Московский государственный технический университет
имени Н.Э. Баумана

М.В. Виноградова, В.И. Белоусова

Унифицированный процесс разработки программного обеспечения

Учебное пособие

Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2015

УДК 681.3.06(075.8)
ББК 32.973-018.2
В49

Издание доступно в электронном виде на портале *ebooks.bmstu.ru*
по адресу: http://ebooks.bmstu.ru/catalog/193/book1303.html

Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Рекомендовано Редакционно-издательским советом
МГТУ им. Н.Э. Баумана в качестве учебного пособия

Рецензент Т.Н. Захарова

Виноградова, М. В.

В49 Унифицированный процесс разработки программного обеспечения : учебное пособие / М. В. Виноградова, В. И. Белоусова. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2015. — 80, [2] с. : ил.

ISBN 978-5-7038-4265-2

Приведены основные сведения об унифицированном процессе разработки программного обеспечения (RUP — Rational Unified Process). Рассмотрены этапы разработки и их рабочие процессы. Детально изложены принципы построения моделей программного проекта от определения требований до тестирования.

Для студентов МГТУ им. Н.Э. Баумана, изучающих дисциплину «Технологии разработки программного обеспечения».

УДК 681.3.06(075.8)
ББК 32.973-018.2

ISBN 978-5-7038-4265-2

Предисловие

В данном учебном пособии рассмотрены базовые принципы унифицированного процесса разработки программного обеспечения (RUP — Rational Unified Process), его этапы, рабочие процессы и создаваемые артефакты. Детально описаны рабочие процессы, выполняемые на каждой итерации. Приведены правила построения моделей программного проекта на всех этапах его разработки, начиная от сбора и формализации требований, анализа и проектирования, до реализации и тестирования.

Учебное пособие предназначено для студентов, изучающих дисциплину «Технологии разработки программного обеспечения», а также другие дисциплины, посвященные вопросам программной инженерии.

В результате освоения изложенного в учебном пособии материала студенты приобретут следующие навыки:

• применения современных программных средств для разработки проектно-конструкторской и технологической документации, отчетов и презентаций;

• реализации и оценки архитектурных решений и проектов как компонентов, так и системы в целом на соответствие требованиям технического задания и стандартам;

• проведения совещаний, участия в подготовке приемки, приемо-сдаточных испытаний, совместных анализов и аудиторских проверок вместе с заказчиком в соответствии с договором и проектными планами;

• анализа требований к проекту, подготовки договора и технического задания на разработку автоматизированной информационной системы;

• разработки архитектуры системы с указанием технических и программных средств, ручных операций и связей между ними;

• проектирования технических и программных компонентов с учетом требований, разработанных интерфейсов и с использованием современных инструментальных средств;

• постановки новых задач анализа и синтеза сложных проектных решений;

• распределения заданий и ресурсов по процессам жизненного цикла аппаратно-программных систем.

Для изучения учебного пособия необходимо наличие у студентов базовых знаний в областях объектно-ориентированного программирования, баз данных и языка UML. Все диаграммы, приведенные в учебном пособии, составлены в нотации UML.

Учебное пособие состоит из шести глав. В главе 1 приведены общие сведения об унифицированном процессе, его особенностях, этапах и рабочих процессах. В главах 2–6 рассмотрены рабочие процессы RUP: сбор требований, анализ, проектирование, реализация и тестирование.

Введение

В настоящее время основными технологиями как промышленной, так и индивидуальной разработки программ является объектно-ориентированный подход с применением пакетов визуальной разработки. Это такие среды, как Microsoft Visual Studio, Java SDK, Qt SDK и многие другие. Данные средства разработки сделали процесс написания исходного кода программ тривиальным и сместили основной акцент при создании программного обеспечения (ПО) на этапы анализа и проектирования.

По этой причине главными задачами разработчиков программных систем являются формализация исходных требований, последующий их анализ и проектирование, результаты которого транслируются в исходные коды программ. Для автоматизации перечисленных задач существует множество программных средств, называемых CASE (Computer-Aided Software Engineering — средства автоматической разработки программ), которые позволяют строить и преобразовывать модели программного проекта с разных точек зрения. Однако для эффективного использования богатых возможностей CASE-средств необходимо знать методику их применения.

Унифицированный процесс разработки ПО RUP — это методология создания программных систем, являющаяся результатом объединения различных подходов к построению программ.

В RUP используется визуальное проектирование с применением языка UML и автоматическая генерация кода на основе построенных моделей.

RUP определяет методику и процессы разработки программ, а также средства для ее автоматизации. RUP позволяет создавать крупные и сложные системы большой командой разработчиков с различным уровнем квалификации. Он обеспечивает автоматизацию процессов разработки ПО, а также сохранность

моделей различного уровня и их согласованность при внесении изменений.

Методология RUP и CASE-средства для ее практического применения были созданы компанией Rational Software и поддерживаются в настоящее время корпорацией IBM, которая поглотила ее. Однако существуют программные продукты других разработчиков, в том числе свободно распространяемые, аналогичными функциями.

Методология RUP объединяет несколько категорий принципов построения ПО. Во-первых, это строгое формализованное описание действий конкретных сотрудников, последовательности этих действий, используемых артефактов и CASE-средств от IBM. Это описание применяется в классическом RUP-подходе. Во-вторых, это методика построения и взаимного преобразования UML-моделей программного проекта, начиная от этапа сбора требований до этапа внедрения. В-третьих, это принципы использования CASE-средств для автоматизации разработки программ.

Если первая категория принципов RUP требует покупки дорогостоящих лицензий, обучения персонала и модернизации всего процесса разработки, то вторая и третья применимы даже при индивидуальной работе. Две последние категории могут быть успешно использованы для разработки как крупных, так и небольших программ с частичной или полной автоматизацией проектирования с помощью CASE-средств любых производителей, в том числе открытым кодом.

Глава 1. МЕТОДОЛОГИЯ RUP

Унифицированный процесс состоит из четырех этапов разработки: сбора требований, проектирования, построения и внедрения. Каждый этап выполняется как серия итераций, каждая из которых обеспечивает приращение функциональных возможностей системы.

Итерация состоит из пяти рабочих процессов: 1) определение требований, 2) анализ, 3) проектирование, 4) реализация, 5) тестирование.

В этой главе подробно рассмотрены этапы и рабочие процессы, а также особенности методологии RUP.

1.1. Особенности RUP

RUP вобрал в себя преимущества спиральной и итерационной методологий разработки ПО. Согласно спиральной методологии, требования к программе формируются и уточняются постепенно, в процессе создания программы. В соответствии с инкрементной методологией происходит последовательное приращение функциональности.

RUP имеет следующие особенности:

- управляется прецедентами;
- ориентирован на архитектуру программы;
- является итерационно-инкрементным.

RUP управляется прецедентами, т. е. на всех этапах разработки в центре внимания находится прецедент. Прецедент — это прикладная задача, которая подлежит автоматизации при написании программы, например задача добавления сведений о персоне или задача поиска по каталогу.

RUP ориентирован на архитектуру — это означает, что на начальном этапе разработки определяется архитектура будущей системы. На последующих этапах разработки архитектура влияет на принятие проектных решений.

RUP является итерационно-инкрементным, т. е. при разработке происходит декомпозиция всей системы на мини-проекты, называемые итерациями. В результате выполнения очередной итерации появляется инкремент — функциональное приращение.

1.2. Этапы RUP

RUP включает в себя четыре этапа:

1) сбор требований (начало),
2) проектирование (развитие),
3) построение (конструирование),
4) внедрение (переход).

На первом этапе — анализа и планирования требований — разработчики должны убедиться, что программный проект осуществим. Для этого они выявляют и анализируют следующие параметры:

- область применения будущей системы (функциональное назначение и взаимодействие с внешней средой);
- основные требования к архитектуре системы (на базе главных функций программы);
- основные (критические) риски;
- примерную стоимость и план проекта.

Также создают демонстрационный макет для проверки основных идей разработки с последующей демонстрацией заказчику в целях уточнения требований.

В конце первого этапа оценивают его результаты. Если проект признан реализуемым в определенные заказчиком сроки и с учетом имеющихся ресурсов, то переходят ко второму этапу разработки — проектированию.

Целью этапа проектирования являются создание архитектурного базиса системы и подготовка к выполнению третьего этапа — построению.

Архитектура системы — это описание основных проектных решений, в том числе компонентов и связей между ними, используемых технологий и аппаратно-программных средств. Архитектура составляется на основе наиболее важных для заказчика функций системы, а также базового функционала, необходимого для работы всей системы. При разработке архитектуры строят модели для анализа, проектирования и реализации архитектурно значимых компонентов системы, составляющих ее «скелет». Также реализу-

ют прецеденты (прикладные функции системы), обеспечивающие и демонстрирующие работу архитектурно значимых компонентов системы.

На этапе проектирования составляют план работ для следующего этапа и осуществляют подготовку к его выполнению, в том числе определяют объем предстоящих работ, последовательность и сроки их выполнения, требуемые ресурсы, например персонал, программно-аппаратные средства и затраты на разработку, строят график работ.

Таким образом, на данном этапе решают следующие задачи:

• создают базовый уровень архитектуры системы;

• определяют существенные риски, методы их отслеживания и устранения или снижения;

• формируют требования к качеству ПО и процесса его разработки, например стандарты;

• составляют финансовый план проекта;

• проводят анализ большинства прецедентов (до 80 %);

• готовят план итераций для выполнения следующего этапа.

В результате проектирования будут созданы основные проектные модели, реализован «скелет» проекта, устранены основные риски, подготовлены ресурсы и среда для дальнейшей разработки. Организация разработчика будет готова перейти к следующему этапу и приступить к реализации ПО.

Третий этап разработки называют построением. Его основной задачей является создание работоспособного программного продукта. На этом этапе проводят полную разработку и тестирование компонентов системы, включая описание требований, анализ, проектирование, реализацию и тестирование всех прецедентов.

В ходе построения решают следующие вопросы:

• управление ресурсами;

• оптимизация процессов разработки;

• отслеживание существенных и критических рисков;

• оценка качества создаваемого ПО.

При построении системы создают ПО, готовое к внедрению, а также документацию, например руководство оператора. Построение выполняется в виде серии инкрементных итераций.

Последним, четвертым, этапом разработки ПО является его внедрение. Готовое и протестированное ПО устанавливают для

эксплуатации конечными пользователями и готовят пользователей для работы с ним.

На данном этапе выполняют следующие задачи:

• завершение реализации ПО;

• формирование рекомендаций по установке и эксплуатации ПО;

• подготовка программно-аппаратного обеспечения для работы с ПО (конечными пользователями);

• применение ПО в среде конечного пользователя;

• составление документации и руководств (в соответствии с требованиями заказчика);

• проведение бета-тестов, выявление и устранение ошибок ПО;

• создание версии ПО для приемосдаточных испытаний.

Стоит напомнить, что альфа-тестирование проводит заказчик в среде разработчика, а бета-тестирование — конечный пользователь в среде заказчика.

Результатом данного этапа будет ПО, разработанное по требованиям заказчика и готовое для дальнейшей эксплуатации. На этом процесс разработки считается завершенным.

1.3. Рабочие процессы и итерации

Каждый этап разработки ПО состоит из нескольких итераций.

Итерация — это полный цикл разработки фрагмента ПО, в результате которого формируется промежуточная версия, реализующая некоторый функционал. В ходе разработки происходит постепенное усложнение создаваемого ПО, наращивание его функциональных возможностей.

Каждая итерация состоит из последовательного выполнения пяти рабочих процессов:

1) сбор требований — определение исходных требований к создаваемой системе (или ее фрагментам);

2) анализ — преобразование требований в классы и объекты (построение моделей анализа);

3) проектирование — создание статического и динамического представлений системы для выполнения исходных требований (построение проектных моделей);

4) реализация — процесс генерации программного кода (включая тестирование и отладку модулей);

5) тестирование — проверка работоспособности системы в целом.

При каждой итерации осуществляют все рабочие процессы, но удельный вес конкретного рабочего процесса зависит от этапа разработки:

- на первом этапе основное внимание уделяют сбору требований;
- на этапе проектирования — анализу и проектированию;
- на этапе построения — реализации;
- на этапе тестирования — тестированию.

Примерное распределение работ по рабочим процессам в зависимости от этапа разработки приведено на рис. 1.1.

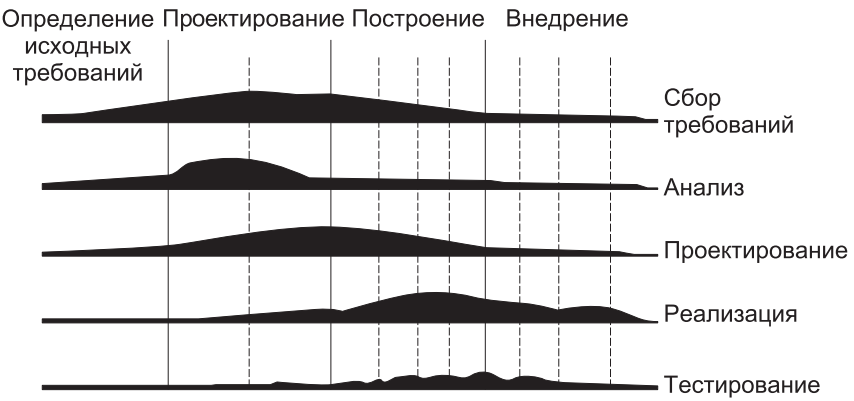


Рис. 1.1. Рабочие процессы на этапах RUP

На первом этапе осуществляется предварительная итерация, на втором — две-три итерации. Первые два этапа выполняет небольшое количество специалистов высокой квалификации. Это самые важные этапы, от качества их проведения зависит успех всего процесса разработки.

Этап построения состоит из многих однотипных итераций, которые могут осуществляться параллельно разными группами разработчиков. Это самый длительный этап, для реализации которого требуется большое количество ресурсов. Этап внедрения тоже включает в себя несколько итераций, но их меньше, чем при построении. Третий и четвертый этапы выполняются многими разработчиками разной квалификации.

В следующих главах подробно рассмотрены действия рабочих процессов.

Выводы

RUP объединяет спиральную и итерационную методологии разработки ПО.

RUP управляется прецедентами, ориентирован на архитектуру программы, является итерационно-инкрементным.

RUP состоит из четырех этапов: сбора требований, проектирования, построения и внедрения.

Разработка ПО выполняется как серия итераций.

Каждая итерация включает пять рабочих процессов: определение требований, анализ, проектирование, реализация и тестирование.

Контрольные вопросы и задания

1. Что такое унифицированный процесс разработки? Для чего он предназначен и каковы его особенности?
2. Что означает «RUP управляется прецедентами»?
3. Как понимать, что RUP ориентирован на архитектуру программы?
4. Почему RUP является итерационно-инкрементным?
5. Перечислите этапы RUP и поясните их содержимое.
6. Из каких действий состоит этап сбора требований? Какова его цель? Что будет результатом его выполнения?
7. Из каких действий состоит этап проектирования? Каковы его задачи? Что будет результатом его выполнения?
8. Из каких действий состоит этап построения? Какова его цель? Что будет результатом его выполнения?
9. Из каких действий состоит этап внедрения? Какова его цель? Что будет результатом его выполнения?
10. Что такое итерация и какие рабочие процессы включены в нее?
11. Определите действия каждого из рабочих процессов.
12. Как рабочие процессы итерации зависят от этапа RUP?

Глава 2. РАБОЧИЙ ПРОЦЕСС «ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ»

Каждая итерация унифицированного процесса начинается с определения требований к создаваемой системе. При этом необходимо выполнить следующие действия:

- перечислить кандидатов в требования;
- осознать контекст системы;
- определить функциональные требования;
- определить нефункциональные требования.

Рассмотрим эти действия подробно.

2.1. Определение требований

Требования — это пожелания заказчика по возможностям создаваемого ПО.

Данная информация используется для последующего планирования итераций. На основе требований будут выделены функции системы и определен порядок их реализации.

2.1.1. *Перечисление кандидатов в требования*

На основе пожеланий заказчика формируется список предложений, который содержит для каждого требования:

- его название и краткое описание;
- оценку затрат на реализацию;
- приоритет;
- возможные риски.

Рассмотрим пример конкретной разработки. Допустим, разрабатывается автоматизированная система управления (АСУ) таксопарком. Диспетчер принимает звонок клиента, оформляет заказ, передает заказ водителю и отслеживает выполнение заказа. Ведется база данных (БД) сотрудников и графиков их дежурств.

На основе пожеланий заказчика выделяются кандидаты в требования: «Ведение БД сотрудников», «Обработка заказа», «Отслеживание состояния выполнения заказа», «Ведение БД графиков работы», «Отслеживание больничных». Назначаются приоритеты, выявляются риски и проводится предварительная оценка затрат на разработку каждого требования.

После выявления начальных требований можно переходить к их детальному рассмотрению.

2.1.2. *Осознание контекста системы*

Перед формализацией требований к ПО исследуется деятельность, которую предполагается автоматизировать, в том числе выявляются участники этой деятельности (сотрудники и клиенты), объекты реального мира, необходимые для выполнения деятельности, и процессы деятельности.

Контекст системы — это формальное описание предметной области и бизнес-процессов, для которых разрабатывается ПО. Под бизнес-процессами понимают деятельность организации или предприятия, необходимую для выполнения их функций, например, по обслуживанию клиентов. Контекст системы позволяет выделить и формализовать основные понятия, сущности и процессы предметной области для их последующей автоматизации. При описании контекста системы формируют модели предметной области и бизнес-модели.

Модель предметной области включает в себя сведения об основных сущностях реального мира, их атрибутах и связях, имеющих значение для разрабатываемой системы. Она задается диаграммой классов и/или глоссарием понятий.

