ГЛАВА 1

Введение. Что такое предметно-ориентированное проектирование

Начинать проект разработки с нуля обычно сплошное удовольствие. Разработчики реагируют на новые требования сразу. Пользователи полны энтузиазма. Проект движется семимильными шагами.

Со временем все меняется. Становится труднее реализовывать программное обеспечение — учащаются ошибки, замедляется прогресс и ухудшается сопровождаемость. До такой степени, что даже мельчайшее изменение попадает в продакшен только через полгода. Куда же делся вдохновляющий чистый лист? Вместо него у нас теперь «легаси-система», «древнее ПО», «монолит», «большой комок грязи» и «проект в резиновых сапогах»[1].

Не теряем надежду! Даже стареющим системам можно вернуть гибкость, устойчивость и динамичность. В этой главе мы начнем с конкретного примера легаси-системы, модернизированной с помощью предметно-ориентированной трансформации, а затем в общих чертах опишем сам метод. Вы получите общее представление о методе, а также поймете, какие темы книги представляют для вас наибольший интерес и какие главы вам стоит прочитать.

Пример: легаси-система в контейнерном порту

Прадед Хеннинга торговал какао. Из своего кабинета в портовой конторе он любил наблюдать за гигантскими океанскими пароходами. Открывая окно, он чувствовал запахи необъятного мира: созревающих бананов, жарящегося кофе, иногда даже диких животных. В 30-е годы прошлого века суда разгружались портовыми грузчиками, которые взваливали на спины мешки и ящики и таскали их с кораблей на большие склады.

---

[1] Rubber boot project — ироничное обозначение проекта, где приходится «пробираться через грязь», то есть через очень запутанный и проблемный «спагетти-код». — *Примеч. ред.*

С изобретением контейнера в 1960-х годах работа в порту кардинально изменилась. Склады перестали использоваться, и их заменили контейнерные терминалы. Грузчики прошли переобучение, чтобы управлять транспортом, который теперь перевозил контейнеры с грузами.

На рис. 1.1 изображена схема такого терминала. Контейнеры выгружаются с контейнеровоза с помощью причального крана, транспортируются от причала на складскую площадку, а затем автопогрузчик ставит их на грузовик или товарный поезд. Контейнеры, доставленные в терминал грузовиком или поездом для транспортировки контейнеровозом, следуют по обратному маршруту.

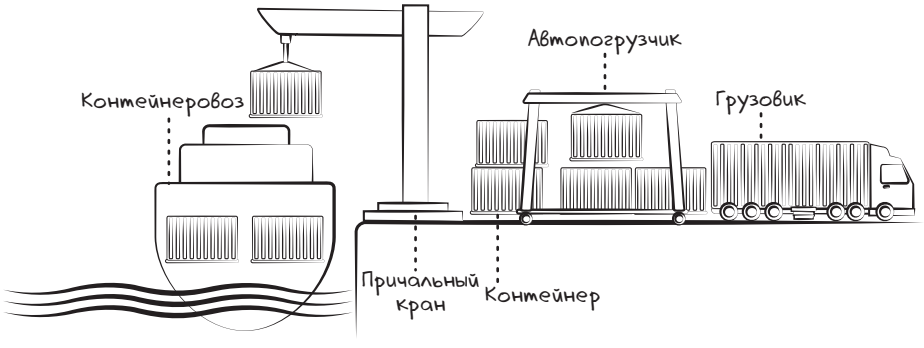

Рис. 1.1. Схема контейнерного терминала

В 1970–1980-х годах в контейнерных терминалах начали использовать программное обеспечение, чтобы контролировать данные и постепенно автоматизировать процессы. Сегодня в том же самом терминале действует целая ИТ-инфраструктура из разных систем: одни планируют погрузку и разгрузку контейнеровозов, другие организуют прибытие и доставку контейнеров, а третьи управляют складом и координируют перемещения на его территории.

Часто несколько функций выполняются одной системой, к которой подключаются периферийные. Этот центральный элемент называется терминальной операционной системой (terminal operating system, TOS). Для простоты[1] дальше мы будем рассматривать только TOS, опуская другие возможные системы (рис. 1.2).

