Объектно-ориентированное моделирование

При компьютерном моделировании сложных систем с успехом используются объектно-ориентированные языки.

Владение объектно-ориентированным языком программирования (например, Java) и доступ к обширной библиотеке ресурсов составляют необходимое, но не достаточное условие для создания объектной системы. Очень важную роль в процессе ее разработки играют анализ и проектирование системы с точки зрения объектной методологии.

Аббревиатура UML означает *Unified Modeling Language* (унифицированный язык моделирования). Этот язык представляет собой систему обозначений, которая базируется на диаграммах и предназначается для моделирования систем на основе объектно-ориентированного подхода.

- Как распределить обязанности между классами и объектами? Как должны взаимодействовать объекты? Какие функции выполняют конкретные классы? Эти вопросы являются определяющими при разработке системы. Некоторые проверенные временем решения проблем, возникающих в процессе разработки, могут быть (и были) сформулированы в виде набора принципов, эвристик или шаблонов (patterns) — именованных формул решения проблем, позволяющих систематизировать процесс разработки конкретных систем.

Унифицированный язык моделирования UML

UML — это "язык для определения, визуализации и конструирования артефактов программных систем" Это система обозначений (включая семантику), предназначенная для моделирования систем на основе объектно-ориентированного подхода.

UML— это важный производственный стандарт для объектно-ориентированного моделирования. Он появился в 1994 году в результате совместных усилий Гради Буча (Grady

Booch) и Джима Румбаха (Jim Rumbaugh) по объединению их популярных методов — метода Буча и OMT (Object Modeling Technique). Впоследствии эти две методология были объединены Айваром Якобсоном (Ivar Jacobson), создателем метода OOSE (Object-oriented Software Engineering). В ответ на запрос группы промышленных стандартов OMG (Object Management Group) об определении стандартного языка моделирования и общепринятой системы обозначений в качестве кандидата в 1997 году был представлен язык UML.

Группа OMG сертифицировала UML, который к тому времени де-факто получил одобрение специалистов многих крупных компаний, Многие организации, специализирующиеся на разработке программного обеспечения, и производители CASE-средств также приняли UML. Поэтому с высокой вероятностью можно утверждать, что этот язык станет мировым стандартом для разработчиков, авторов и производителей CASE-средств.

Полное описание системы обозначений UML можно найти на Web-узле группы OMG по адресу www.omg.org.

Для представления артефактов объектно-ориентированного анализа и проектирования существует порядка десяти различных систем-обозначений. Эта ситуация затрудняет эффективное сотрудничество между группами разработчиков, обучение и использование CASE-средств. Авторы UML— Буч, Якобсон и Румбах — создали стандартизированный, элегантный, выразительный и гибкий язык моделирования и тем самым внесли значительный вклад в развитие объектной технологии проектирования.

UML— это язык моделирования, а не руководство разработчика по объектно-ориентированному анализу и проектированию.

Естественно, методы, модели и средства создания эффективных, программных систем будут развиваться и в дальнейшем. Однако лишь сейчас специалисты получили возможность пользоваться единым языком — UML.

Анализ и проектирование

Для создания программного приложения необходимо описать проблему и требования к системе. Этап *анализа* (analysis) состоит в исследовании проблемы, а не в поисках путей ее решения. Например, при разработке новой информационной системы для компьютерной **библиотеки** необходимо описать экономические процессы, связанные с ее использованием,

При разработке приложения необходимо также обеспечить высокий уровень и подробное описание логики решения, удовлетворяющего требованиям к системе и налагаемым ограничениям. В процессе *проектирования*(design) основное внимание уделяется логическому решению, обеспечивающему выполнение основных требований. Например, как на самом деле будет функционировать информационная библиотечная система? Безусловно, проект может быть реализован в виде аппаратных средств и программного обеспечения.