Диаграмма классов предметной области содержит:

- бизнес-объекты — сущности, используемые в бизнес-процессах, например счета или заявки;
- объекты и понятия реального мира, например поезд или маршрут;
- события в реальном мире, например прибытие поезда или перерыв на обед.

Диаграмма составляется только по самым важным классам.

Глоссарий понятий состоит из терминов и их определений и позволяет разработчику и заказчику общаться на одном языке.

Для рассматриваемого примера разработки АСУ таксопарком (см. 2.1.1) может быть составлена диаграмма классов предметной области, приведенная на рис. 2.1.

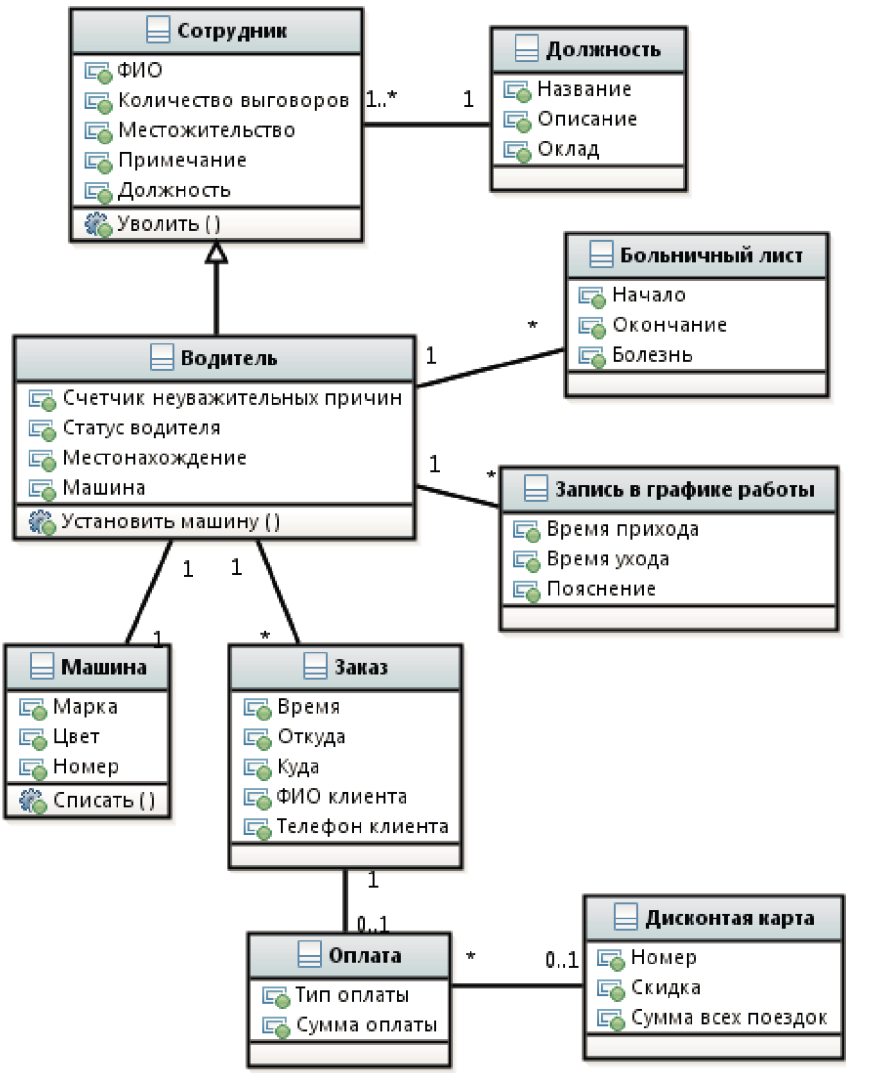


Рис. 2.1. Диаграмма классов предметной области

Бизнес-модель описывает бизнес-процессы в организации в терминах бизнес-прецедентов (деятельность бизнеса, полезная для участников бизнеса) и бизнес-актеров (участников бизнеса, например, клиентов). Она представляет разрез будущей системы с точки зрения клиентов организации. Бизнес-модель включает в себя бизнес-модели прецедентов и объектные бизнес-модели.

Бизнес-модель прецедентов является диаграммой прецедентов и содержит:

- бизнес-актеров;
- бизнес-прецеденты.

На рис. 2.2 приведена диаграмма бизнес-прецедентов для АСУ таксопарком.

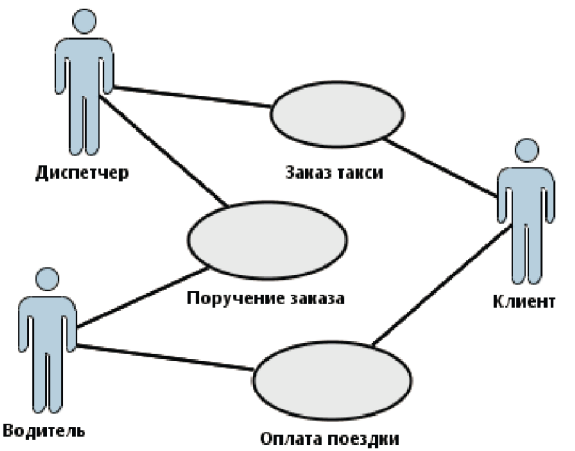


Рис. 2.2. Диаграмма бизнес-прецедентов для АСУ таксопарком

Объектная бизнес-модель описывает реализацию бизнес-прецедента и включает в себя:

- сотрудников организации (например, оператора);
- бизнес-сущности (например, изделие, счет или заказ);
- рабочие модули, которые являются объединением бизнес-сущностей и представляют для конечного пользователя единое целое.

Объектные бизнес-модели составляют в форме диаграмм взаимодействия и/или диаграмм деятельности.

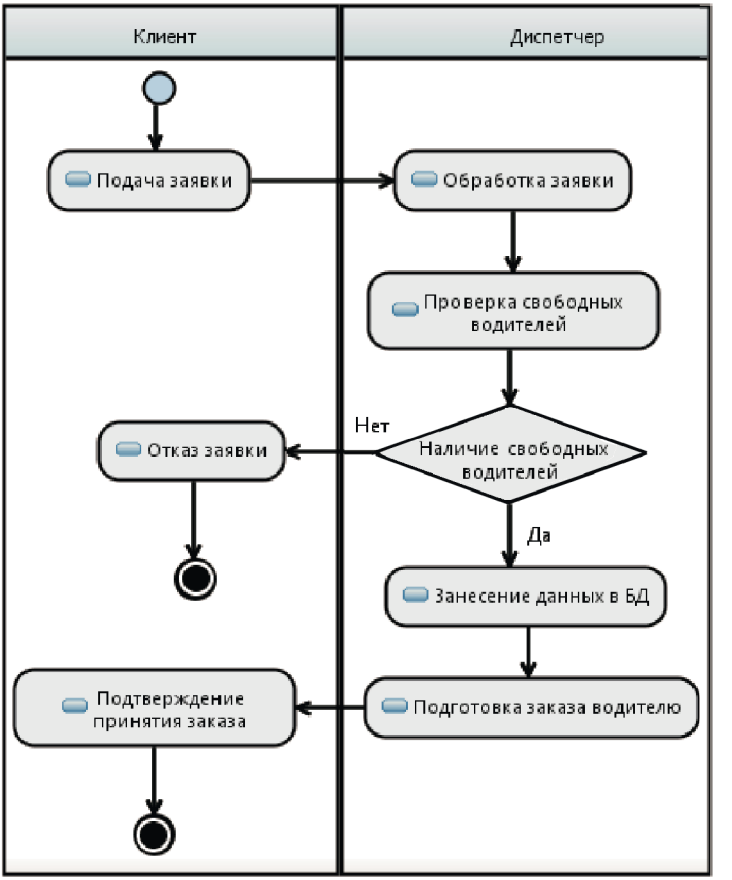


Рис. 2.3. Диаграмма деятельности «Заказ такси»

На рис. 2.3 приведена объектная бизнес-модель для бизнес-прецедента «Заказ такси» в форме диаграммы деятельности.

2.1.3. *Определение функциональных требований*

Функциональные требования — это требования к функциям, выполняемым разрабатываемой системой. В результате определения функциональных требований составляют модель прецедентов, которая содержит:

• актеров — изначально по одному на каждого сотрудника или клиента бизнеса;

• роли актеров — по одной на каждый бизнес-прецедент, где участвует этот актер;

• прецеденты — по одному на каждую роль актера.

Далее выполняют детализацию прецедентов, после чего выявляют прецеденты для автоматизации (необязательно все из рассмотренных) и проводят реорганизацию модели прецедентов.

Отметим, что на этом этапе формируется модель прецедентов для процессов, подлежащих автоматизации, в то время как модель бизнес-прецедентов описывает реальную деятельность организации. Не все бизнес-процессы будут и могут быть автоматизированы.

Поскольку определение функциональных требований является основной задачей данного рабочего процесса, состоит из нескольких шагов, выполняется командой разработчиков и занимает существенное время, то далее это определение будет рассмотрено более подробно.

2.1.4. *Определение нефункциональных требований*

К нефункциональным требованиям относится все, что не является функционалом, но должно быть учтено при разработке ПО:

• ограничения среды и реализации;

• требования по производительности (скорость, пропускная способность, время ответа, оперативная память и т. д.);

• зависимость от платформы;

• расширяемость;

• надежность (точность, средняя наработка на отказ и т. д.);

• требования к пользовательскому интерфейсу;

• требования к форматам и протоколам передачи данных;

• наличие внешних компонентов или подсистем;

• требования по безопасности;

• легкость установки, использования и обучения персонала.

Эти требования не связаны с конкретным прецедентом. Они относятся к нескольким прецедентам сразу или не относятся ни к одному прецеденту.

В результате выявления нефункциональных требований составляют дополнительные требования к существующим прецедентам или добавляют новые прецеденты.

2.2. Определение функциональных требований в виде прецедентов

Основным шагом при определении требований является формализация функциональных требований в виде прецедентов, что позволяет выявить будущих пользователей и функции разрабатываемой системы, а также определить пользовательский интерфейс. В результате выполнения данной задачи формируются следующие артефакты:

- актеры, соответствующие группам пользователей системы;
- прецеденты — набор функций системы, реализующих полезную для пользователей деятельность;
- прототип пользовательского интерфейса;
- модель прецедентов, которая содержит систему прецедентов из множества актеров, прецедентов и связей между ними;
- глоссарий (словарь) терминов.

Первые три артефакта создает системный аналитик. За прецеденты отвечает спецификатор прецедентов, за прототип пользовательского интерфейса — разработчик пользовательского интерфейса.

Рассматриваемый процесс состоит из выполнения следующих действий:

- нахождение актеров и прецедентов;
- определение приоритетов прецедентов;
- детализация прецедентов;
- создание прототипа пользовательского интерфейса;
- структурирование модели прецедентов.

Рассмотрим перечисленные действия более подробно.

2.2.1. *Нахождение актеров и прецедентов*

Прежде всего следует идентифицировать актеров. Актер — это внешний субъект, который взаимодействует с системой. Актером может быть как человек — оператор, так и внешняя программа или устройство. При идентификации актеров необходимо выделить:

- по одному актеру на каждого бизнес-сотрудника или бизнес-клиента (из бизнес-моделей, построенных ранее);
- по одному или более актеров на каждую категорию пользователей системы;

• по одному актеру на внешнее устройство;

• по одному или более актеру на внешнюю систему.

Для актеров определяют их роли: по одной на каждый бизнес-процесс для данного актера. Для уменьшения возможного пересечения ролей применяют обобщение.

Идентификацию актеров и выделение ролей проводят итерационно, на основе консультаций с заказчиком.

После выявления актеров и определения их ролей переходят к идентификации прецедентов. Прецедент — это функция разрабатываемой системы, выявленная на основе функционального требования. Множество прецедентов системы — это набор задач, подлежащих автоматизации.

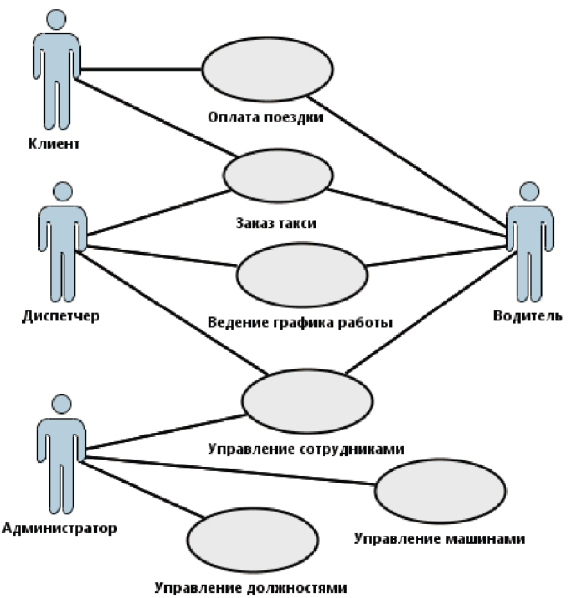


Рис. 2.4. Диаграмма прецедентов для АСУ таксопарком

Для правильной идентификации прецедентов необходимо учитывать следующее:

• прецедент приносит ценный (видимый) результат отдельному актеру;

• прецедент является самодостаточным и не является продолжением другого прецедента;

• прецедент инициируется актером;

• прецедент взаимодействует только с актером или актерами, но не с другими прецедентами.

Первоначально прецеденты выделяются на основе построенной ранее бизнес-модели: по одному прецеденту для каждой роли актера.

Для каждого прецедента составляется его краткое описание.

После определения актеров и прецедентов выполняется описание модели прецедентов в целом. Модель прецедентов содержит произвольный набор диаграмм прецедентов, глоссарий понятий и комментарии. На рис. 2.4 приведена диаграмма прецедентов для АСУ таксопарком.

На диаграммах прецедентов размещают актеров и прецеденты, а также связи ассоциаций между ними, которые показывают участие актеров в прецедентах. Каждый прецедент должен быть связан не менее чем с одним актером, который его инициирует. Допускается указывать связи обобщения между актерами для выделения общих ролей.

2.2.2. *Определение приоритетов прецедентов*

Каждому прецеденту назначается приоритет, показывающий его важность. Наибольшей важностью обладают прецеденты, реализующие ключевые функции системы, а также базовый функционал, обеспечивающий работу всей системы.

2.2.3. *Детализация прецедентов*

Все выявленные прецеденты, содержащиеся в модели прецедентов, необходимо детализировать, структурировать и описать в терминах формальных моделей. Для каждого прецедента необходимо указать порядок и условия его запуска и завершения, алгоритм выполнения и потоки данных между системой и актерами при выполнении прецедента.

Структурирование прецедентов. Структурирование прецедентов выполняется на основе графа переходов между состояниями системы. Граф переходов включает:

• начальное состояние;

• активацию (запуск) прецедента внешним событием, инициируемым актером;

• переходы между состояниями и выполняемые при этом действия, в том числе посылку сигналов другим актерам;

• уничтожение прецедента.

При структурировании прецедентов различают основной путь как наиболее типовой алгоритм выполнения прецедента и альтернативные пути, реализующие либо варианты базового алгоритма, обусловленные выбором пользователя, либо обработку исключений, например ошибки ввода, ошибки системных ресурсов или внешние сигналы.

Структурированное описание прецедента включает:

• предусловие — исходное состояние системы до начала прецедента;

• поток событий — запуск прецедента, порядок шагов и завершение прецедента по основному пути;

• постусловие — возможное конечное состояние системы после завершения прецедента;

• описание альтернативных путей в основном пути (варианты выполнения алгоритма);

• описание альтернативных путей вне основного пути (обработка ошибок);

• сообщения, которыми обмениваются актеры и прецеденты, и последовательность этих сообщений;

• атрибуты прецедента, необходимые для его выполнения, например ресурсы, объекты или значения.

Все действия системы и актеров в ходе работы прецедента должны быть описаны явно и однозначно.

Далее приведен пример спецификации прецедента «Управление сотрудниками».

Предусловие: прецедент начинается, когда диспетчер запускает программу «Ведение БД».

Поток событий: запуск прецедента; диспетчеру на выбор предлагаются варианты: «Управление сотрудниками», «Управление должностями», «Управление машинами». Он выбирает «Управление сотрудниками».

Основной поток: система отображает окно, в котором находится таблица, включающая в себя список всех сотрудников и инфор-

мацию о них; диспетчер выбирает сотрудника, уточняет информацию о нем, изменяет эту информацию прямо в таблице, нажимает кнопку «Сохранить изменения», или кнопку «Удалить сотрудника» (переход к альтернативному потоку Е-1), или кнопку «Добавить сотрудника» (переход к альтернативному потоку Е-2). Система вносит изменения в БД, формирует и отображает сообщение об успешности изменений.

Альтернативные потоки:

• Е-1 — для удаления информации о сотруднике диспетчер нажимает кнопку «Удалить сотрудника», выделив нужную строку; система удаляет запись в БД, формирует и отображает сообщение об успешности;

• Е-2 — для добавления информации о сотруднике диспетчер заполняет пустую строку в таблице, нажимает кнопку «Добавить сотрудника»; система добавляет запись о новом сотруднике в БД, формирует и отображает сообщение об успешности.

Постусловие: в БД внесены изменения. Диспетчер получил сообщения о результате внесения изменений.

К структурированному описанию прецедента добавляют специальные (нефункциональные) требования, например ограничения на время отклика или объемы используемых ресурсов.

Формализация прецедентов. После структурирования проводят формализацию описания прецедента. Для этого составляют произвольные наборы диаграмм, в том числе:

• диаграмму состояний, содержащую состояния системы и задающую переходы между ними в прецеденте;

• диаграмму деятельности, которая описывает алгоритм выполнения прецедента и участвующих в нем актеров;

• диаграмму взаимодействия, показывающую сообщения между актерами и системой в прецеденте.