[1] Такая трансформация имеет смысл только для сложных систем. Чтобы дать общее представление о методе, мы описали пример очень схематично, упростив предметную область, архитектуру и рабочие процессы. Надеемся, нам удалось передать общую идею и вдохновить вас на дальнейшее изучение. Аспекты, кратко затронутые здесь, подробно и вдумчиво будут раскрыты в следующих частях книги.

Рис. 1.2. Терминальная операционная система

Директор контейнерного терминала нанял нас, Каролу и Хеннинга, в качестве консультантов для модернизации TOS. По схемам на рис. 1.1 и 1.3 нам объяснили, что TOS нужен сотрудникам КПП, операторам автопогрузчиков, операторам кранов, диспетчерам и планировщикам размещения груза[1]. У каждой категории пользователей в TOS есть свои роли с конкретными функциями для соответствующих рабочих задач. У операторов кранов и автопогрузчиков в кабинах есть мобильное устройство с приложением TOS.

В контейнерном терминале вот уже 30 лет верой и правдой служит TOS. За это время она не раз расширялась и обновлялась — правда, обычно это происходило точечно, а не в рамках хорошо спланированной стратегии, поэтому TOS бесконтрольно разрасталась во всех направлениях и в коце концов превратилась в ненадежный, неповоротливый и старомодный монолит.

Обозначения (модель C4) на рис. 1.2 и 1.3 разъясняются в главе 5. Чтобы получить общее представление, см. врезку «Справка: модель C4».

Справка: модель C4

Модель C4 помогает спроектировать понятную всем стейкхолдерам иерархию программной архитектуры на основе четырех типов схем, соответствующих четырем уровням абстракции. Здесь мы используем представление самого высокого уровня — контекстную диаграмму, на которой видно, как система вписывается в окружение, включающее пользователей, внешние системы и их взаимодействия.

[1] В реальном контейнерном терминале существуют и другие роли, которые мы опускаем для упрощения.

