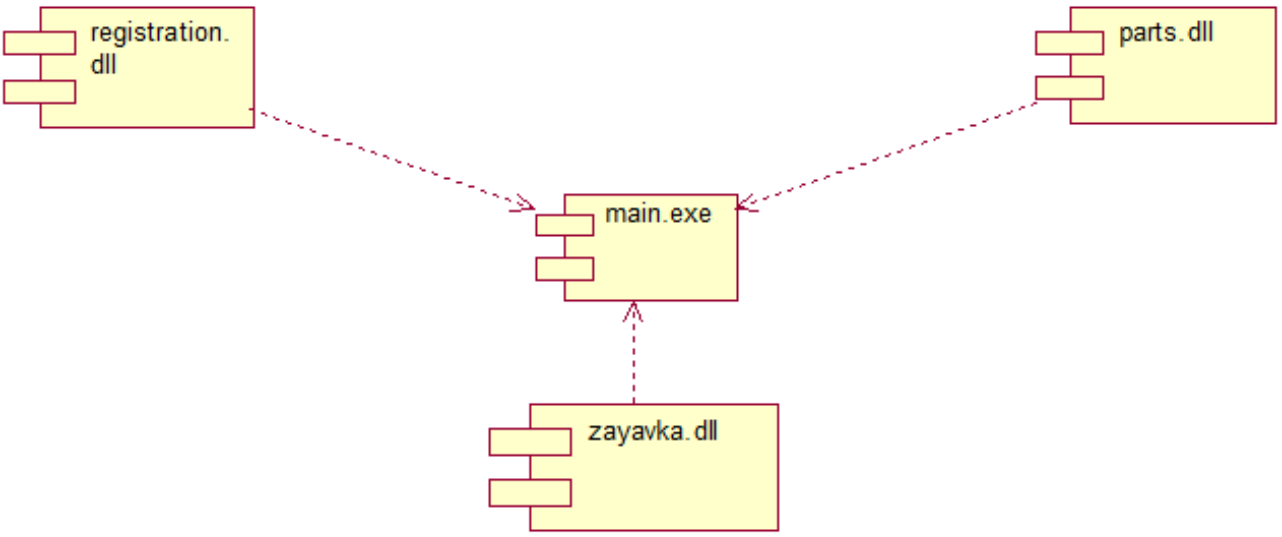


Рис. 54. Диаграмма компонентов

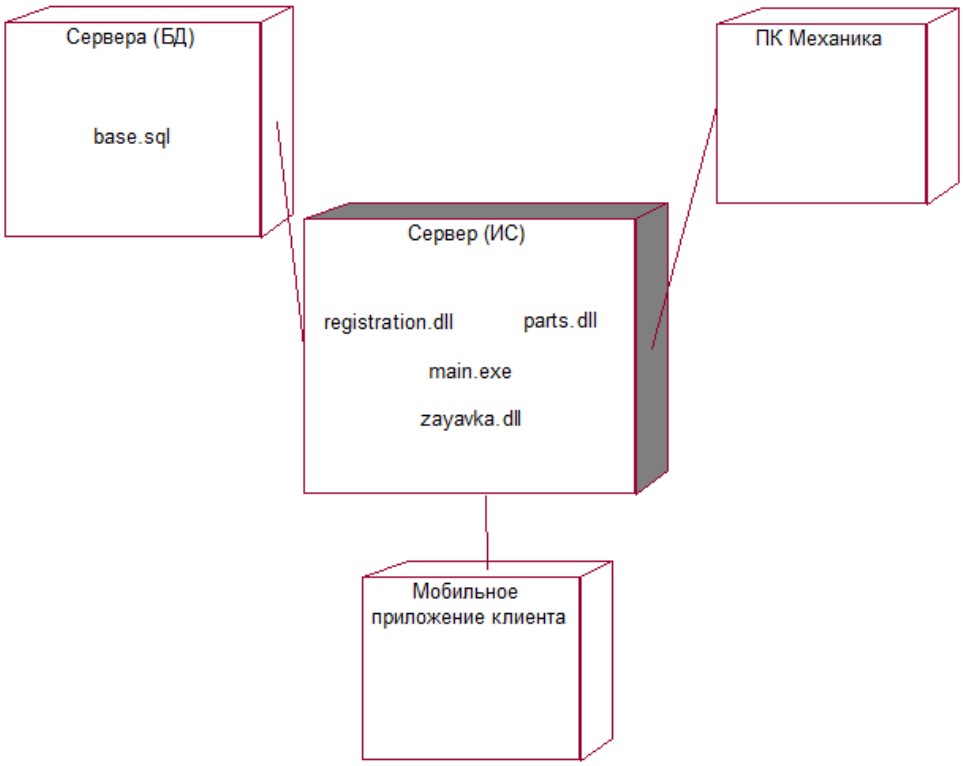


Рис. 55. Диаграмма развертывания

Диаграмма развертывания моделирует физическое развертывание артефактов на узлах.

В ИС будет четыре элемента – «Мобильное приложение клиента», «ПК механика», «Сервер (ИС)» и «Сервера (БД)».

В элементе «Сервера (БД)» содержится, в свою очередь, файл базы данных «*base.sql*». В элементе «Сервер (ИС)» содержатся компоненты «*main.exe*», «*registration.dll*», «*zayavka.dll*» и «*parts.dll*» (рис. 55).

ЗАКЛЮЧЕНИЕ

В пособии достаточно подробно и системно изложен теоретический материал с примерами реализации, что позволит студентам получить навыки проектирования информационных систем.

В первой главе рассмотрены термины и определения применительно к информационной системе, ее признаки классификации и архитектура.

Во второй главе рассмотрены особенности создания программных изделий.

В третьей главе – основные стандарты и руководящие документы, регламентирующие проектирование информационных систем.

Четвертая, пятая и шестая главы посвящены вопросам канонического проектирования, недостаточно освещенным в учебной литературе: стадии и этапы создания автоматизированных систем по ГОСТ 34.601–90, по ГОСТ 24.601–86, этапы и содержание работ по ГОСТ 19.102–77.

Седьмая глава посвящена жизненному циклу программного обеспечения и средствам его поддержки. Рассмотрены, в частности, общая схема самого жизненного цикла, его модели.

Восьмая глава содержит сведения о средствах CASE-технологий.

Девятая глава – содержание этапов проектирования информационной системы.

Десятая и одиннадцатая главы включают основы объектно-ориентированного подхода к анализу и проектированию, моделированию бизнес-процессов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Об информации, информационных технологиях и о защите информации : федер. закон от 27.07.2006 № 149-ФЗ.