2.2.4. *Создание прототипа пользовательского интерфейса*

Прототип пользовательского интерфейса позволяет на начальной стадии разработки ПО согласовать с заказчиком внешний интерфейс создаваемой системы, а также уточнить набор и описание прецедентов.

Исходными данными для разработки прототипа пользовательского интерфейса являются:

• модель и описания прецедентов;
• дополнительные требования;
• глоссарий понятий.

Прототип пользовательского интерфейса разрабатывается в два этапа: сначала создается логический проект, а затем составляются проект и прототип.

Создание логического проекта пользовательского интерфейса. При создании логического проекта пользовательского интерфейса выявляют данные, используемые актером в его прецедентах, а также действия актера с этими данными. Необходимо для каждого актера по всем его прецедентам определить:

• элементы пользовательского интерфейса для прецедента, выявляемые на основе атрибутов классов предметной области или бизнес-объектов, используемых в прецеденте, например поля для ввода номера счета или банковских реквизитов;
• связи между выявленными элементами;
• внешний вид элементов и возможности работы с ними, например редактирование текста или увеличение масштаба изображения;
• действия и принимаемые в прецеденте решения актера, например получение выписки по счету или отмена заказа;
• информацию, отображаемую и передаваемую в прецеденте от актера и к актеру;
• средние или экстремальные значения отображаемых и передаваемых параметров, например количество элементов в списке или размер текста.

Логический проект является предварительным наброском и используется для создания проекта и прототипа пользовательского интерфейса.

Создание проекта и прототипа пользовательского интерфейса. На последующем этапе к наборам элементов, созданных при логическом проектировании, добавляют вспомогательные элементы, например кнопки или полосы прокрутки, в результате чего будут получены эскизы пользовательского интерфейса и наброски экранов.

На рис. 2.5 и 2.6 приведены прототипы пользовательского интерфейса для АСУ таксопарком.

После создания проекта и прототипа пользовательского интерфейса проводится их сравнение с детальным описанием прецедентов, что позволяет проверить:

• целостность и логичность описания последовательности действий, необходимых для выполнения алгоритма прецедента;

• достаточность информации для выполнения прецедента;

• полноту возможностей взаимодействия актера (пользователя) с системой;

• отсутствие лишних сведений при работе актера (пользователя).

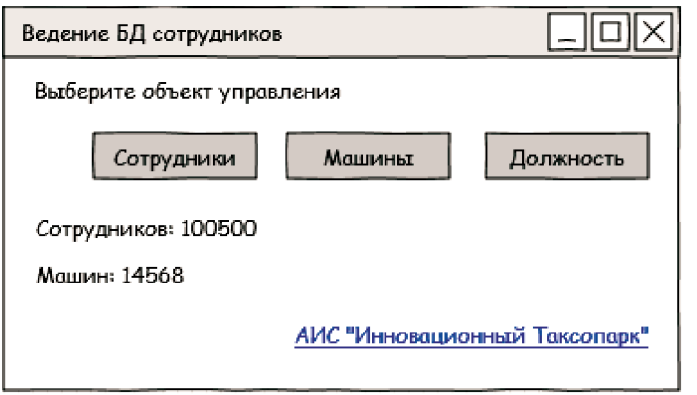


Рис. 2.5. Прототип 1 пользовательского интерфейса АСУ таксопарком

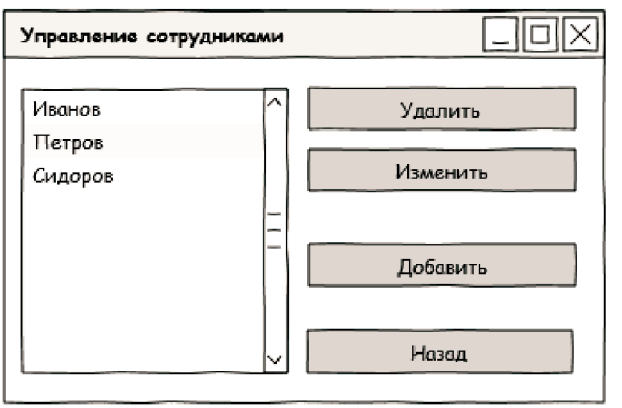


Рис. 2.6. Прототип 2 пользовательского интерфейса АСУ таксопарком

По итогам проверки возможна корректировка как прототипа пользовательского интерфейса, так и описания прецедента.

2.2.5. *Структурирование модели прецедентов*

После того как будут детализированы, структурированы, формализованы и выверены описания всех прецедентов, следует уточнить и скорректировать общую модель прецедентов.

При структурировании модели прецедентов происходит изменение набора прецедентов и их описаний. Прецеденты могут быть декомпозированы, а на диаграмму прецедентов добавлены связи обобщения, включения или расширения между прецедентами. Кроме того, обновляются диаграммы прецедентов и описания прецедентов.

На рис. 2.7 приведен пример структурированной диаграммы прецедентов для подсистемы «Ведение БД».

Общие или совместно используемые описания функциональности в различных прецедентах выделяют в отдельные прецеденты и указывают связь между прецедентами, реализующими общий функционал, и прецедентами, реализующими уникальный функционал, через обобщение.

Дополнительные или необязательные описания функциональности в прецедентах выделяют в отдельные прецеденты и указывают связь между прецедентами, реализующими базовый функционал, и прецедентами, реализующими дополнения, через зависимость со стереотипом «extend». Такая связь называется расширением. При этом в описании базового прецедента указывается точка расширения, в которой подключается дополнительный функционал, и условия расширения, при которых он будет подключен.

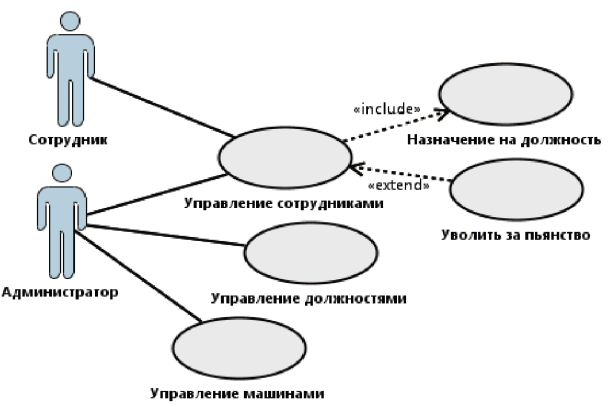


Рис. 2.7. Структурированная диаграмма прецедентов для подсистемы «Ведение БД» АСУ таксопарка

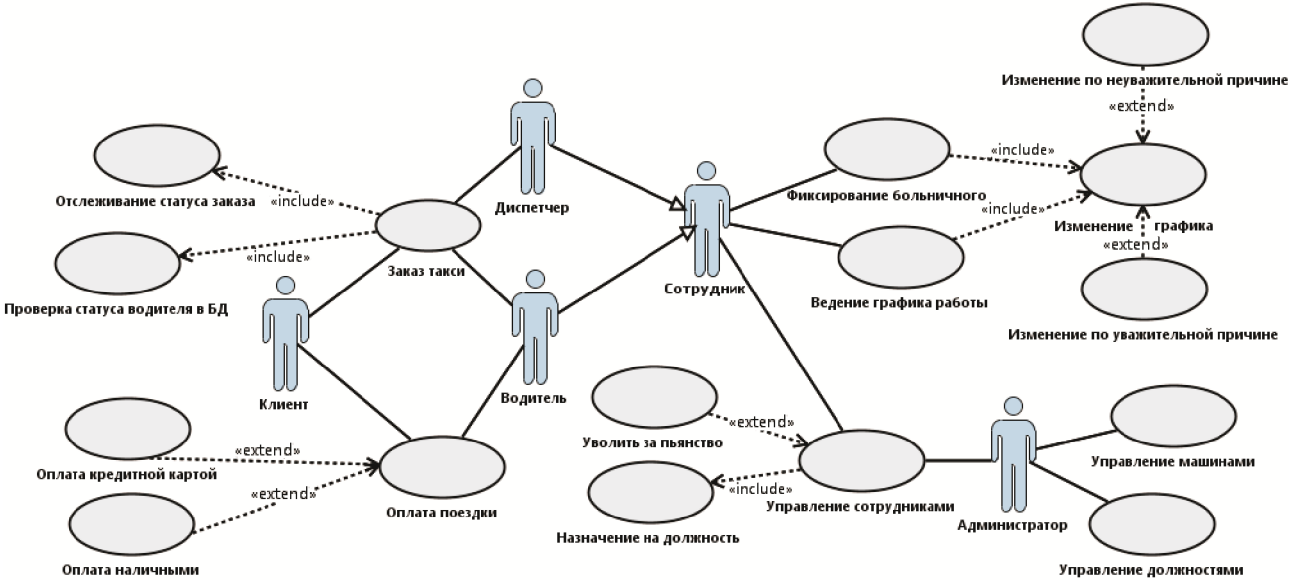


Рис. 2.8. Общая модель прецедентов для АСУ таксопарком

Включение в прецедент закрытой функциональности без доступа к ее внутренним атрибутам задается зависимостью со стереотипом «include». Такую связь называют включением. Описание закрытой функциональности выносят в отдельный прецедент.

На диаграмме (см. рис. 2.7) через расширение «extend» выделен прецедент «Уволить за пьянство», через включение «include» — «Назначение на должность».

Расширяющий прецедент («Уволить за пьянство») дополняет расширяемый («Управление сотрудниками») функционалом, который включается при некотором условии. Расширяемый не знает своего расширяющего. Расширяющий имеет доступ к атрибутам расширяемого.

Подключаемый прецедент («Назначение на должность») является для расширяемого («Управление сотрудниками») внешней функцией (закрытой), которая будет реализована как подключение сервиса. Подключаемый функционал независим и не знает о тех, кто его использует.

Определение функциональных требований завершается обновлением общей модели прецедентов. Ее пример для АСУ таксопарком приведен на рис. 2.8.

Выводы

Рабочий процесс определения требований начинает итерацию. На основе пожеланий заказчика и анализа предметной области определяют пользователей и функции разрабатываемой системы, создают прототип пользовательского интерфейса и модель прецедентов.

Процесс состоит из следующих действий: перечисление кандидатов в требования; осознание контекста системы, определение функциональных и нефункциональных требований.

Описание контекста системы содержит модель предметной области и/или бизнес-модель. Модель предметной области включает диаграммы классов предметной области и глоссарий понятий. Бизнес-модель объединяет диаграммы бизнес-прецедентов и объектные бизнес-модели.

Рабочий процесс «Определение функциональных требований» выполняется как последовательность таких действий, как нахождение актеров и прецедентов; определение приоритетов прецеден-

тов; детализация прецедентов; создание прототипа пользовательского интерфейса; структурирование модели прецедентов.

В результате определения функциональных требований выявляют актеров и прецеденты, формируют прототип пользовательского интерфейса, модель прецедентов и глоссарий терминов.

Контрольные вопросы и задания

1. Каково назначение рабочего процесса определения требований?
2. Какие артефакты создаются в процессе выполнения рабочего процесса?
3. Объясните, из каких действий состоит рабочий процесс.
4. Что такое требование? В чем заключаются различия функциональных и нефункциональных требований?
5. Приведите примеры нефункциональных требований.
6. Что такое контекст системы? Из чего состоит его описание?
7. Расскажите, что содержит и как составляется модель предметной области.
8. Объясните, что содержит и как составляется бизнес-модель. Какие диаграммы в нее входят? Расскажите о содержании диаграммы бизнес-прецедентов и объектной бизнес-модели.
9. Из каких действий состоит определение функциональных требований?
10. Объясните, какие модели и диаграммы будут построены в результате определения функциональных требований.
11. Что такое актер и прецедент? Как их идентифицируют?
12. Что такое прототип пользовательского интерфейса? Как его составляют?
13. Объясните содержание модели прецедентов, ее элементы и диаграммы.
14. Как и для чего выполняют структурирование прецедентов?
15. Что содержится в спецификации прецедента? Для чего ее составляют?
16. Как и зачем выполняют структурирование модели прецедентов? Что означают связи включения и расширения и в чем их различия?

Глава 3. РАБОЧИЙ ПРОЦЕСС «АНАЛИЗ ТРЕБОВАНИЙ»

После определения требований к создаваемому ПО их подвергают анализу. Анализ переводит требования, полученные от заказчика, в предварительную модель будущей системы.

В ходе анализа выделяют основные компоненты будущего ПО и связи между ними. Анализ осуществляют на инфологическом уровне, и его результаты не привязаны к конкретным технологиям разработки, программно-аппаратным платформам или языкам программирования.

Процесс анализа позволяет:

- уточнить спецификацию требований;
- перейти от языка заказчика к языку разработчика;
- повысить формализм описаний;
- провести анализ внутренних механизмов работы системы;
- структурировать требования для дальнейшей разработки ПО.

В результате выполнения анализа требований будут получены следующие артефакты:

- модель анализа, содержащая систему анализа из множества пакетов, классов и кооперaций анализа;
- классы анализа, представляющие начальное разбиение системы на классы;
- анализ кооперaций, который показывает реализацию прецедентов в терминах классов и связей между ними;
- пакеты анализа — логические контейнеры классов анализа;
- сервисные пакеты, включающие классы для выполнения сервисных функций.

Модель анализа составляет архитектор, классы и пакеты разрабатывает инженер по компонентам, кооперации анализирует инженер по прецедентам.

На стадии анализа необходимо выполнить следующие действия:

- анализ архитектуры системы;
- анализ прецедентов;

• анализ классов;
• анализ пакетов.

Рассмотрим эти действия более подробно.

3.1. Анализ архитектуры

Архитектура системы задает ее основные компоненты и связи между ними. При анализе архитектуры необходимо определить пакеты, выявить их общие части и сервисные пакеты, определить зависимости между пакетами, выявить классы сущностей и сформулировать специальные требования к системе [8].

3.1.1. *Идентификация пакетов анализа*

Анализ архитектуры начинают с идентификации пакетов анализа. Пакет — это термин, используемый на этапе анализа для обозначения логического контейнера классов. Классы объединяют в пакеты на основе их функциональной связности.

Первоначально пакеты выделяют на основе функциональных требований — например, пакет как набор классов для реализации множества прецедентов одного актера или одного бизнес-процесса. Затем выделенное множество пакетов подвергают пересмотру путем обобщения и расширения.

Пакет анализа содержит классы анализа, вложенные пакеты анализа и анализ кооперации. Поскольку анализ проводится на логическом уровне, то пакет содержит классы для двух верхних (прикладных) уровней приложения.

На рис. 3.1 приведен пример пакетов для АСУ таксопарком.

При идентификации пакетов анализа стремятся достичь:

• сильной связности классов внутри пакета и слабого сцепления между пакетами;

• возможности параллельного анализа отдельных пакетов.

3.1.2. *Определение общих частей пакетов анализа*

После первичной идентификации пакетов анализа проводят определение их общих частей. Это позволит избежать дублирования кода в разных пакетах.

Если два и более пакетов используют общий класс анализа, то класс надо вынести в отдельный пакет анализа или просто отдельно, обычно это классы бизнес-сущностей.

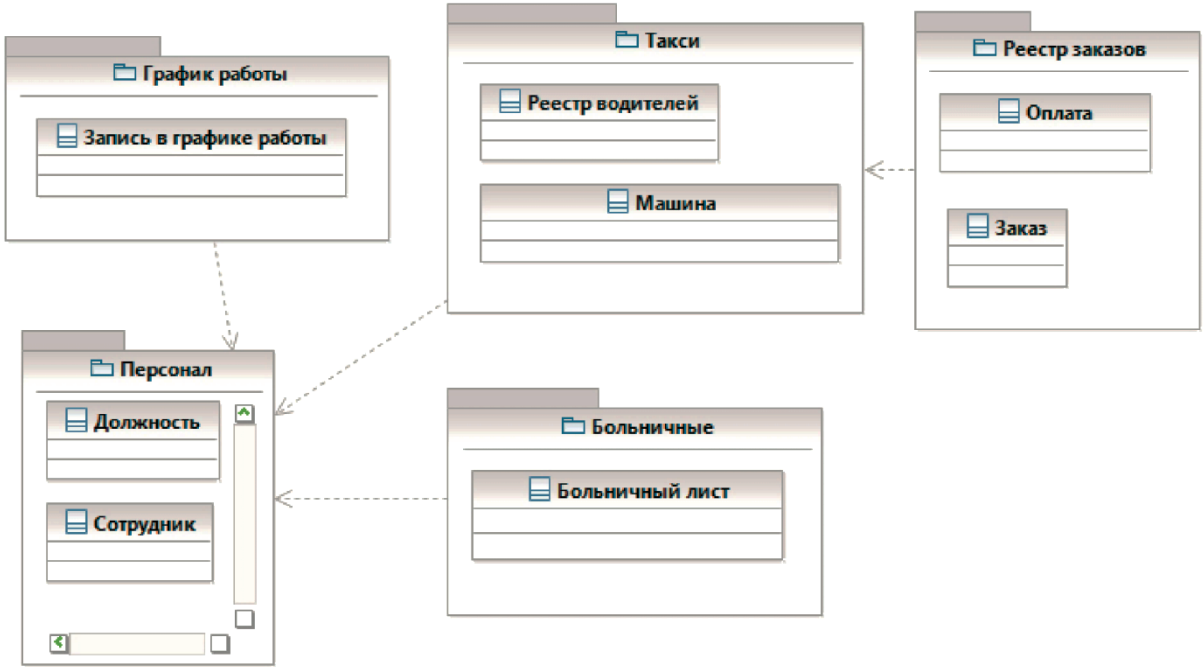


Рис. 3.1. Пакеты анализа для АСУ таксопарком

3.1.3. *Идентификация сервисных пакетов*

После идентификации пакетов анализа следует идентифицировать сервисные пакеты.

Сервисный пакет — это прообраз будущего компонента, который реализует некоторую законченную и подключаемую функциональность для клиента. Это может быть блок математических функций или блок по обслуживанию банковского счета. Сервисный пакет можно заменить на аналогичный (например, от другого производителя). Обычно сервисные пакеты взаимозаменяемы.