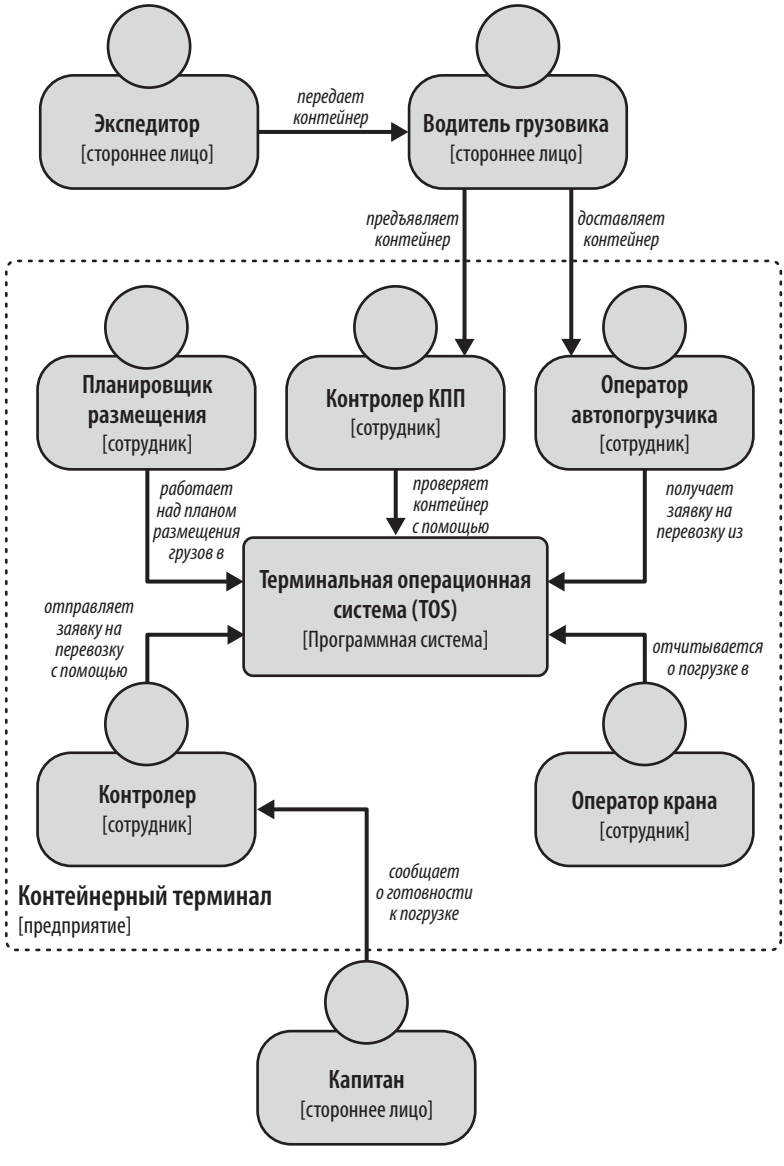

Рис. 1.3. TOS в контексте

Переопределение предметной области

Узнав, кто работает со стареющей TOS, мы приглашаем пользователей — сотрудников из разных отделов контейнерного терминала — и просим в общих чертах описать их работу.

Этот шаг удивляет архитекторов и разработчиков устаревающей системы. Они ожидали, что сначала мы проанализируем исходный код, чтобы найти фрагменты, требующие модернизации. Такой подход помог бы в какой-то степени повысить качество кода, но уж точно не решил бы главную проблему легаси-систем. Ее архитектура не отражает структуру бизнеса, она развивалась бессистемно. Реальные бизнес-области можно выявить только через *переопределение предметной области*, поэтому мы начинаем с опроса сотрудников.

Мы узнаем, какой путь проделывает типичный контейнер в терминале:

1. Перед транспортировкой экспедитор отправляет в контейнерный терминал сопроводительные документы на контейнер, чтобы терминал заранее спланировал нагрузку.
2. Когда контейнер заполнен грузами, а все документы подготовлены экспедитором, грузовик подвозит контейнер к воротам терминала.
3. Контролер КПП проверяет сопроводительные документы и груз. Если все в порядке, грузовик с контейнером получает разрешение на подъезд к пункту перегрузки на территории терминала. Оттуда оператор автопогрузчика забирает контейнер и размещает его на складской площадке.
4. За несколько дней до отправки контейнеровоза планировщик составляет план размещения груза на судне и включает в него наш контейнер. Как только судно прибывает в контейнерный терминал, капитан докладывает об этом ответственному диспетчеру, и начинается погрузка.
5. Диспетчер направляет оператору автопогрузчика заявку на перевозку. Оператор забирает контейнер и доставляет его к причалу. Там кран загружает контейнер в соответствующий отсек на контейнеровозе.
6. Наконец, судно отплывает и идет до порта назначения, где контейнер выгружают.

Выслушивая сотрудников, мы записываем историю предметной области (domain story) (рис. 1.4)[1]. История включает разные роли с рис. 1.3 и показывает, как они работают, не затрагивая пока саму программную систему.

Сообща составляя историю предметной области, мы — и консультанты, и эксперты — узнаем много нового. Например, оператор автопогрузчика не знал, как проверяются транспортные документы. Теперь он понимает, почему иногда контейнеры передаются с такой задержкой.

Методы создания таких историй, как на рис. 1.4, и способы обозначения подробно описаны в главе 4. Чтобы получить общее представление, см. врезку «Справка: метод Domain Storytelling» на с. 29.

[1] Для упрощения некоторые детали процесса опущены.