2. ГОСТ Р 51189–98 Средства программные систем вооружения. Порядок разработки.
3. ГОСТ РВ 15.201–2003. Тактико-техническое (техническое) задание на выполнение опытно-конструкторских работ, ГОСТ РВ 15.203–2001. Порядок выполнения опытно-конструкторских работ по созданию изделий и их составных частей.
4. ГОСТ РВ 15.210–2001. Испытания опытных образцов и опытных ремонтных образцов изделий. Основные положения.
5. ГОСТ РВ 15.211–2002. Порядок разработки программ и методик испытаний опытных образцов изделий. Основные положения.
6. ГОСТ 2.119–73. Разработка эскизного проекта.
7. ГОСТ 2.120–73. Разработка технического проекта.
8. ГОСТ 34.601–90. Автоматизированные системы. Стадии создания.
9. ГОСТ 34.602–89. Техническое задание на создание автоматизированной системы.
10. ГОСТ 34.201–89. Виды, комплектность и обозначение документов при создании автоматизированных систем.
11. ГОСТ РД 50–34.698–90. Автоматизированные системы. Требования к содержанию документов.
12. ГОСТ Р 50–34.126–92. Правила проведения работ при создании информационных систем.
13. ГОСТ РВ 51987–2002. Термины и определения. Основные характеристики качества функционирования ИС, соотнесенные с потенциальными угрозами информации.
14. ГОСТ 34.003–90. Автоматизированные системы. Термины и определения.
15. ГОСТ 24.602–86. Автоматизированные системы стадии и этапы создания АСУ.
16. ГОСТ 19.102–77. Единая система программной документации. Стадии разработки
17. ГОСТ Р ИСО/МЭК 12207–2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств
18. ГОСТ Р ИСО/МЭК ТО 10000-1–99. Информационная технология. Основы и таксономия международных функциональных стандартов. Часть 1. Общие положения и основы документирования
19. Р 50.1.053–2005. Рекомендации по стандартизации. Информационные технологии. Основные термины и определения в области технической защиты информации.