Для идентификации сервисных пакетов необходимо иметь в виду, что сервисный пакет:

- содержит множество функционально связанных классов;
- реализует набор функциональности для клиента;
- неделим при приобретении в качестве компонента системы (принцип «все или ничего»);
- может повторно использоваться;
- обычно используется несколькими прецедентами.

На этапе анализа следует определить по одному сервисному пакету на каждый необязательный сервис, представленный функционально связанными классами.

3.1.4. *Определение зависимостей между пакетами анализа*

Для выявления связей между пакетами анализа необходимо учитывать следующее:

- пакеты связаны между собой зависимостью в соответствии с использованием функционала (функционал зависимого пакета используется зависящим пакетом);
- внутри пакета существует высокая связность, между пакетами — низкое сцепление;
- при определении зависимостей пакетов происходит деление на уровни.

Прикладной уровень приложения, рассматриваемый на этапе анализа, состоит из двух подуровней: специфического и общего уровней приложения. Пакеты специфического уровня реализуют оригинальный функционал подсистем или рабочих мест. Пакеты

общего уровня приложения реализуют функционал, используемый различными подсистемами. Пакеты специфического уровня зависят от пакетов общего уровня.

3.1.5. *Определение очевидных классов сущностей*

Кроме пакетов при анализе архитектуры выделяют и описывают очевидные классы сущностей. Классы сущностей показывают логическую структуру данных. Они используются для моделирования долгоживущей, часто сохраняемой информации о человеке, объекте или событии реального мира и могут иметь сложное поведение.

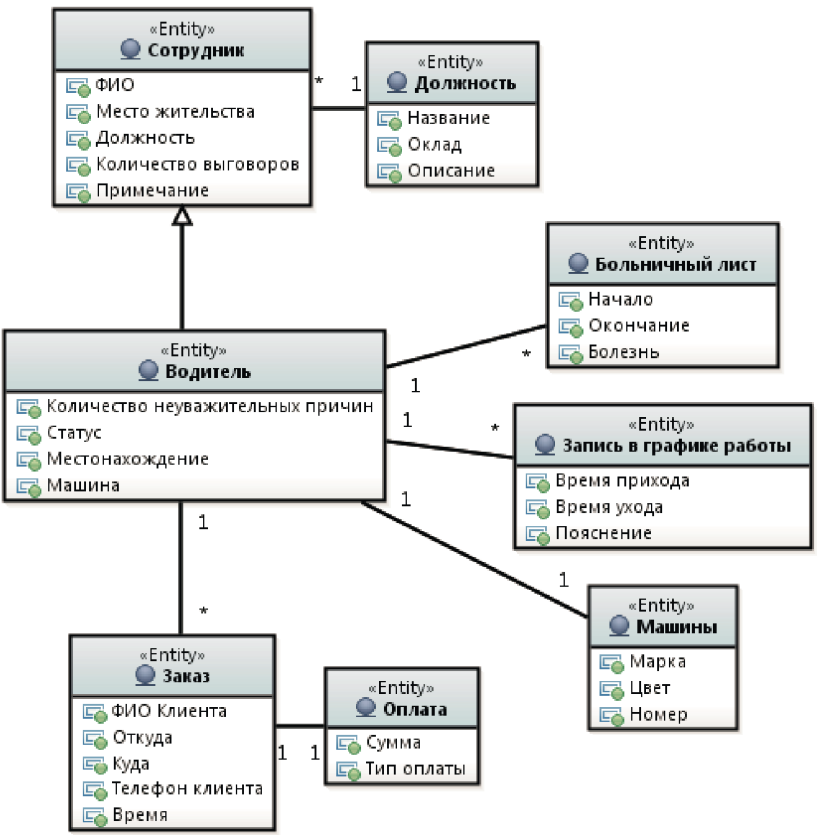


Рис. 3.2. Диаграмма классов сущностей для АСУ таксопарком

Классы сущностей определяют на основе модели предметной области или бизнес-модели. Выделяют те классы, которые участвуют в прецедентах. Обычно их от десяти до двадцати.

На рис. 3.2 приведен пример диаграммы классов сущностей для АСУ таксопарком.

Для каждого очевидного класса сущности определяют его атрибуты. Затем задают связи ассоциации между классами сущностей.

3.1.6. *Определение общих специальных требований*

Анализ архитектуры завершается определением общих специальных (нефункциональных) требований. К специальным требованиям архитектуры системы относят:

- требования к хранению данных (хранимые и передаваемые объемы, частота изменений);
- распределенную и параллельную обработку;
- требования по безопасности;
- устойчивость к сбоям;
- управление транзакциями.

Перечисленные требования влияют на анализ и последующее принятие решений при проектировании и реализации.

Характеристики требований должны быть приписаны к тому классу (или кооперации), на который ссылаются.

3.2. Анализ прецедента или кооперации

После анализа архитектуры системы в целом переходят к анализу отдельных прецедентов. При анализе прецедента необходимо определить классы анализа, участвующие в прецеденте, и связи между этими классами, обеспечивающие выполнение конкретных прецедентов.

Прецедент — это некоторая функция, осуществляемая системой в интересах ее пользователя. Для реализации прецедента необходим программный компонент, выполняющий данную функцию. Реализация прецедента называется кооперацией и имеет две составляющие: статическую и динамическую. Статическая составляющая задается диаграммой классов, необходимых для выполнения прецедента. Динамическая составляющая определяет алгоритм реализации прецедента и задается диаграммами взаимодействия.

3.2.1. *Определение классов анализа*

Класс анализа является концептуальной единицей, не привязан к конкретному языку программирования, и при его определении следует учитывать, что класс анализа:

• ориентирован на функциональные требования;

• имеет текстовое описание ответственности (перечень выполняемых функций) без сигнатур;

• содержит атрибуты логических типов.

При этом отношения между классами анализа концептуальны.

Классы анализа бывают трех типов: граничный, управляющий или класс сущности.

Классы сущности были рассмотрены ранее (см. 3.1.5). Их назначением является хранение долговременной информации. Класс сущности обозначается стереотипом «entity».

Граничный класс выделяют для моделирования взаимодействия между системой и актерами (получение и передача информации или запроса). Граничный класс соответствует пользовательскому интерфейсу, интерфейсу внешних устройств или коммуникационному протоколу, библиотеке API на высоком уровне абстракции без физической реализации. Граничный класс обозначается стереотипом «boundary».

Управляющий класс выполняет координацию, последовательность, взаимодействие и управление другими объектами для прецедента, обрабатывает и координирует действия и потоки управления, реализует бизнес-логику. Управляющий класс обозначается стереотипом «control».

Классы анализа могут участвовать в нескольких прецедентах. Поэтому в начале анализа прецедентов следует выделить все классы анализа, которые могут быть выделены, а затем определить те из них, которые задействованы в конкретных прецедентах.

Чтобы выделить классы анализа (для нескольких или всех прецедентов), необходимо определить:

• классы сущностей для моделирования хранимых данных на основе описаний прецедентов и моделей предметной области;

• по одному основному граничному классу на каждого актера-человека. Это будет прототип главной формы его пользовательского интерфейса. Основной граничный класс может быть набором простых граничных классов, соответствующих формам его пользовательского интерфейса;

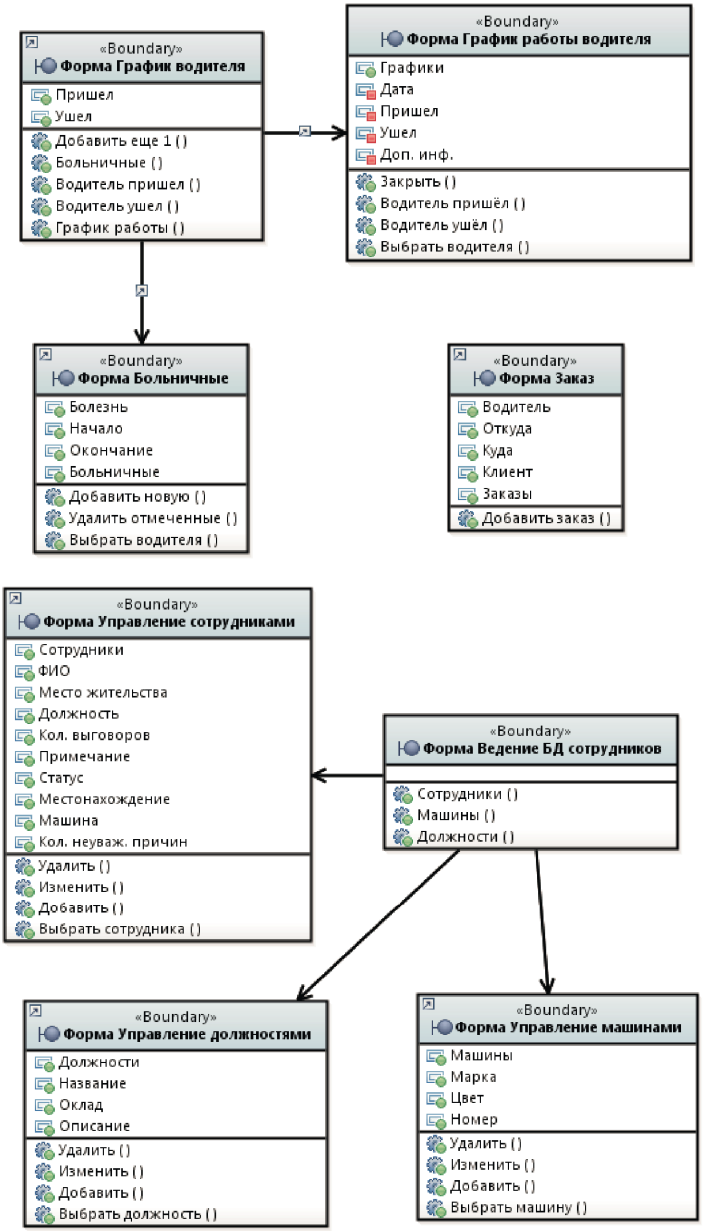


Рис. 3.3. Диаграмма граничных классов для АСУ таксопарком

• по одному граничному классу (простому) на каждую сущность. Это будут логические объекты для работы с сущностями через собственный интерфейс;

• по одному основному граничному классу на каждого актера (внешнюю систему). Это будет коммуникационный интерфейс. Если имеется несколько уровней коммуникации, то выделяют по одному граничному классу на каждый уровень;

• по одному управляющему классу на каждый прецедент (его кооперацию) для реализации бизнес-логики. В отдельных случаях может быть управляющий класс, включенный в граничный для очень простой бизнес-логики (например, для добавления, удаления или просмотра записей), или два и более управляющих классов для сложного алгоритма управления.

На рис. 3.3 приведена диаграмма граничных классов для АСУ таксопарком (в соответствии с прототипом пользовательского интерфейса), а на рис. 3.4 — диаграмма управляющих классов для АСУ таксопарком (по одному на прецедент).

После выявления классов анализа для каждого прецедента составляют диаграмму классов, на которую помещают:

• актера, инициирующего данный прецедент (из модели прецедентов);

• других актеров (при их наличии), взаимодействующих с прецедентом;

• граничные классы указанных актеров;

• управляющий класс данного прецедента;

• классы сущностей, используемые в данном прецеденте.

Затем указывают связи ассоциации между классами анализа, участвующими в анализируемой кооперации, имеющие следующие направления:

• от актеров к их граничным классам;

• от граничных классов к управляющему классу;

• от управляющего класса к классам сущностей.

Связи ассоциаций показывают взаимодействия между классами анализа и соответствуют будущим вызовам методов классов.

На рис. 3.5 приведена диаграмма классов участников для прецедента (кооперации) «Заказ такси».

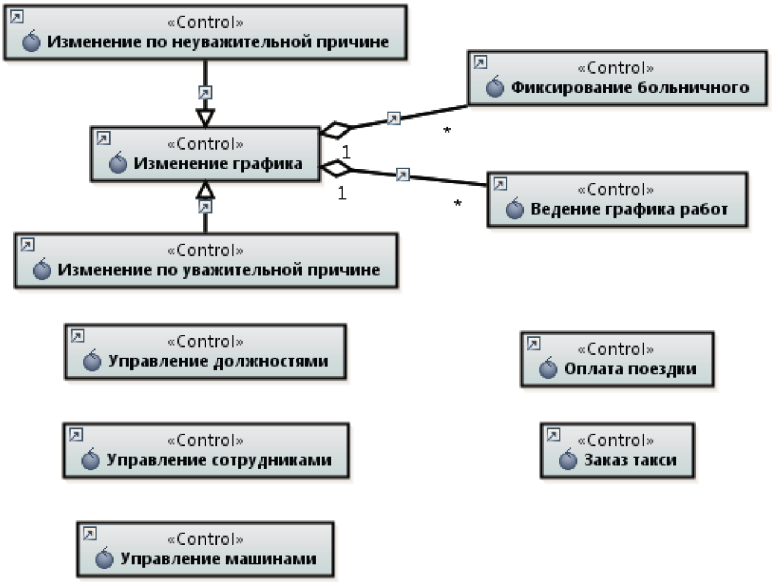


Рис. 3.4. Диаграмма управляющих классов для АСУ таксопарком

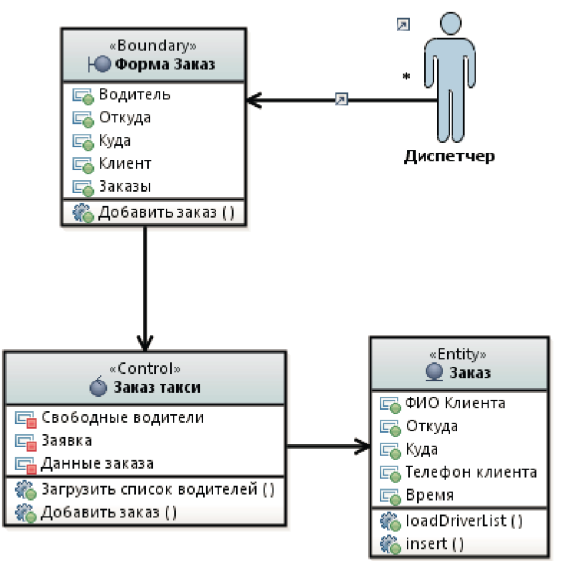


Рис. 3.5. Диаграмма классов участников для прецедента «Заказ такси»

3.2.2. *Описание взаимодействия объектов анализа*

Для описания алгоритма выполнения прецедента составляют диаграммы кооперaций — диаграммы взаимодействия или диаграммы последовательностей. Построение диаграммы кооперации для прецедента происходит на основе построенной диаграммы классов — участников прецедента. Если в прецеденте существует несколько возможных потоков управления, то составляют по одной диаграмме кооперации на каждый поток.

На диаграмму кооперации помещают объекты классов, участвующих в ней, и добавляют сообщения, которыми обмениваются объекты при реализации прецедента. Линии и направления сообщений соответствуют ассоциациям на диаграмме классов-участников.

Последовательность сообщений определяется порядком взаимодействия объектов. Начинается кооперация всегда сообщением от актера-инициатора. Сообщения указываются без сигнатур, только названием ответственности.

На рис. 3.6 приведена диаграмма последовательностей для кооперации «Заказ такси».

Все сообщения, которые являются входящими для класса анализа, будут трансформированы в его методы.

3.2.3. *Определение специальных требований*

Завершается анализ прецедента формированием специальных, в том числе нефункциональных, требований к нему, например, необходимым объемом передаваемых или отображаемых данных или количеством транзакций в минуту.

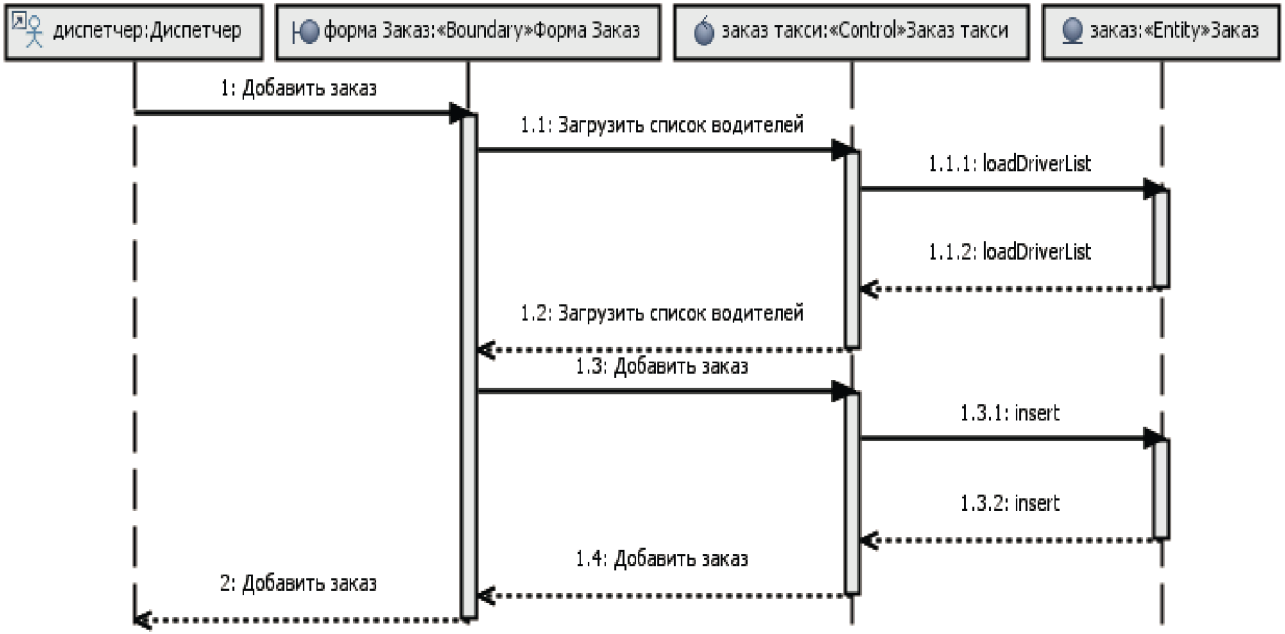


Рис. 3.6. Диаграмма последовательностей для кооперации «Заказ такси»

3.3. Анализ класса

После анализа прецедентов можно переходить к анализу классов. При анализе классов определяются и уточняются их атрибуты, методы и связи.