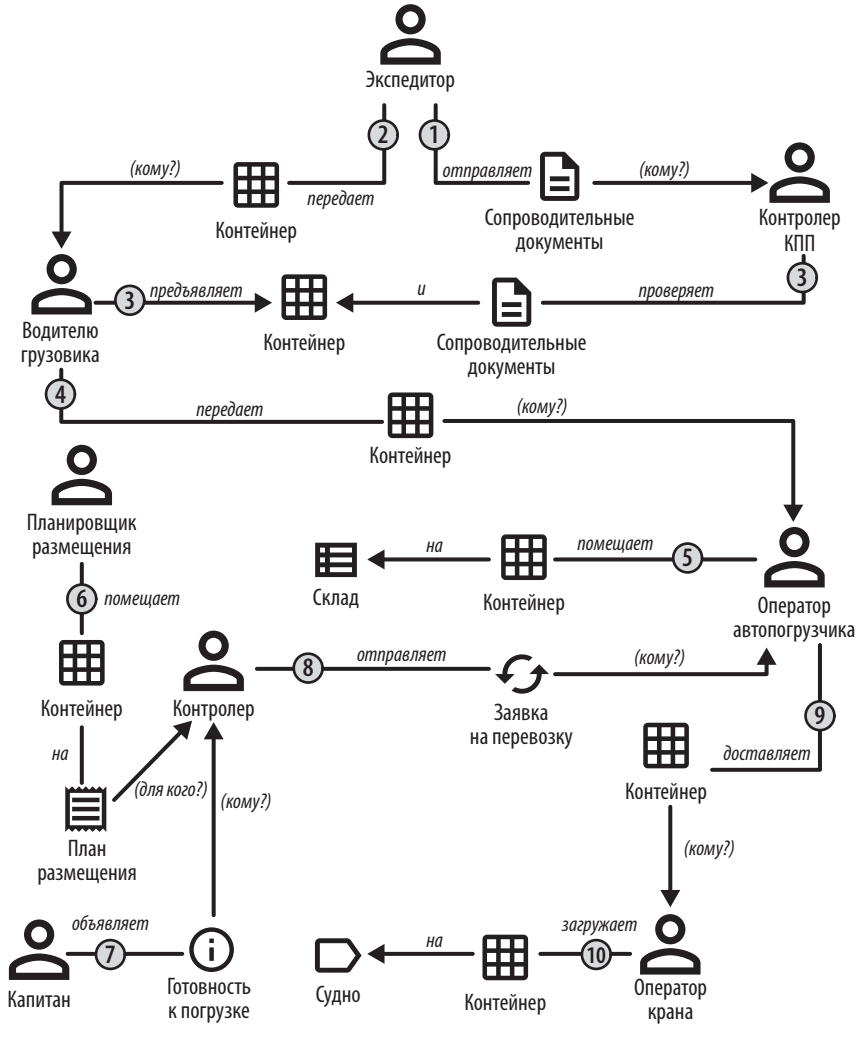

Рис. 1.4. Общая схема процесса в контейнерном терминале. Переходите между числами по стрелкам, чтобы проследить все действия в правильном порядке

Определение целевой архитектуры

Составив общую схему и узнав много новых деталей о работе контейнерного терминала, мы переходим к следующей задаче — определить бизнес-области или подобласти (business areas, subdomains).

Для этого мы приглашаем к активному обсуждению экспертов предметной области. Снова и снова мы размышляем над несколькими вопросами:

- Какие шаги или действия в предметной истории связаны между собой?
- Где можно выделить связанные бизнес-области, формирующие подобласть?
- Какие названия наиболее точно отражают суть подобластей?

Мы узнаем, например, что экспедитор, водитель грузовика и контролер КПП взаимодействуют в одной бизнес-области, цель которой — доставка контейнера в порт. Сотрудники терминала называют ее «въезд в порт». Границу можно определить, например, по тому, что шаг 5 выполняется только в одну сторону. После него процесс не возвращается к въезду в порт и его участникам — экспедитору, контролеру КПП и водителю.

В результате обсуждения мы разделили предметную область на несколько подобластей (рис. 1.5).

Как мы выделили подобласти и какие правила и индикаторы нам в этом помогли, объясняется в главах 3 и 10.

Далее мы определяем, какие подобласти поддерживаются системой, — выписываем их названия на доску и рисуем зависимости между этими частями процесса. В результате мы получаем карту контекстов с ограниченными контекстами (рис. 1.6). По этой карте мы будем выстраивать целевую архитектуру.

Каждому ограниченному контексту на карте в целевой архитектуре соответствует предметно-ориентированный модуль, который взаимодействует с другими модулями через узкие интерфейсы, как на схеме выше.

В главах 3 и 11 мы узнаем, как выделять ограниченные контексты на основе подобластей и как рисовать и использовать карты контекстов.

Справка: метод Domain Storytelling

Domain Storytelling — это метод совместного моделирования, в ходе которого эксперты предметной области и разработчики пытаются прийти к общему пониманию бизнес-процессов. Вместе они визуализируют свое понимание в виде диаграммы с действующими лицами (человечки), рабочими объектами (другие значки) и действиями (стрелки). Чтобы прочитать историю целиком, нужно переходить по стрелкам по порядку, обозначенному числами.