20. ГОСТ Р 53622–2009. Информационные технологии. Информационно-вычислительные системы. Стадии и этапы жизненного цикла, виды и комплектность документов.

21. ISO/IEC 2382-1:1993. Информационные технологии. Словарь. Часть 1. Основные термины.

22. ГОСТ 28195–1989. Оценка качества программных средств. Общие положения и ГОСТ Р ИСО/МЭК 9126-1993 Информационная технология. Оценка программной продукции. Характеристики качества и руководство по их применению.

23. Грекул, В. И. Проектирование информационных систем. Курс лекций : учебное пособие / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. – М. : Интернет-университет ИТ, 2005.

24. Боггс, У. UML и Rational Rose / У. Боггс, М. Богс. – Изд-во «ЛОРИ», 2000. – 580 с.

25. Фаулер, М. UML в кратком изложении. Применение стандартного языка объектного моделирования / М. Фаулер, К. Скотт ; пер. с англ. – М. : Мир, 1999. – 191 с.

26. Буч, Г. Язык UML. Руководство пользователя : рук. / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2008. – 496 с.

27. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани. – М. : ДМК Пресс, 2009. – 176 с.

28. Вендров, А. М. Проектирование программного обеспечения экономических информационных систем : учебник / А. М. Вендров. – М. : Финансы и статистика, 2000.

29. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч ; пер. с англ. – 2-е изд. – М. : Изд-во Бином ; СПб. : Невский диалект, 1999.

30. Ларман, Крэг. Применение UML и шаблонов проектирования / Ларман Крэг ; пер. с англ. : учебное пособие – М. : Вильямс, 2001.

31. Буч, Г. Язык UML. Руководство пользователя / Буч Г., Рамбо Дж., Джекобсон А. ; пер. с англ. – М. : ДМК, 2000.

32. Фаулер, М. UML в кратком изложении. Применение стандартного языка объектного моделирования / М. Фаулер, К. Скотт ; пер. с англ. – М. : Мир, 1999.

33. Боггс, У. UML и Rational Rose / У. Боггс, М. Боггс ; пер. с англ. – М. : ЛОРИ, 2000.

34. Вендров, А. М. CASE-технологии. Современные методы и средства проектирования информационных систем. – М. : Финансы и статистика, 1998.

35. Шлеер, С. Объектно-ориентированный анализ: моделирование мира в состояниях / С. Шлеер, С. Меллор ; пер. с англ. – Киев : Диалектика, 1993.

36. Ломако, Е. И. Математические и понятийные средства системантики / Е. И. Ломако. – М. : Системная Энциклопедия, 2008.

ОГЛАВЛЕНИЕ

Учебное электронное издание

ИВАНОВСКИЙ Михаил Андреевич
ГЛАЗКОВА Инга Александровна

МЕТОДЫ И СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

Учебное пособие

Редактирование Е. С. Мордасовой
Графический и мультимедийный дизайнер Т. Ю. Зотова
Обложка, упаковка, тиражирование Е. С. Мордасовой

ISBN 978-5-8265-2787-0

Подписано к использованию 04.07.2024.
Тираж 50 шт. Заказ № 81

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14
Тел./факс (4752) 63-81-08.
E-mail: izdatelstvo@tstu.ru