3.3.1. *Определение ответственности класса*

Ответственностью класса анализа является его функциональное назначение, которое определяется на основе комбинации всех ролей данного класса из тех кооперации, в которых он участвует.

Ответственность — это прообраз будущего метода класса. В отличие от метода она не имеет сигнатуры и не привязана к языку программирования. Набор ответственностей класса формируется объединением всех сообщений, которые получает данный класс во всех кооперациях, где он участвует. Таким образом обеспечивается обработка классом всех сообщений, которые ему отправляют другие классы.

В процессе объединения ответственностей класса из разных кооперации возможно дополнение ответственности или изменение ранее выделенных ответственностей. Это позволит избежать дублирования или расхождений.

3.3.2. *Определение атрибутов класса*

Классы анализа используют свои атрибуты для выполнения своих обязанностей. В общем случае набор атрибутов класса анализа зависит от его типа.

Атрибуты классов сущностей выделяют из описания предметной области. Они обычно просты и содержат полезные исходные данные.

Атрибуты граничных классов содержат свойства коммуникационных протоколов или поля для ввода данных.

Атрибуты управляющих классов малочисленны. Это могут быть вычисляемые или временные значения, используемые при выполнении кооперации.

Определение атрибутов класса проводят на основе ролей класса в кооперациях по следующим правилам:

• атрибутам класса назначают концептуальные типы;

• если система сложная и класс имеет много атрибутов, то часть атрибутов выносят в отдельный класс;

• общие атрибуты у нескольких объектов выделяют в отдельный класс.

Если класс содержит атрибуты, которые имеют собственные зависимости, то может быть составлена диаграмма частей класса, отражающая зависимости между его атрибутами.

3.3.3. *Определение ассоциаций и агрегаций*

После анализа отдельных классов составляется одна или более общих диаграмм классов, которые содержат все классы, выявленные при анализе, и связи между ними. Эти диаграммы формируются объединением диаграмм классов, построенных для отдельных кооперaций.

Связи между классами анализа соответствуют либо зависимостям, полученным для предметной области (для классов сущностей), либо линиям и направлениям передачи сообщений между классами в кооперациях. В обоих случаях классы связываются ассоциациями.

Ассоциация между классами показывает взаимодействие объектов соответствующих классов в кооперации. Возможно использование направленной ассоциации.

Связи ассоциации уточняются, и для каждой назначаются параметры: роли классов, видимость связей, арность, множественность, ограничение на количество связей и т. д.

Кроме ассоциации на диаграмме классов указывают связи агрегации:

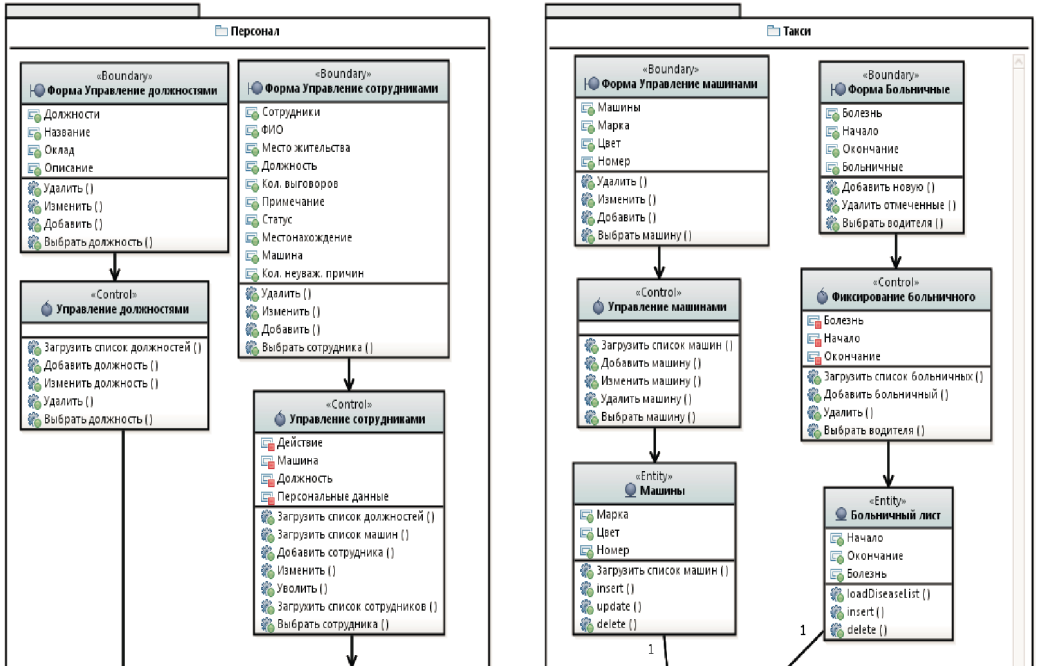
• композицию для физической вложенности объектов классов, например автомобиль и его детали;

• агрегацию для логической связи, например коллекция объектов, связи родителей и их детей.

Для связей агрегации также задают их параметры.

3.3.4. *Определение обобщений*

Для сокращения дублирования поведения или хранимых данных в разных классах используют обобщение. Чтобы выполнить это, разделяемое или общее поведение классов выносят в отдельный класс и определяют связь типа обобщения.


Персонал
«Boundary»
Форма Управление должностями
Должности
Название
Оклад
Описание
Удалить ()
Изменить ()
Добавить ()
Выбрать должность ()
«Control»
Управление должностями
Загрузить список должностей ()
Добавить должность ()
Изменить должность ()
Удалить ()
Выбрать должность ()
«Boundary»
Форма Управление сотрудниками
Сотрудники
ФИО
Место жительства
Должность
Кол. выговоров
Примечание
Статус
Местонахождение
Машина
Кол. неуваж. причин
Удалить ()
Изменить ()
Добавить ()
Выбрать сотрудника ()
«Control»
Управление сотрудниками
Действие
Машина
Должность
Персональные данные
Загрузить список должностей ()
Загрузить список машин ()
Добавить сотрудника ()
Изменить ()
Уволить ()
Загрузить список сотрудников ()
Выбрать сотрудника ()
Такси
«Boundary»
Форма Управление машинами
Машины
Марка
Цвет
Номер
Удалить ()
Изменить ()
Добавить ()
Выбрать машину ()
«Control»
Управление машинами
Загрузить список машин ()
Добавить машину ()
Изменить машину ()
Удалить машину ()
Выбрать машину ()
«Entity»
Машины
Марка
Цвет
Номер
Загрузить список машин ()
insert ()
update ()
delete ()
1
«Boundary»
Форма Больничные
Болезнь
Начало
Окончание
Больничные
Добавить новую ()
Удалить отмеченные ()
Выбрать водителя ()
«Control»
Фиксирование больничного
Болезнь
Начало
Окончание
Загрузить список больничных ()
Добавить больничный ()
Удалить ()
Выбрать водителя ()
«Entity»
Больничный лист
Начало
Окончание
Болезнь
loadDiseaseList ()
insert ()
delete ()
1

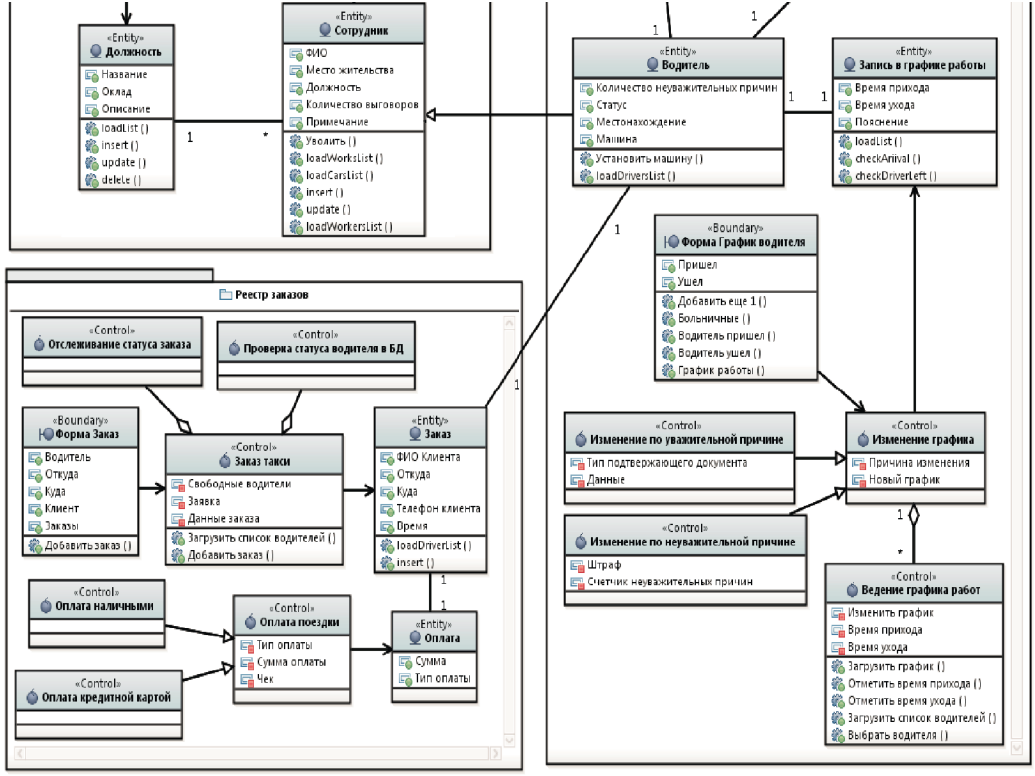


Рис. 3.7. Общий вид итоговой диаграммы классов анализа для АСУ таксопарком

3.3.5. *Определение специальных требований*

Завершается анализ класса формированием специальных и нефункциональных требований к нему.

При описании этих требований желательно сослаться на общие специальные требования, определенные при анализе архитектуры.

3.4. Анализ пакетов

После окончания анализа классов выполняется пересмотр распределения классов по пакетам с учетом их итогового состава и зависимостей. Эта деятельность называется анализом пакетов.

Целью анализа пакетов является достижение следующих итогов:

- независимости или малой зависимости пакета от других пакетов;
- реализации пакетом его прецедентов;
- содержания пакетом функционально ориентированных классов.

Для анализа пакетов необходимо найти их зависимости. Зависимость пакетов определяют на основе зависимостей входящих в них классов. Например, пакет А зависит от пакета Б, если класс в пакете А зависит от класса в пакете Б. Реально зависимости отражают трудозатраты на последующее внесение изменений, поэтому их желательно минимизировать.

На рис. 3.7 приведена итоговая диаграмма классов для АСУ таксопарком.

Для достижения поставленных целей возможно перемещение классов между пакетами. Следует помнить, что класс может находиться только в одном пакете, который, в свою очередь, может быть вложен в другой пакет.

Выводы

Рабочий процесс анализа переводит описания требований в описания классов и их взаимодействия.

В результате анализа выделяют классы и пакеты анализа и составляют модель анализа. Модель анализа содержит диаграммы классов, пакетов и диаграммы взаимодействия.

Процесс анализа состоит из следующих действий: анализ архитектуры системы; анализ прецедентов, классов и пакетов.

При анализе архитектуры системы выделяют пакеты прикладного уровня и сервисные пакеты и указывают их зависимости.

Анализируя прецеденты, определяют кооперации и составляют их модели. Статическая модель кооперации — это диаграмма классов, которые в этой кооперации участвуют. Динамическая модель — это диаграммы последовательностей (или взаимодействия), которые описывают алгоритм выполнения кооперации объектами ее классов.

Классы анализа бывают трех типов: граничный (интерфейс пользователя), управляющий (бизнес-логика) и класс сущности (хранимая информация).

В ходе анализа классов составляют их концептуальное описание.

При анализе пакетов проводят перераспределение классов анализа по пакетам с тем, чтобы снизить зависимость между пакетами.

Контрольные вопросы и задания

1. Объясните назначение рабочего процесса анализа требований.
2. Какие артефакты создаются в процессе выполнения рабочего процесса?
3. Перечислите действия, из которых состоит рабочий процесс.
4. Объясните, из чего состоит модель анализа, из каких элементов и диаграмм.
5. Расскажите, как выполняют анализ архитектуры.
6. Что такое пакет и сервисный пакет? Как их идентифицируют?
7. Назовите стереотипы классов анализа.
8. Что такое граничный класс и как его выявить? Каковы его атрибуты?
9. Что такое управляющий класс и как его выявить? Что такое класс сущности и как его выявить? Каковы атрибуты класса сущности?
10. Как выполняют анализ кооперaций? Что содержат диаграммы классов-участников и диаграммы их взаимодействия?
11. Как выполняют анализ класса? Откуда находят атрибуты и ответственности классов анализа, связи между ними?
12. Как выполняют анализ пакетов? В чем его цель?
13. Объясните, откуда появляются и что означают зависимости пакетов.

Глава 4. РАБОЧИЙ ПРОЦЕСС «ПРОЕКТИРОВАНИЕ»

Проектирование является важным и трудоемким процессом разработки ПО. В процессе проектирования формируют модели, представляющие разрабатываемую систему с разных точек зрения. Создают модели, описывающие структуру данных и алгоритмы программы, компоненты системы и связи между ними, сетевую конфигурацию.

Итогом проектирования является детальное и исчерпывающее описание системы и ее компонентов, которое на последующем этапе реализации будет преобразовано в программный код. В результате проектирования создаются следующие артефакты:

• модель проектирования, включающая в себя систему проектирования, которая содержит множество подсистем проектирования, классов проектирования, проектов кооперации и интерфейсов;

• классы проектирования — описания классов в терминах языка реализации;

• проекты кооперации — описания проектных представлений прецедентов, содержащих диаграммы классов, диаграммы взаимодействия и проекты потоков событий для выполнения прецедента;

• подсистемы проектирования и сервисные подсистемы — описания проектов будущих компонентов системы;

• интерфейсы, включающие в себя перечень доступных извне функций подсистем;

• описание архитектуры, которое содержит наиболее важные проектные решения, определяет компоненты и технологии, используемые при разработке и функционировании системы;

• модель развертывания — описание сетевой конфигурации.

Модель проектирования, описание архитектуры и модель развертывания составляет архитектор. Проекты классов, подсистем и интерфейсов прорабатывает инженер по прецедентам. За проекты кооперации отвечает инженер по компонентам.

Процесс проектирования состоит из выполнения следующих действий:

• проектирование архитектуры;
• проектирование прецедентов;
• проектирование классов;
• проектирование подсистем.

Рассмотрим эти действия более подробно.

4.1. Проектирование архитектуры

Процесс проектирования можно разделить на два крупных этапа: предварительное проектирование и детальное проектирование. На этапе предварительного проектирования определяют компоненты системы и связи между ними, выбирают используемые средства и технологии, т. е. формируют архитектуру системы. На этапе детального проектирования проводят декомпозицию компонентов системы на модули или классы и составляют их детальные проекты.

Проектирование архитектуры очень важно, поскольку принятые на этом этапе решения влияют на всю последующую разработку системы, и изменить их впоследствии будет очень трудозатратно. По этой причине прежде всего создают проект архитектуры системы.

4.1.1. *Определение узлов и сетевых конфигураций*

Проектирование архитектуры начинают с определения физических компонентов и связей между ними. На этом шаге формируют следующие описания системы:

• перечень узлов и их конфигураций (центрального процесса, оперативной памяти и т. д.);

• конфигурации сети с указанием каналов связи между узлами, используемых протоколов и характеристик передачи, в том числе скорости и качества передачи;

• дополнительные требования, включая требования к безопасности, достоверности, резервному копированию и т. д.

Возможно определение отдельных конфигураций для тестирования или моделирования ПО.

По итогам определения узлов и сетевых конфигураций составляют диаграмму развертывания.

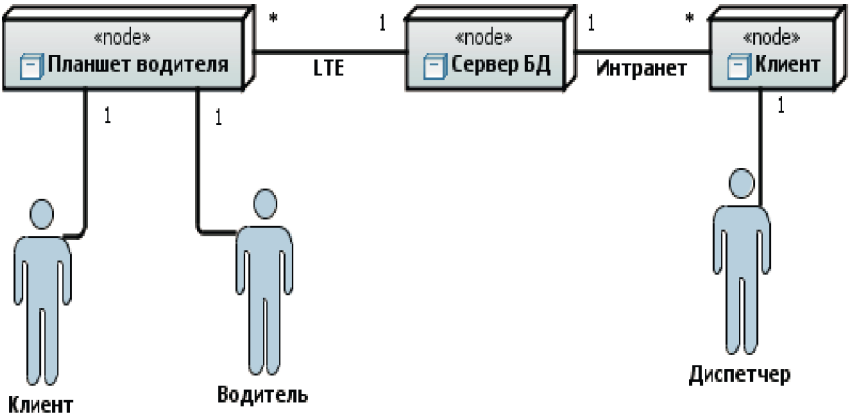


Рис. 4.1. Диаграмма развертывания для АСУ таксопарком

На рис. 4.1 приведена диаграмма развертывания для АСУ таксопарком.

4.1.2. *Определение подсистем и их интерфейсов*

Следующим шагом проектирования архитектуры является определение подсистем и их интерфейсов.

Подсистема проектирования — это прообраз будущего программного компонента системы. Она предоставляет способ группировки элементов системы на этапе проектирования в более крупные блоки.