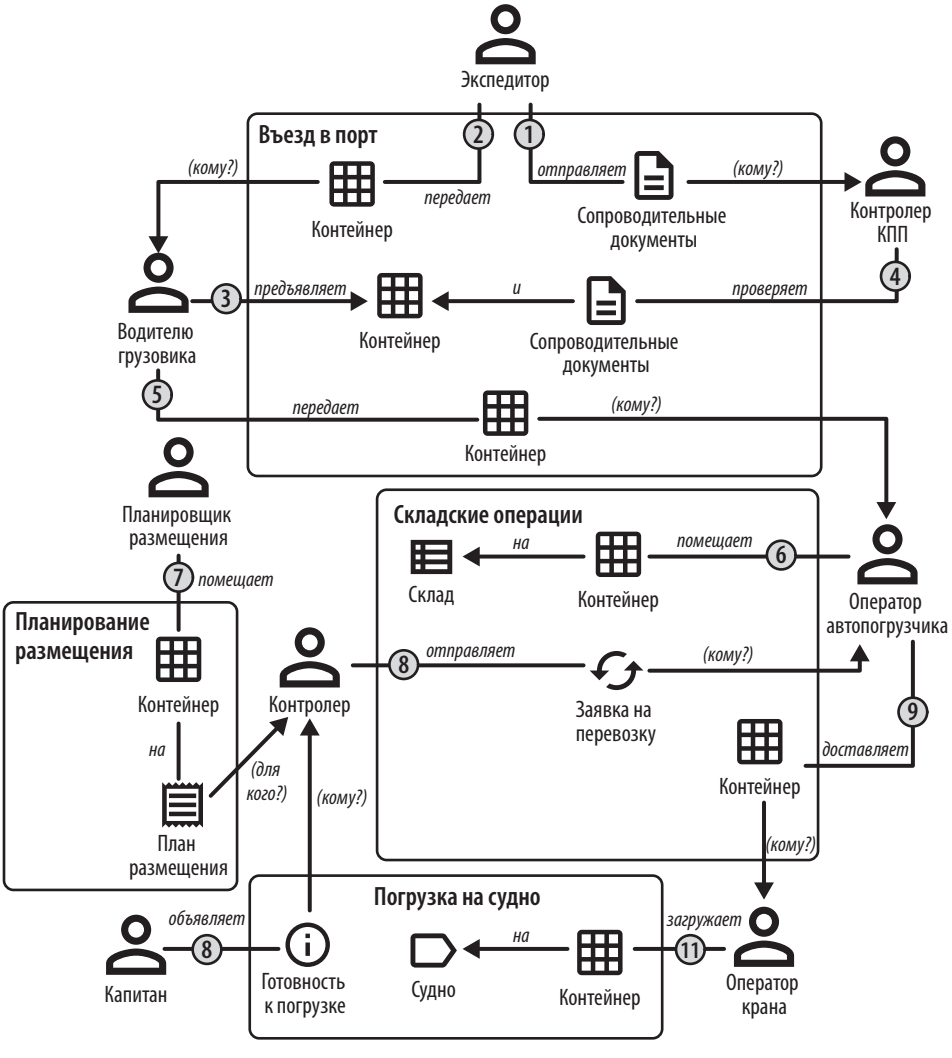

Рис. 1.5. Общая схема процесса в контейнерном терминале — с границами

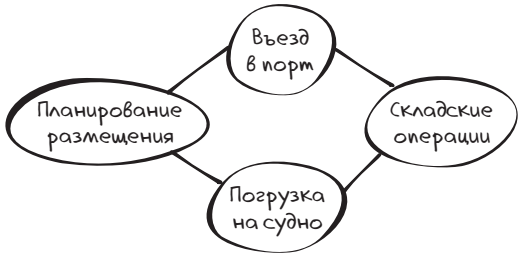

Рис. 1.6. Целевая карта контекстов контейнерного терминала

Сравнение фактической архитектуры с целевой

Итак, мы определили *желаемую* структуру системы, а теперь сравниваем целевую архитектуру с фактической. Этот этап радует архитекторов и разработчиков, ведь мы наконец обращаем внимание на код. TOS написана на Java. Для сравнения архитектур мы загружаем исходный код TOS в инструмент анализа архитектуры и моделируем целевую структуру поверх кодовой базы. Подходящие инструменты для этого этапа см. в книге *Sustainable Software Architecture* [60].