Подсистема проектирования содержит классы проектирования, кооперации, интерфейсы и вложенные подсистемы. При выделении подсистем следует учитывать следующее:

• разработка подсистем может быть параллельной;

• подсистемы имеют сильную связность внутри себя и слабое сцепление между подсистемами;

• подсистема является крупным блоком системы и будет транслирована в исполняемые файлы или библиотеки;

• подсистемы обеспечивают связь с повторно используемыми компонентами;

• подсистемы выполняют обертку унаследованных подсистем.

В приложении выделяют четыре уровня подсистем: специфический и общий прикладной уровни, средний уровень и уровень системного ПО.

4.1.3. *Определение прикладных подсистем*

Подсистемы прикладного уровня определяют на основе пакетов анализа. Изначально пакеты анализа транслируются в подсистемы проектирования прикладных уровней, сервисные пакеты — в сервисные подсистемы.

Далее проводят уточнение и декомпозицию полученных подсистем по следующим правилам:

- часть пакета анализа, совместно используемую несколькими подсистемами, выделяют в отдельную подсистему;
- часть пакета анализа для многократного применения выделяют в отдельную подсистему, в том числе подсистему среднего уровня или уровня системного ПО;
- унаследованные подсистемы выделяют в отдельные подсистемы;
- выполняют декомпозицию подсистем для возможности разделения работ;
- осуществляют декомпозицию подсистем в целях разнесения их по узлам так, чтобы каждая подсистема была на отдельном узле.

4.1.4. *Определение подсистем среднего уровня и системного программного обеспечения*

К подсистемам среднего уровня и уровня системного ПО относят следующие компоненты:

- операционную систему;
- СУБД;
- средства коммуникации;
- средства распределенной обработки;
- средства выполнения транзакций;
- средства разработки интерфейсов.

Поскольку часть из перечисленных компонентов имеет смысл покупать, а не разрабатывать самостоятельно, то при проектировании подсистем среднего уровня и уровня системного ПО осуществляют:

- подбор и интеграцию покупаемых и создаваемых компонентов;
- оценку эффективности и пригодности выбранных компонентов.

Критерием эффективности выбора является уменьшение зависимости от продуктов сторонних производителей и тем самым снижение рисков несовместимости или отсутствия поддержки.

Поэтому каждый продукт рассматривают как подсистему с явным интерфейсом, что дает возможность последующей замены на аналог. Возможно вынесение стандартных типов и используемых компонентов в отдельные подсистемы.

4.1.5. *Определение зависимостей между подсистемами*

Зависимости подсистем соответствуют зависимостям классов подсистемы или пакетов анализа. Зависимости подсистем проектирования, как и зависимости пакетов анализа соответствуют вызовам при работе системы. Подсистемы верхних уровней зависят от подсистем нижних уровней, но не наоборот.

На рис. 4.2 приведены уровни подсистем проектирования для АСУ таксопарком и их зависимости.

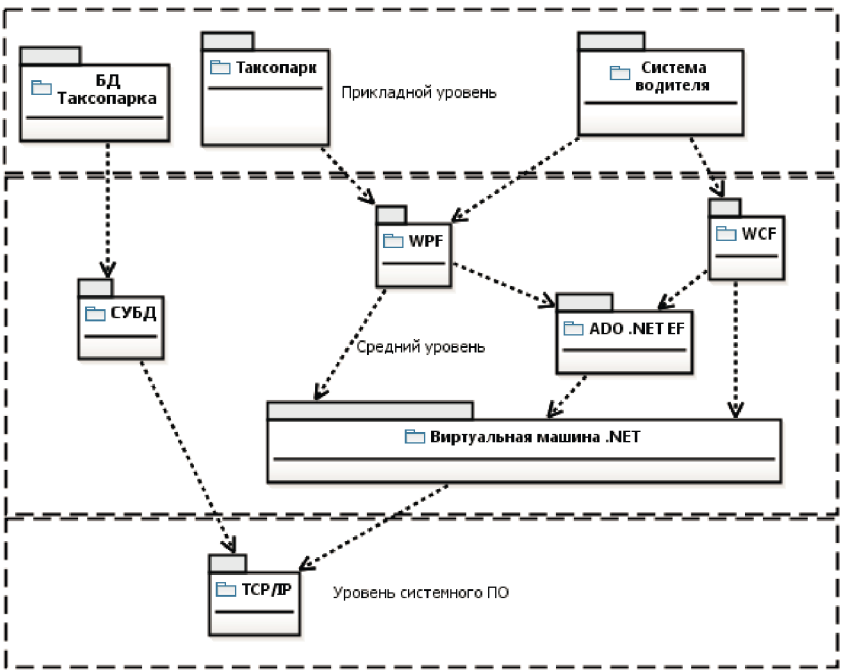


Рис. 4.2. Уровни подсистем проектирования для АСУ таксопарком

Для уменьшения зависимости подсистем проводят замену зависимости подсистем на зависимость их интерфейсов. Это позволит скрыть реализацию подсистемы от зависящих от нее компонентов.

4.1.6. *Определение интерфейсов подсистем*

Для каждой подсистемы следует определить ее интерфейс. Интерфейс подсистемы — это перечень операций, выполняемых данной подсистемой и доступных извне. В интерфейсе указывают полные сигнатуры операций.

Интерфейс подсистемы формируется на основе выявленных зависимостей подсистем и определенных ранее классов в пакете анализа. Если подсистема транслирована из пакета анализа, содержащего класс, к которому имеется обращение извне, то интерфейс подсистемы должен обеспечивать внешнюю ответственность этого класса.

Определение интерфейсов начинают с подсистем верхнего уровня. Для подсистем среднего и нижнего уровня интерфейсы могут быть известны заранее, если это существующее ПО.

Интерфейсы влияют на зависимости подсистем, поскольку скрывают их реализацию.

4.1.7. *Определение архитектурно значимых классов проектирования*

Кроме подсистем в описание архитектуры включают описания некоторых классов, имеющих значение для работы всей системы или нескольких ее компонентов. Такие классы называют архитектурно значимыми.

К архитектурно значимым классам проектирования относят классы, определенные на основе классов анализа, и активные классы. Классы проектирования, определенные на основе классов анализа — это прежде всего классы сущностей. Активные классы — классы, объекты которых имеют свою нить управления или свой процесс.

Правила определения активных классов:

- не менее одного класса на каждый узел;
- один класс — на обмен между узлами;

• не менее одного класса — для увеличения производительности, например, для ответа без задержки;

• не менее одного класса — для запуска, остановки, конфигурирования, снятия блокировки, восстановления и т. д.

Привязку активных объектов к узлам проводят с учетом их возможности и конфигурации.

Если подсистема размещается на узле, то активный класс размещается в подсистеме. На следующем этапе реализации активный класс будет транслирован в исполняемый файл, а подсистема — в компонент.

4.1.8. *Определение обобщенных механизмов проектирования*

Проектирование архитектуры завершается определением обобщенных механизмов проектирования. Обобщенные механизмы проектирования — это технологии и средства, которые обеспечивают следующие возможности:

• длительное хранение данных;

• распределенную обработку;

• средства безопасности;

• контроль и исправление ошибок;

• управление транзакциями.

При проектировании архитектуры необходимо сформулировать конкретные требования к системе, а также выбрать технологии и средства для их обеспечения. Например, для длительного хранения можно использовать реляционные или объектные БД, или файлы; для распределенной обработки — технологии java.rmi, corba или com.

Поскольку каждая технология имеет свои преимущества и недостатки, проводится сравнительный анализ с учетом возможного развития системы. Обычно поставляемое и приобретаемое ПО относится к среднему уровню и уровню системного ПО, а создаваемое ПО — к прикладному уровню.

Для рассматриваемого примера АСУ таксопарком в качестве системы построения приложения принята технология WPF, так как она обеспечивает простоту и скорость разработки, а также обладает высоким быстродействием и, что немаловажно, подходит под шаблон проектирования MVVM. Средством коммуникации выбрана WCF, поскольку она имеет такие преимущества, как высо-

кая надежность и безопасность. Для доступа к БД принята технология ADO. NET Entity Framework, так как она предоставляет возможность работы с объектами с помощью LINQ, а также легко интегрируется с WPF и вписывается в MVVM.

К обобщенным механизмам проектирования относят применение обобщенных кооперации — паттернов, типовых проектных решений на основе абстрактных кооперации. Это позволяет сократить время на разработку и повысить качество проекта.

Для АСУ таксопарком в качестве паттерна выбран паттерн «Одиночка», его детальное рассмотрение будет приведено далее.

4.2. Проектирование прецедентов

После проектирования архитектуры системы в целом и принятия проектных решений, имеющих значение для всей системы (в том числе после выделения подсистем, их интерфейсов и определения используемых технологий и компонентов), можно перейти к проектированию отдельных фрагментов системы: прецедентов, классов и подсистем.

При проектировании прецедента определяют классы и подсистемы, необходимые для реализации этого прецедента, а также описывают их взаимодействие.

4.2.1. *Определение участвующих классов*

Проектирование прецедента как прикладной функции системы начинают с выявления классов проектирования, которые участвуют в его реализации. Классы проектирования выделяют на основе классов анализа, участвующих в кооперации рассматриваемого прецедента.

Классы анализа транслируются в классы проектирования. Если есть специальные требования к кооперации, то создается новый класс проектирования для их реализации. Если требуемых классов проектирования нет, их надо определить.

Строится диаграмма классов, содержащая классы проектирования, участвующие в прецеденте, и связи между ними. Правила построения диаграммы классов и определения связей между классами на этапе проектирования те же, что и на этапе анализа. Поскольку классы проектирования соответствуют классам анализа

с сохранением их ответственностей, связи между классами проектирования большей частью транслируются из моделей анализа.

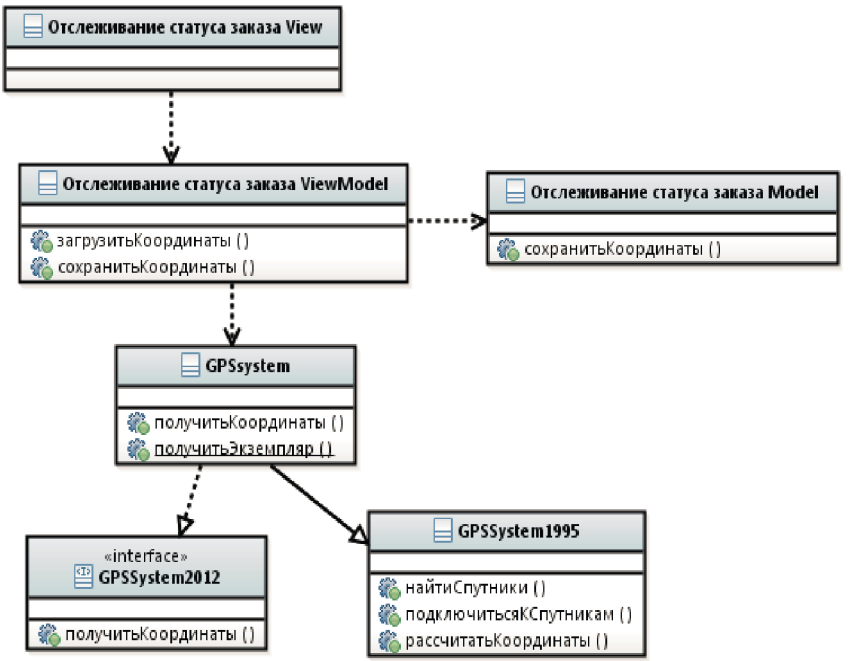


Рис. 4.3. Диаграмма классов участников для кооперации «Отслеживание статуса заказа»

На рис. 4.3 приведена диаграмма классов участников для кооперации «Отслеживание статуса заказа». Классы проектирования выбраны на основе классов анализа (граничный, управляющий и сущности), проектного решения об использовании структурного шаблона MVVM, а также паттерна «Одиночка» для реализации единственного класса обращения к GPS.

4.2.2. *Описание взаимодействия объектов проектирования*

Для описания алгоритма выполнения прецедента составляют диаграммы последовательностей, по одной на каждый поток событий прецедента (возможную последовательность действий).

Диаграммы последовательностей содержат объекты классов, участвующих в прецеденте, и поток событий между ними.

На отдельные диаграммы выносят общие последовательности действий или действия для повторного использования.

Правила построения диаграмм последовательностей при проектировании прецедента:

• начальное событие всегда инициируется актером — инициатором прецедента;

• для каждого класса, участвующего в прецеденте, на диаграмму последовательностей помещают его объект или несколько объектов;

• направления сообщений, которыми обмениваются объекты, соответствуют связям между их классами на диаграмме классов и означают передачу управления в ходе выполнения прецедента (вызов методов класса);

• поток управления не может прерываться, в ответ на каждое входное сообщение объект отправляет одно или более сообщений другим объектам или возвращает результат его обработки отправителю;

• при необходимости помещают ссылки на обобщенные диаграммы взаимодействия.

При построении диаграмм последовательностей сообщениям дают вре́менные имена, которые позднее будут заменены на названия методов классов. На диаграммах указывают метки для альтернативных вариантов алгоритма, например: «тайм-аут соединения», «неверный ввод», «сообщение об ошибке от внешнего компонента».

На рис. 4.4 приведена диаграмма последовательностей для кооперации «Отслеживание статуса заказа».

4.2.3. *Определение участвующих подсистем и интерфейсов*

Кроме классов, участвующих в прецеденте, необходимо определить подсистемы, участвующие в нем. Это, с одной стороны, позволяет выполнять проектирование «сверху вниз», с другой стороны — при необходимости заменить подсистему на аналог.

Подсистемы, участвующие в прецеденте, — это подсистемы, которые:

• были транслированы из пакетов анализа, включающих классы анализа, участвующие в данном прецеденте;

• содержат классы проектирования, участвующие в данном прецеденте;

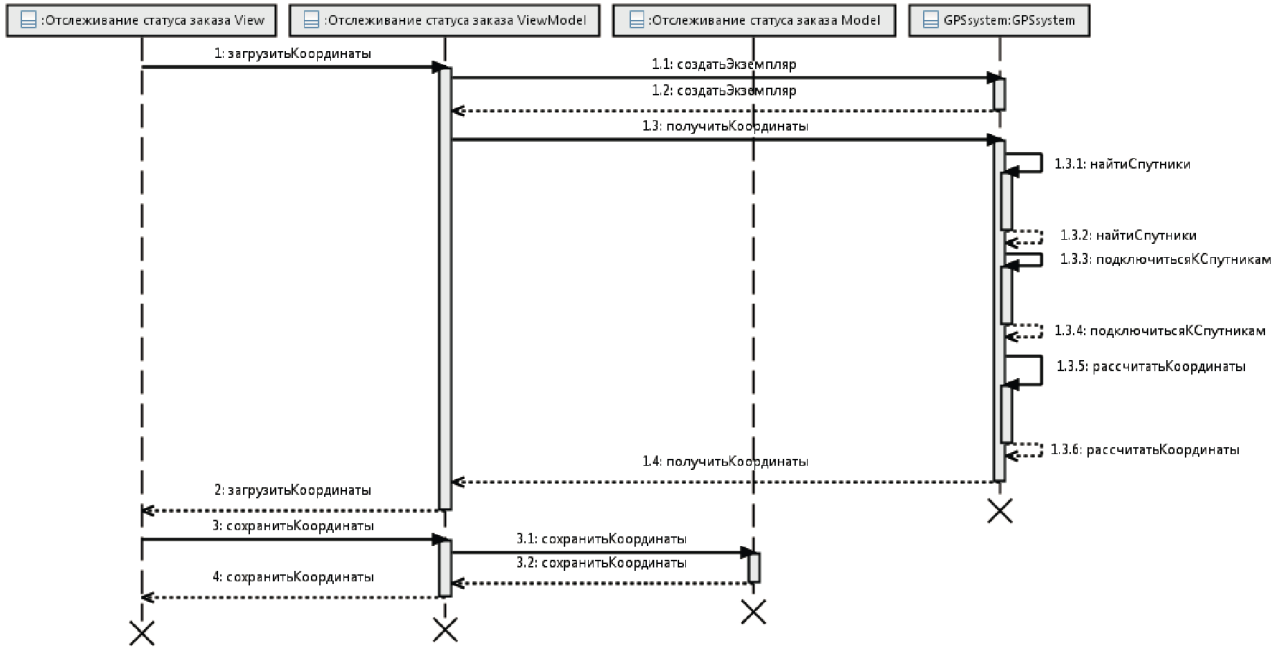


Рис. 4.4. Диаграмма последовательностей для кооперации «Отслеживание статуса заказа»

• включают классы проектирования, полученные для реализации специальных требований к данному прецеденту.

Составляют диаграмму классов, на которую помещают подсистемы, участвующие в прецеденте, их интерфейсы и связи. Связи между подсистемами показывают передачу сообщений, которыми они обмениваются в процессе выполнения прецедента. Интерфейсы подсистем содержат названия операций, которые при этом вызываются.

4.2.4. *Описание взаимодействия подсистем*

Кроме диаграммы классов, составляют одну или более диаграмм последовательностей, которые показывают выполнение прецедента на уровне подсистем. Эти диаграммы демонстрируют взаимодействие подсистем при реализации прецедента. Их получают на основе диаграмм последовательностей для классов путем объединения классов в подсистемы.

Диаграммы последовательностей содержат сообщения между актерами и подсистемами. Линии жизни указывают для подсистем, для каждой подсистемы не менее одной линии жизни. Сообщения отображаются в терминах операций интерфейсов с учетом того, что у подсистемы может быть больше одного интерфейса.

4.2.5. *Определение требований к кооперации*

Завершается проектирование прецедента определением специальных и нефункциональных требований к нему. Здесь указывают требования, которые необходимо реализовать для данного прецедента.