На рис. 1.7 показан неутешительный результат первого сравнения. На диаграмме в виде прямоугольников изображены четыре ограниченных контекста из карты контекстов. Мы называем их *модулями*. Мы создали их с помощью инструмента анализа архитектуры и назначили им части дерева каталогов с соответствующими классами.

Рис. 1.7. Скриншот инструмента анализа архитектуры: сравнение фактической и целевой архитектуры. Если вы незнакомы с такими инструментами, см. врезку «Справка: инструменты анализа архитектуры»

Справка: инструменты анализа архитектуры

Инструмент анализа архитектуры помогает визуализировать текущую архитектуру и прийти к единому пониманию ее структуры. Еще в нем удобно сравнивать текущую архитектуру с целевой.

В этой книге приводятся скриншоты из инструмента Sonargraph. Если нажать на значок «плюс» перед квадратами (модулями), дерево каталогов и классы развернутся. В правой части схемы видны реальные связи из исходного кода, представленные дугами. Слева тоже есть дуги, которые отображают связи между классами, рассортированными по модулям. Дуги слева направлены сверху вниз, а справа — снизу вверх.

Мы рисуем текущую схему компонентов на доске (снова в нотации модели C4; см. рис. 1.8), чтобы все разработчики и архитекторы поняли результат анализа, выполненного с помощью инструмента с рис. 1.7. Получается, что в системе, с одной стороны, должно быть четыре предметно-ориентированных модуля, а с другой — техническая структура, состоящая из нескольких уровней: пользовательский

интерфейс, приложение для процессов в предметной области, модель предметной области, база данных и т. д. Так должна выглядеть хорошо структурированная модульная архитектура, но *нынешняя TOS даже близко на нее не похожа*! Мы видим структуру со множеством переплетений и циклических зависимостей.

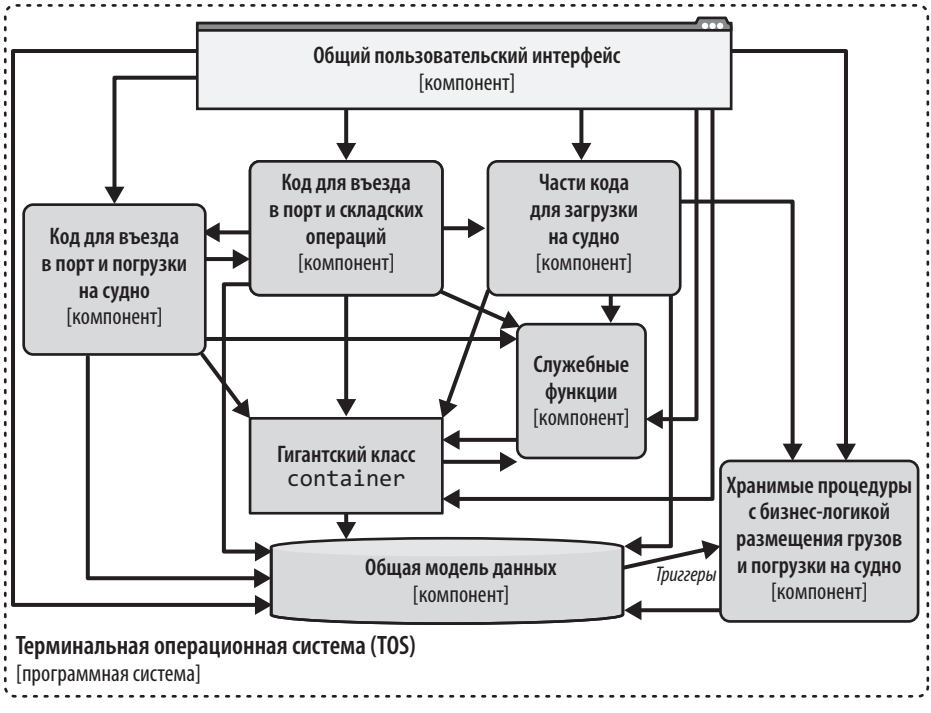

Рис. 1.8. Сейчас TOS — это сильно переплетенная структура со множеством циклических зависимостей и некоторым разделением на технические уровни

Стрелки между блоками на рис. 1.8 указывают на множество связей между компонентами — и некоторые из них даже циклические. Система далека от идеала, но худо-бедно обеспечивает функционал для четырех подобластей. По крайней мере, тут прослеживается некоторое *разделение на технические уровни*: общий пользовательский интерфейс формирует верхний уровень, общая база данных — нижний, а все остальное («бизнес-логика») — посередине.

Мы видели немало систем в подобном состоянии и уже не удивляемся, что TOS работает ненадежно и с трудом поддается расширению. Чтобы понять истинный масштаб проблемы, мы определяем индекс зрелости модульной системы.

Какие аспекты оценивает индекс зрелости модульной системы и как он рассчитывается, мы рассказываем в главе 5. Пока же ознакомьтесь со справкой по индексу зрелости модульной системы.

Справка: индекс зрелости модульной системы

Индекс зрелости модульной системы (Modularity Maturity Index, MMI) рассчитывается на основе ряда метрик и критериев качества по шкале от 0 до 10 (рис. 1.9).

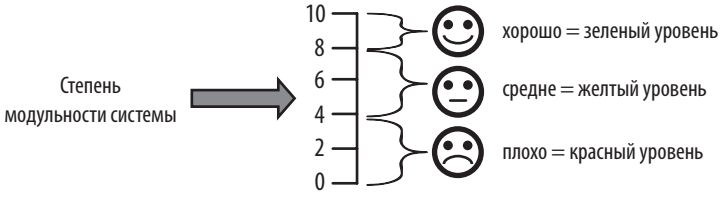

Рис. 1.9. Индекс зрелости модульной системы

Рассчитанное значение входит в один из трех уровней: хорошо структурированные системы имеют индекс от 8 до 10, средненькие — от 4 до 8, а системы с плохой структурой — ниже 4.

Для нашей системы TOS мы рассчитали индекс 4,9 — так себе показатель на всех уровнях.

Низкое значение частично связано с тем, что предметная концепция «контейнер» со всеми ее приключениями (въезд в порт, складские операции, планирование размещения, погрузка на судно) смоделирована в одном огромном классе с несколькими другими классами, выполняющими подзадачи. Данные из класса `Container` со всеми связями хранятся в одной гигантской таблице в базе данных.

В начале разработки TOS класс `Container` и связанная таблица были небольшими и прекрасно подходили для ранней версии, когда она еще была компактной TOS для поддержки первых процессов в контейнерном терминале. Но с каждым добавлением в TOS нового функционала класс контейнера и его таблица в базе данных разрастались-разрастались и разрослись до *божественного класса (god class)*, или *неограниченной предметной модели (unbounded domain model)*.

Мы хотим прийти к структуре, как на рис. 1.10, где у каждого ограниченного контекста с рис. 1.6 есть свой класс Container, вписанный в свой контекст и взаимодействующий с другими классами через узкие интерфейсы.

В инструменте анализа архитектуры эта структура также будет выглядеть значительно аккуратнее (рис. 1.11).

В таких случаях, чтобы навести порядок в структурах и связях, необходим продуманный план, разбивающий путь от фактической архитектуры к целевой на выполнимые этапы. В части III мы рассмотрим методы, с помощью которых вы

сможете самостоятельно разработать такой план для реструктуризации легаси-системы (см. главу 12).