4.3. Проектирование класса

Следующим шагом, выполняемым при проектировании, является проектирование класса. Составляют подробное описание класса в терминах выбранного языка программирования.

В результате проектирования класса будут определены следующие его параметры:

• операции (внешний интерфейс);

• атрибуты;

• отношения;
• методы (внутренняя реализация);
• состояния;
• зависимость от обобщенных механизмов проектирования;
• требования к реализации;
• правила реализации всех интерфейсов класса.

4.3.1. *Описание класса проектирования*

Классы проектирования определяют на основе классов анализа. В большинстве случаев применяют транслирование класса анализа в один или несколько классов проектирования с учетом особенностей среды разработки.

Граничные классы в зависимости от выбранной технологии разработки транслируют в следующие классы проектирования:

• классы для создания интерфейса взаимодействия с внешним устройством или системой;

• классы, соответствующие элементам пользовательского интерфейса, например: экранной форме, полю ввода или элементу управления с указанием их стереотипов.

Если применялась среда визуальной разработки, то трансляция проходит автоматически, на основе созданного ранее прототипа пользовательского интерфейса.

Классы сущности в зависимости от выбранной СУБД и технологии работы с БД транслируются в реляционные таблицы или объектные и объектно-реляционные сущности. Возможно применение объектно-реляционного отображения. Часто такая трансляция выполняется автоматически.

Управляющие классы транслируются в классы проектирования с учетом следующих факторов:

• размещение — по одному классу на каждый узел;

• производительность — совмещение управляющего класса с граничным классом или классом сущности в целях увеличения скорости работы;

• транзакции — использование существующих технологий управления транзакциями в качестве дополнительных управляющих классов.

На рис. 4.5 приведена схема БД, построенная для АСУ таксопарком на основе классов сущностей.

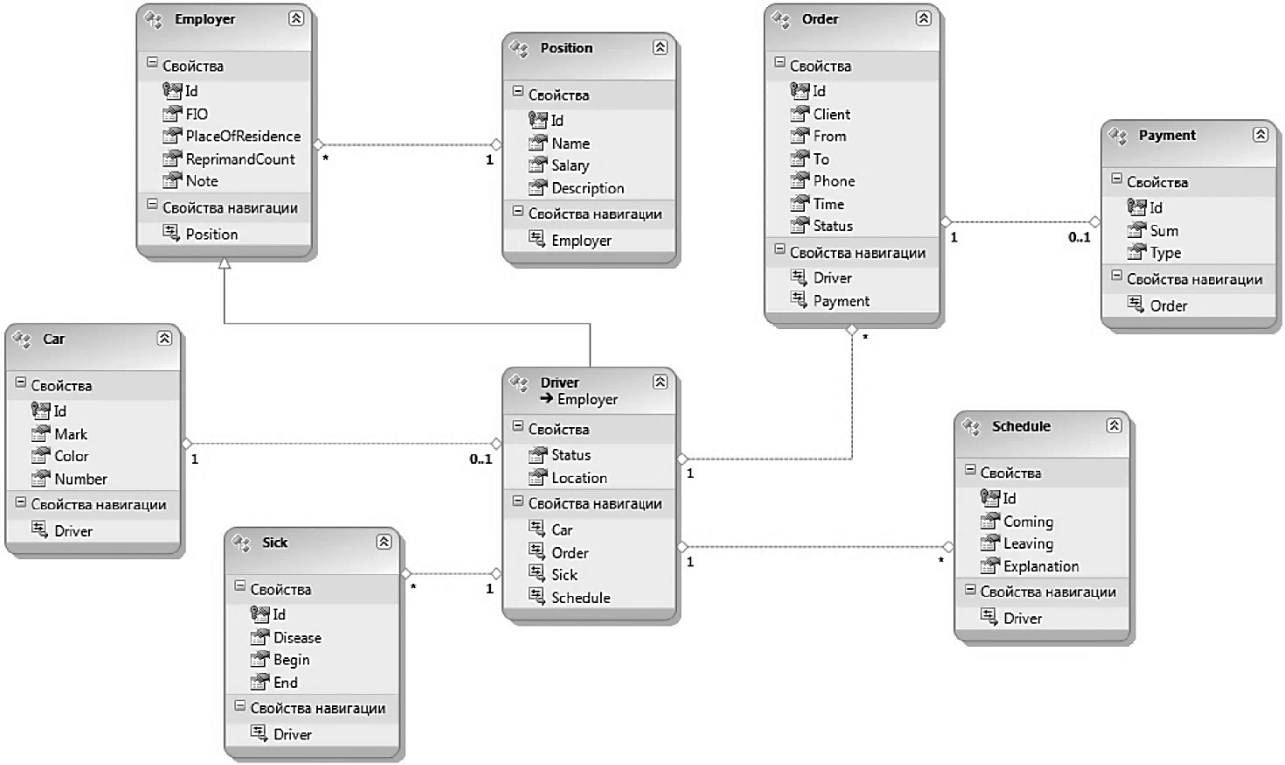


Рис. 4.5. Схема БД для АСУ таксопарком

4.3.2. *Определение операций классов*

Операции классов проектирования определяют на основе описаний классов анализа и их зависимостей по следующим правилам:

• сигнатуры операций записывают с учетом синтаксиса языка программирования;

• указывают область видимости операции;

• для каждой зависимости класса определяют одну или более операций;

• на основе описания ответственности формируют список параметров операции и указывают возвращаемое значение;

• специальные требования должны быть учтены.

В итоге определения операций составляется интерфейс для его последующей реализации в классе.

4.3.3. *Определение атрибутов класса*

Атрибуты классов проектирования также определяют на основе описаний классов анализа и их зависимостей по следующим правилам:

• типы и параметры атрибутов указывают с учетом синтаксиса языка программирования и поддерживаемых типов данных;

• атрибуты являются свойством класса и участвуют в операциях классов;

• на основе каждого атрибута класса анализа определяют один или более атрибутов класса проектирования;

• если несколько классов имеют общий атрибут, то его выделяют в отдельный класс.

При сложной структуре атрибутов класса проводят декомпозицию классов с целью их упрощения. Для лучшего понимания зависимостей между атрибутами класса возможно построение диаграммы классов для его атрибутов.

4.3.4. *Определение ассоциаций и агрегаций*

При проектировании класса следует уточнить его связи с другими классами и параметры этих связей.

Изначально все ассоциации между классами анализа транслируются в ассоциации между соответствующими классами проекти-

рования. Далее на основе диаграмм последовательностей, построенных при проектировании прецедентов, корректируется и пополняется набор связей между классами типа ассоциаций и агрегаций.

Если на диаграмме последовательностей объект одного класса отправляет сообщение объекту другого класса, то между классами указывается ассоциация или агрегация. Отправка сообщения соответствует вызову метода класса. Агрегация означает, что вызывающий класс (его объект) имеет ссылку на объект вызываемого класса и через эту ссылку вызывает его метод. Ассоциация означает передачу сообщения или вызов метода. При проектировании ассоциаций и агрегаций определяют тип ссылки: глобальная или локальная переменная, параметр метода или указатель на самого себя. В зависимости от выбранного типа связь помечают стереотипом.

После корректировки набора связей уточняют их роли, арность, множественность, классы ассоциации в соответствии с языком программирования. Например, роль транслируется в имя атрибута, хранящего ссылку на связанный класс, класс ассоциации — в новый класс, реализующий связь типа «многие ко многим».

4.3.5. *Определение обобщений*

После проектирования атрибутов и операций классов и связей между ними определяют связи типа обобщений между классами. Обобщения либо транслируются из модели анализа, либо определяются в целях вынесения общего поведения или данных классов в базовый класс.

Обобщения проектируют в соответствии с языком программирования. Обычно обобщение транслируется в наследование. При отсутствии в языке конструкций для задания наследования возможна замена обобщения на ассоциацию.

4.3.6. *Описание методов*

Если операции класса задают его внешний интерфейс, то методы определяют внутреннюю реализацию поведения классов. Они реализуют операции класса.

Описание метода содержит его алгоритм на естественном языке или псевдокоде. Для тривиальных методов составлять описание необязательно.

4.3.7. *Описание состояний*

Если класс имеет сложное поведение, зависящее от его состояния, то составляют диаграмму состояний. На ней отображают, каким образом состояние объекта определяет его поведение при получении сообщения.

4.3.8. *Учет специальных требований*

Завершают проектирование класса определением специальных и нефункциональных требований к нему, в том числе с применением обобщенных механизмов проектирования, например наследования от стандартных классов.

4.4. Проектирование подсистем

После проектирования отдельных классов выполняют итоговое (уточняющее) проектирование подсистем. Классы распределяют или перераспределяют по подсистемам, чтобы достичь:

- сильной связности классов внутри подсистемы;
- слабого сцепления подсистемами;
- обеспечения классами подсистемы выполнения ее задач;
- наличия в подсистеме правильного интерфейса.

Для достижения перечисленных целей необходимо проверить сохранение:

- зависимостей подсистем;
- представляемых подсистемой интерфейсов;
- содержимого подсистемы.

Сохранение зависимостей подсистем означает, что зависимость подсистем происходит от зависимости их элементов (классов или вложенных подсистем). Если существует связь между элементами разных подсистем, то должна присутствовать связь между самими подсистемами. При этом связь между интерфейсами предпочтительнее, чем связь между подсистемами, поскольку позволяет скрыть внутреннюю реализацию.

Сохранение представляемых подсистемой интерфейсов означает, что интерфейс подсистемы складывается из тех ролей, которые она выполняет в прецедентах. Операции интерфейса подсистемы должны соответствовать всем ролям подсистемы и включать в себя

все сообщения (операции), получаемые подсистемой (и ее элементами) извне.

Сохранение содержимого подсистемы означает, что функционал подсистемы определяется поведением ее классов. Для каждого интерфейса подсистемы должен быть класс, который его реализует. Для каждого интерфейса каждого прецедента, в котором участвует подсистема, должна быть определена и описана кооперация в терминах элементов этой подсистемы для реализации указанного прецедента.

Выводы

Рабочий процесс проектирования преобразует концептуальную модель анализа в детальное и исчерпывающее описание системы и ее компонентов.

В результате проектирования создаются описания подсистем и классов проектирования, архитектуры системы, проектов кооперации и интерфейсов, составляются модели проектирования и развертывания.

Процесс проектирования состоит из проектирования архитектуры, прецедентов, классов и подсистем.

При проектировании архитектуры определяют сетевую конфигурацию, выделяют подсистемы, их интерфейсы и зависимости, выбирают технологии и средства разработки.

В процессе проектирования прецедентов выявляют классы и подсистемы, участвующие в них, и описывают их взаимодействие.

При проектировании классов составляют описания классов в терминах выбранного языка программирования.

В ходе проектирования подсистем уточняют их состав и интерфейсы, а также полноту описаний.

Контрольные вопросы и задания

1. Объясните назначение рабочего процесса проектирования.
2. Перечислите артефакты, которые создаются в процессе выполнения рабочего процесса.
3. Укажите действия, из которых состоит рабочий процесс.
4. Расскажите, что содержит модель проектирования.

5. Что такое архитектура системы? Что входит в ее описание?

6. Как выполняют проектирование архитектуры?

7. Что содержит модель развертывания и как ее составляют?

8. Как определяют подсистемы проектирования и сервисные подсистемы? Назовите уровни подсистем и охарактеризуйте назначение подсистем каждого уровня.

9. Что такое интерфейс? Как формируют интерфейс подсистемы? Кто реализует интерфейс подсистемы?

10. Что такое активный класс? Каковы правила его выявления?

11. Что такое обобщенные механизмы проектирования? Как их определяют?

12. Для чего применяют паттерны и шаблоны проектирования?

13. Как выполняют проектирование прецедентов? Как составляют диаграммы классов-участников, диаграммы участвующих подсистем и диаграммы их взаимодействия?

14. Каким образом пакеты и классы анализа транслируются в подсистемы и классы проектирования? На что влияет стереотип класса анализа?

15. Как выполняют проектирование класса? Что будет его результатом?

16. Объясните, как определяют и описывают атрибуты, операции, методы и связи классов.

17. В чем заключается цель проектирования подсистем? Как определяют зависимости подсистем?

18. Что такое специальные требования и как их учитывают на этапе проектирования?

Глава 5. РАБОЧИЙ ПРОЦЕСС «РЕАЛИЗАЦИЯ»

Итогом проектирования является множество моделей, подлежащих дальнейшему их преобразованию в работающую программную систему. Во время реализации создается программный продукт, готовый к использованию. При этом модели, построенные при проектировании, транслируются следующим образом:

• подсистема проектирования — в подсистему реализации, содержащую исполняемые файлы, исходные тексты и т. д.;

• сервисная подсистема — в сервисную подсистему;

• класс проектирования — в один или более (это зависит от языка программирования) файлов компонентов, содержащих исходный текст программы;

• активный класс — в исполняемый компонент;

• один или более проектов кооперации — в билд (сборку);

• модель развертывания и конфигурация сети — в распределенную систему и развертывание исполняемых компонентов.

Процесс реализации достаточно тривиален, и существенная его часть успешно автоматизирована.

В результате реализации создаются следующие артефакты:

• модель реализации — система реализации, содержащая множество подсистем, компонентов и интерфейсов реализации;

• описание архитектуры — перечень наиболее важных компонентов системы и их размещение по узлам;

• модель размещения — описание сетевой конфигурации;

• план сборки — перечень прецедентов, которые необходимо реализовать на очередном шаге разработки, и перечень компонентов, которые будут при этом созданы;

• компоненты — физические компоненты программной системы, обеспечивающие ее работу, например: исполняемые программы, базы данных, файлы настроек и т. д.;

• подсистемы реализации — контейнеры для компонентов, чья конкретная реализация зависит от используемых технологий;

• интерфейсы — перечень доступных извне функций компонентов.

Модель реализации, описание архитектуры и модель размещения составляет архитектор. План сборки — системный интегратор. За реализацию компонентов, подсистем и интерфейсов отвечает инженер по компонентам.

Процесс реализации состоит из выполнения следующих действий:

• реализация архитектуры;
• сборка системы;
• реализация подсистем;
• реализация классов;
• тестирование модулей.

Рассмотрим эти действия более подробно.

5.1. Реализация архитектуры

Процесс реализации, как и прочие рабочие процессы, начинают с проработки архитектуры системы.

Сначала определяют архитектурно значимые компоненты, обеспечивающие выполнение основного функционала системы, которые должны быть реализованы в первую очередь.

Компонент — это физический контейнер для элементов системы, в том числе классов. Их реализация зависит от языка программирования. Компоненты могут иметь следующие стереотипы:

«process» — исполняемый файл;

«message» — исходные или выходные данные;

«service» — сервис;

«entity» — таблица БД;

«specification» — данные для обработки в определенном формате.

Архитектурно значимые компоненты реализации транслируют из архитектурно значимых подсистем и классов проектирования.

Далее проводят определение исполняемых компонентов как наиболее значимых элементов системы и осуществляют распределение компонентов по узлам.

Исполняемые компоненты выявляют на основе моделей проектирования, прежде всего активных классов. Каждый активный

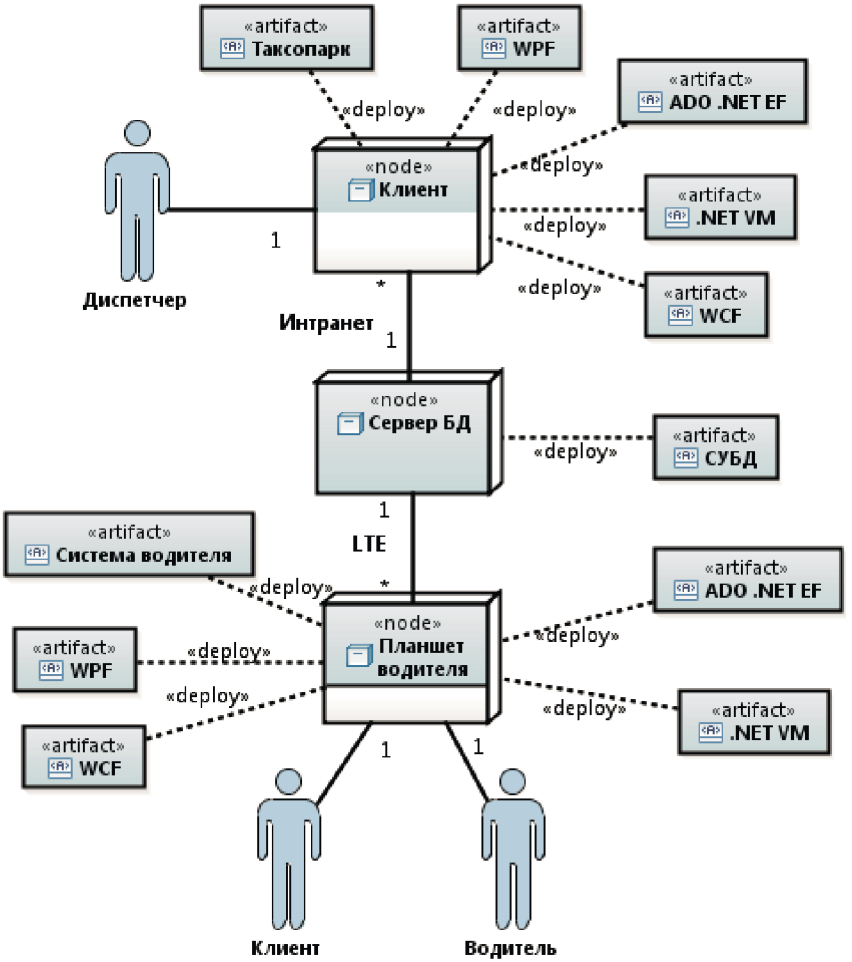


Рис. 5.1. Размещение компонентов по узлам для АСУ таксопарком

класс транслируется в исполняемый компонент (утилиту, приложение или сервис) и располагается в соответствующем узле.