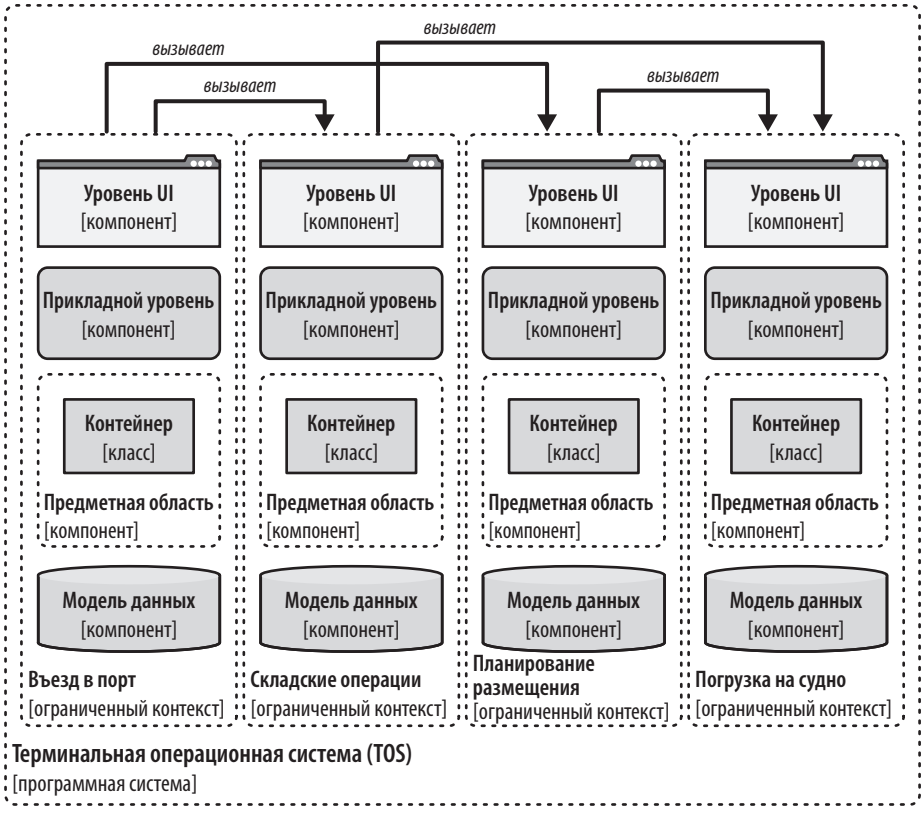

Рис. 1.10. Целевая архитектура: ограниченные контексты с разными реализациями класса Container

Рис. 1.11. Скриншот инструмента анализа архитектуры: желаемый результат. Справа от блоков больше нет восходящих связей

Рассчитав индекс зрелости модульной системы, мы определили, где у TOS будет начало преобразований. Это мы обсуждаем с разработчиками и архитекторами,

чтобы вместе составить план. Наша отправная точка — шаблоны проектирования, которые разработчики TOS использовали для упорядочивания исходного кода. В коде мы обнаруживаем серию классов *Service*, созданных по общему шаблону и предоставляющих через свои методы функционал без сохранения состояния. Например, `PlanningService`, интерфейс которого показан на схеме классов на рис. 1.12.

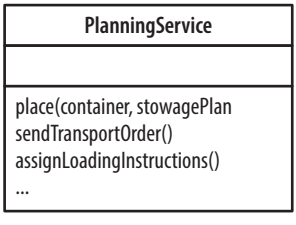

Рис. 1.12. Фактическая архитектура (фрагмент). Проблема: PlanningService объединяет неограниченную бизнес-логику в огромный класс сервиса

При детальном анализе интерфейса вместе с архитекторами и разработчиками мы понимаем, что этот класс предоставляет функционал для двух ограниченных контекстов с карты контекстов (рис. 1.6): для планирования размещения на судне и для операций на складе. На рис. 1.13 мы наложили два контекста на схему классов `PlanningService`, чтобы наглядно показать проблему.

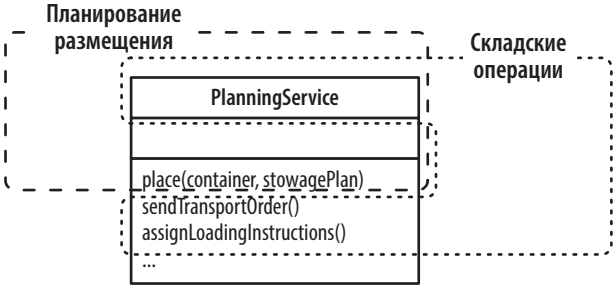

Рис. 1.13. Сопоставление фактической архитектуры с целевой для класса PlanningService. Проблема: в программном обеспечении еще не выражены границы; «Планирование размещения» (пунктирная линия) и «Складские операции» (точечная линия) пересекаются, усложняя чтение схемы

Архитекторы и разработчики TOS ранее проходили у нас обучение по предметно-ориентированному проектированию (DDD). Теперь они вспомнили о стратегическом проектировании в DDD и начали понимать, к чему мы стремимся (если вы не знакомы с DDD, см. врезку «Справка: предметно-ориентированное проектирование» на с. 36).