На рис. 5.1 приведены компоненты реализации АСУ таксопарком и их размещение по узлам.

5.2. Сборка системы

По методологии унифицированного процесса разработка ПО выполняется итерационно. Каждая итерация дает функциональное приращение программного продукта. Итерация состоит из нескольких билдов.

Билд — это реализация нескольких прикладных функций системы, обычно одного или более прецедентов, а также дополнительных требований к ним. В результате создания билда получают несколько компонентов и/или подсистем, которые реализуют этот функционал.

До начала итерации определяют набор и состав ее билдов. Поскольку разработка ведется несколькими командами разработчиков, то в пределах итерации реализация билдов может осуществляться последовательно и/или параллельно.

При реализации выполняют планирование следующего билда и сборку текущего билда.

5.2.1. *Планирование следующего билда*

Выбор прецедентов, которые должны быть реализованы в следующем билде, осуществляют на основе следующих соображений:

- билд реализует набор связанных прецедентов или сценариев;
- билд должен дать инкремент функциональности;
- за один билд следует реализовывать не слишком много новых компонентов, возможно использование заглушек;
- программный проект развивается вверх и в сторону по иерархии подсистем, начало идет с нижнего уровня.

Если билд удовлетворяет перечисленным требованиям, его называют правильным.

После отбора прецедентов для следующего билда его планирование выполняют по следующей схеме:

1) определить проекты выбранных прецедентов (их кооперации);

2) определить классы и подсистемы проектирования, входящие в эти прецеденты;

3) определить подсистемы и компоненты реализации, соответствующие этим классам и подсистемам;

4) определить действия по их реализации и проверить реализуемость билда и его соответствие требованиям к правильному билду;

5) если проверка прошла успешно, перейти к реализации, последующему тестированию и интеграции;

6) иначе — внести корректировку в план билда.

После реализации всех компонентов, входящих в билд, осуществляют его сборку.

5.2.2. *Сборка билда*

Сборка билда — это интеграция всех его компонентов с целью получить работающую программную систему.

При сборке билда проводят подбор правильных версий подсистем и компонентов, а затем компиляцию компонентов снизу вверх.

Перед проведением сборки билда необходимо реализовать и протестировать все компоненты и подсистемы, входящие в него.

5.3. Реализация подсистемы

Подсистема реализации — это контейнер физических компонентов ПО, конкретная структура которого зависит от языка и технологии разработки. Например, подсистема — это проект при разработке на языке Visual C++ или пакет при разработке на языке Java.

При реализации подсистемы осуществляют работу с ее содержимым. При этом необходимо обеспечить правильную трансляцию подсистемы проектирования в подсистему реализации, провести сравнение подсистемы проектирования и ее реализации, а также выполнить следующие правила:

• каждый класс проектирования, входящий в подсистему проектирования, имеет реализацию в качестве компонента подсистемы реализации;

• каждый интерфейс подсистемы проектирования должен быть представлен как интерфейс компонента подсистемы реализации.

Подсистема реализации содержит компоненты, реализующие классы, полученные во время проектирования.

5.4. Реализация класса

Реализация класса — это перевод проекта класса в исходный код программы. Реализация класса состоит из выполнения следующих действий:

• описание файловых компонентов;
• генерация кода;
• реализация операций;
• создание компонентов.

Описание файловых компонентов — это оформление будущего компонента, в который преобразуется класс проектирования в терминах языка программирования. Описание файловых компонентов зависит от языка программирования и стереотипа класса.

Генерация кода для класса проектирования позволяет сформировать команды для его создания. Обычно это действие выполняется автоматически средствами разработки. В результате будет получена конструкция (одна или несколько команд на языке программирования) для создания класса с указанием сигнатур операций и его атрибутов или команды для создания таблицы БД, если класс транслируется в таблицу.

При генерации кода учитывают параметры класса проектирования и возможности языка программирования, например, однонаправленная ассоциации преобразуется в ссылку, а множественная связь — в коллекцию ссылок; роль связи будет преобразована в имя атрибута и т. д.

Реализация операций обычно выполняется программистом вручную. Это кодирование алгоритмов методов.

Завершается реализация классов созданием из них компонентов. При этом необходимо, чтобы компонент представлял правильные интерфейсы.

После реализации классов и подсистем выполняют тестирование.

5.5. Тестирование модулей

Тестирование — это проверка правильности работы ПО. Созданные программы запускают с целью выявить ошибки в их работе. Тестирование модулей позволяет проверить корректность реализации компонентов перед их интегрированием в систему.

Выделяют два вида тестирования модулей:
• тестирование спецификации — «черный ящик»;
• тестирование структуры — «белый ящик».

Тестирование спецификации дает возможность на основе описания требований к модулю (прецедентов, которые он реализует)

проверить выполнение прикладных функций, функциональных и специальных требований. При тестировании спецификации выявляются ошибки в составлении алгоритмов, ошибки передачи входных или выходных данных, а также ошибки инициализации внутренних структур программы. Для этого используют методы функционального тестирования, например методы граничных значений или разбиение на классы эквивалентности.

Тестирование структуры позволяет проверить корректность перевода проекта в программный код, выявить синтаксические и семантические ошибки в программе. При тестировании структуры обнаруживают ошибки кодирования алгоритмов и использования переменных, составления условий или задания циклов. Для этого применяют методы структурного тестирования, например методы ветвей и границ или проверки циклов.

Далее рассмотрим тестовый пример для АСУ таксопарком.

Описание теста: проверка правильности добавления объектов в БД.

Сценарий теста:

1) запустить программу и перейти на форму «Добавление автомобилей»;

2) создать новую машину с номером «aa123a», маркой «BMW» и цветом «Red»;

3) добавить ее в коллекцию машин;

4) заново получить коллекцию машин.

Ожидаемый результат: количество элементов в коллекции возросло на единицу и последний ее элемент имеет данные параметры.

При выявлении ошибок в модулях их исправляют. Компоненты, успешно прошедшие тестирование, собирают в билд.

Однако тестирование модулей обеспечивает проверку только отдельных элементов системы. Для проверки работы системы в целом необходимо выполнить еще один процесс разработки — тестирование.

Выводы

Рабочий процесс реализации переводит множество проектных моделей в работающую программную систему.

В результате реализации создают модели реализации и размещения, описание архитектуры, программные компоненты, подсистемы реализации и интерфейсы, составляют планы сборки.

Итерация выполняется как последовательность билдов, реализующих набор прецедентов и дающих приращение функциональности. Набор прецедентов для билда определяют до начала его разработки.

Процесс реализации состоит из реализации архитектуры, подсистем и классов, сборки системы и тестирования модулей.

На основе проектов классов и подсистем автоматически генерируются описания компонентов и подсистем реализации, проводятся кодирование методов классов, компиляция компонентов и сборка билдов. Компоненты размещают по узлам в соответствии с моделью размещения. Созданные компоненты тестируют и интегрируют в систему. Составляют план сборки следующего билда.

Контрольные вопросы и задания

1. Объясните назначение рабочего процесса реализации.
2. Какие артефакты создаются в процессе выполнения рабочего процесса реализации?
3. Из каких действий состоит рабочий процесс реализации?
4. Изложите содержание модели реализации.
5. Куда транслируются классы, активные классы, интерфейсы и подсистемы проектирования при реализации?
6. Как реализуют архитектуру и что будет получено в результате?
7. Объясните, что такое компонент и как определяется его интерфейс.
8. Перечислите действия, из которых состоит сборка системы.
9. Что такое план сборки? Как и когда его составляют?
10. Что такое билд? Каким образом определяют набор прецедентов для следующего билда?
11. Что такое подсистема? Как ее реализуют и какие проверки при этом выполняют?
12. Перечислите действия, из которых состоит реализация классов. Какие из них выполняются автоматически?
13. Как и для чего выполняют тестирование модулей? В чем заключается различие между функциональным и структурным тестированием?

Глава 6. РАБОЧИЙ ПРОЦЕСС «ТЕСТИРОВАНИЕ»

Тестирование позволяет проверить корректность работы всей системы в целом и отдельных ее компонентов, а также выявить и локализовать ошибки в программах. Успешное тестирование означает обнаружение ранее неизвестных ошибок. Следует отметить, что проверить или доказать отсутствие в программе всех ошибок невозможно.

В результате тестирования создаются следующие артефакты:

• план тестирования — перечень и последовательность проведения тестов;

• тестовые примеры — наборы описаний исходных данных и ожидаемых результатов для проведения тестирования;

• процедуры тестирования — описания последовательности выполнения тестов;

• тестовые компоненты — программные модули для проведения автоматического тестирования;

• дефекты — описания ошибок, выявленных в работе тестируемого ПО.

Процесс тестирования состоит из выполнения следующих действий:

• планирование тестирования;

• разработка тестов;

• реализация тестов;

• тестирование целостности;

• тестирование системы;

• оценка результатов тестирования.

Рассмотрим эти действия более подробно.

6.1. Планирование тестирования

Перед проведением тестирования необходимо составить план предстоящих работ, определить стратегию тестирования, провести оценку необходимых ресурсов, сформировать график работ.

Под ресурсами понимаются тестировщики и разработчики тестов, аппаратное и программное обеспечение, время и средства для выполнения работ.

6.2. Разработка тестов

Тест, или тестовый пример, — это описание исходных данных, условий запуска и выполнения программы и ожидаемых результатов для проведения тестирования. Ожидаемым результатом может быть как правильное значение, так и сообщение об ошибке, если это предусмотрено в спецификации. Тест позволяет проверить конкретную функцию или выявить некоторые ошибки.

Тесты составляют таким образом, чтобы при проведении минимального количества экспериментов было обнаружено наибольшее число ошибок.

Тесты подразделяют на следующие категории:

• тесты целостности — проверка выполнения функций системы;

• системные тесты — проверка использования системой ресурсов, например, при параллельной работе;

• регрессионные тесты — повторная проверка уже протестированных программ после их модификации или интеграции с другими компонентами.

Регрессионное тестирование позволяет проверить, не повредило ли добавление новых компонентов работоспособности созданных ранее программ.

Кроме тестов на данном шаге рабочего процесса вырабатывают процедуры тестирования. Они задают последовательность проведения тестов для выявления накапливающихся или комплексных ошибок, а также ошибок взаимодействия компонентов.

Все перечисленные категории тестов выполняют для каждого билда после его сборки и интеграции.

6.3. Реализация тестов

Реализация тестов — это создание программных модулей, используемых для проведения автоматического тестирования. Их разрабатывают на основе описаний процедур тестирования.

Тестовые модули могут быть реализованы как заглушки для тестирования модулей, расположенных выше по иерархии управ-

ления. В этом случае тестовые модули имеют более или менее сложный алгоритм и возвращают тестовые данные или сообщения в ответ на вызов.

Для тестирования модулей, расположенных ниже по уровням иерархии вызовов, тестовые модули реализуют в формате драйверов тестирования. В данном случае они передают в тестируемые модули некоторые данные, получают результаты их обработки и сравнивают с ожидаемыми значениями.

Многие современные средства разработки позволяют автоматизировать реализацию тестовых модулей, а также процессы проведения тестов.

6.4. Тестирование целостности

После подготовки тестовых примеров и тестовых модулей проводят тестирование целостности разрабатываемого ПО.

Тестовые примеры для проверки целостности создают для каждого билда. Проверяется выполнение прикладных функций, которые они реализуют, в соответствии с описанием требований к прецедентам, на основе которых они реализованы. Программные компоненты должны полно и правильно осуществлять функции прецедента, включая перехват и корректную обработку ошибок и исключений.

6.5. Тестирование системы

После тестирования отдельных билдов проводят тестирование итерации. Программные компоненты билда интегрируют в систему, после чего выполняют системные тесты для интеграции.

Проверяется использование компонентами системы ресурсов, а также корректность взаимодействия компонентов системы при их совместной работе.

6.6. Оценка результатов тестирования

После проведения тестирования необходимо оценить полученные результаты. Поскольку отсутствие дефектов в программах может свидетельствовать как об отсутствии ошибок, так и о плохом тестировании, следует проанализировать полноту проведенного тестирования и надежность работы ПО.

Все дефекты, выявленные при тестировании, документируют и передают разработчикам для устранения. Тестирование проводят итерационно, поскольку некоторые ошибки могут выявляться только после устранения сопутствующих.

Выводы

В ходе рабочего процесса «Тестирование» проверяют корректность работы ПО и выявляют его дефекты.

В результате тестирования создают план тестирования, тестовые примеры, процедуры тестирования, тестовые компоненты и описания выявленных дефектов.

Процесс тестирования состоит из планирования тестирования, разработки и реализации тестов, тестирования целостности и системного тестирования, оценки результатов тестирования.

Данный процесс завершает выполнение итерации унифицированного процесса и обеспечивает приращение функционала программного продукта, создание промежуточной или итоговой версии ПО.

Контрольные вопросы и задания

1. Объясните назначение рабочего процесса тестирования.
2. Перечислите артефакты, которые создаются в процессе выполнения тестирования.
3. Назовите действия, из которых состоит рабочий процесс тестирования.
4. Как выполняется планирование тестирования?
5. Что такое план тестирования?
6. Что такое тестовые примеры, процедуры тестирования и тестовые компоненты? Для чего их используют?
7. Как выполняются разработка и реализация тестов?
8. Как и для чего проводят тестирование целостности и системное тестирование?
9. Что такое дефекты? Как их описывают?
10. Объясните, как проводится оценка результатов тестирования. Что она позволяет оценить?

Литература

Бек К. Шаблоны реализации корпоративных приложений: пер. с англ. М.: Вильямс, 2008. 176 с.

Вигерс К. Разработка требований к программному обеспечению: пер. с англ. СПб: БХВ-Петербург, 2014. 716 с.

Гусятников В.Н. Стандартизация и разработка программных систем. М.: Финансы и статистика, 2010. 288 с.

Джалота П. Управление проектами в области информационных технологий: пер. с англ. М.: Лори, 2013. 300 с.

Йордан Э. Объектно-ориентированный анализ и проектирование систем: пер. с англ. М.: Лори, 2014. 264 с.

Ларман К. Применение UML 2.0 и шаблонов проектирования: пер. с англ. М.: Вильямс, 2013. 736 с.

Мацяшек Л.А., *Лионг Б.Л.* Практическая программная инженерия на основе учебного примера: пер. с англ. М.: Бином. Лаборатория знаний, 2013. 956 с.

Орлов С.А., *Цилькер Б.Я.* Технологии разработки программного обеспечения. СПб.: Питер, 2012. 609 с.

Полис Г., *Огастин Л.*, *Мадхар Д.* Разработка программных проектов на основе Rational Unified Process (RUP): пер. с англ. М.: Бином-Пресс, 2009. 256 с.

Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес; пер. с англ. СПб.: Питер, 2009. 368 с.

Рэшка Дж. Тестирование программного обеспечения: пер. с англ. М.: Лори, 2012. 568 с.

Фаулер М. Шаблоны корпоративных приложений: пер. с англ. М.: Вильямс, 2014. 544 с.

Якобсон А., *Буч Г.*, *Рамбо Дж.* Унифицированный процесс разработки программного обеспечения: пер. с англ. СПб.: Питер, 2002. 493 с.

Оглавление

Предисловие 3
Введение 5
Глава 1. МЕТОДОЛОГИЯ RUP 7
1.1. Особенности RUP 7
1.2. Этапы RUP 8
1.3. Рабочие процессы и итерации 10
Выводы 12
Контрольные вопросы и задания 12
Глава 2. РАБОЧИЙ ПРОЦЕСС «ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ» 13
2.1. Определение требований 13
2.2. Определение функциональных требований в виде прецедентов 19
Выводы 28
Контрольные вопросы и задания 29
Глава 3. РАБОЧИЙ ПРОЦЕСС «АНАЛИЗ ТРЕБОВАНИЙ» 30
3.1. Анализ архитектуры 31
3.2. Анализ прецедента или кооперации 35
3.3. Анализ класса 42
3.4. Анализ пакетов 46
Выводы 46
Контрольные вопросы и задания 47
Глава 4. РАБОЧИЙ ПРОЦЕСС «ПРОЕКТИРОВАНИЕ» 48
4.1. Проектирование архитектуры 49
4.2. Проектирование прецедентов 55
4.3. Проектирование класса 59
4.4. Проектирование подсистем 64
Выводы 65
Контрольные вопросы и задания 65
Глава 5. РАБОЧИЙ ПРОЦЕСС «РЕАЛИЗАЦИЯ» 67
5.1. Реализация архитектуры 68
5.2. Сборка системы 70
5.3. Реализация подсистемы 71
5.4. Реализация класса 71
5.5. Тестирование модулей 72

Выводы 73
Контрольные вопросы и задания 74
Глава 6. РАБОЧИЙ ПРОЦЕСС «ТЕСТИРОВАНИЕ» 75
6.1. Планирование тестирования 75
6.2. Разработка тестов 76
6.3. Реализация тестов 76
6.4. Тестирование целостности 77
6.5. Тестирование системы 77
6.6. Оценка результатов тестирования 77
Выводы 78
Контрольные вопросы и задания 78
Литература 79

Учебное издание

Виноградова Мария Валерьевна
Белоусова Валентина Ивановна

Унифицированный процесс разработки программного обеспечения

Редактор *О.М. Королева*
Художник *А.С. Ключева*
Корректор *Л.В. Забродина*
Компьютерная верстка *С.А. Серебряковой*

В оформлении использованы шрифты
Студии Артемия Лебедева.

Оригинал-макет подготовлен
в Издательстве МГТУ им. Н.Э. Баумана.

Подписано в печать 06.11.2015. Формат 60×90/16.
Усл. печ. л. 5,125. Тираж 50 экз. Изд. № 159-2015. Заказ

Издательство МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru
www.baumanpress.ru

Отпечатано в типографии МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
baumanprint@gmail.com