Объектно-ориентированный анализ и проектирование

Основная идея *объектно-ориентированного анализа и проектирования* (object-oriented analysis and design) состоит в рассмотрении предметной области и логического решения задачи с точки зрения объектов (понятий или сущностей).

В процессе объектно-ориентированного анализа основное внимание уделяется определению и описанию объектов (или понятий) в терминах предметной области. Например, в случае библиотечной информационной системы среди понятий должны присутствовать Book(книга), Library (библиотека) и Patron(клиент).

В процессе объектно-ориентированного проектирования определяются логические программные объекты, которые будут реализованы средствами объектно-ориентированного языка

программирования. Эти программные объекты включают в себя атрибуты и методы. Например, в библиотечной **системе** программный объект Book может содержать атрибут title (название) и **метод** print (печатать)

И наконец, в процессе *конструирования* (construction) или *объектно-ориентированно программирования* (object-oriented programming) обеспечивается реализация разработанных компонентов, таких как класс Book на языке C++, Java. Smalltalk или Visual Basic.

Введение в процесс разработки

Процесс разработки программного обеспечения (software development process) представляет собой метод организации видов деятельности, связанных с созданием, поставкой и поддержкой программных систем.

Ниже представлено краткое введение в основы данного процесса, а также приведено описание его основных этапов.

Рекомендуемые процесс и модели

Язык UML не определяет стандарт процесса. Его авторы признают, что важное значение имеют как надежный язык моделирования, так и сам процесс разработки. Они лишь приводят свои рекомендации относительно организации процесса в отдельных не относящихся к UML публикациях, поскольку стандартизация процесса разработки программных систем выходит за рамки языка UML.

Следование стандартам процесса разработки Приложений не играет основной роли в создании Системы. Разработчику гораздо важнее обладать

навыками создания хороших проектов. Для этого необходимо освоить ряд принципов и эвристик, связанных с идентификацией и выделением основных абстрактных объектов, а также с распределением обязанностей между ними.

Предлагаемые этапы разработки и модели — это лишь иллюстрация многообразия процесса разработки программного обеспечения с использованием объектной технологии, а не готовая формула. Предлагаемые примеры видов деятельности можно рассматривать в качестве отправной точки для обсуждения и выработки удобного процесса разработки, а также для экспериментирования с ним.

Обычно описание процесса включает в себя все виды деятельности, от формулировки требований к системе до вопросов ее распространения. Кроме того, зачастую полное описание процесса включает более широкий круг вопросов, связанных с автоматизацией процесса разработки программного обеспечения, определением долгосрочного жизненного продукта, документированием, поддержкой и обучением персонала, распараллеливанием работ и их координацией между отдельными группами. Мы сосредоточимся лишь на основных видах деятельности, уделяя вопросам взаимодействия групп разработчиков лишь незначительное внимание.

Некоторые существенные этапы процесса разработки останутся вне нашего внимания, а именно**:** планирование, параллельное взаимодействие групп разработчиков, управление проектом, документирование и тестирование.

Язык UML позволяет стандартизировать артефакты и систему обозначений, но не определяет стандарт процесса разработки. Это объясняется следующими причинами.

1. Стандартную систему обозначений можно использовать при создании систем самого

различного назначения, процесс разработки которых может отличаться от стандартного.

2. Процесс разработки системы может зависеть от множества факторов, таких как квалификация специалистов, доля творчества в общем процессе, природа проблемы, средства ее решения и т.д.

Здесь будут рассмотрены лишь общие принципы и типичные этапы успешного процесса разработки программных систем.

На высоком уровне можно выделить следующие основные этапы разработки программного приложения

1. **Планирование** — планирование, определение требований, создание прототипов и т.д.
2. **Построение** — конструирование системы.
3. **Развертывание** — ввод системы в действие.

Итеративный процесс разработки

Итеративный жизненный цикл программного продукта основывается на успешной доработке и усложнении системы в процессе *нескольких* циклов анализа, проектирования, реализации и тестирования.

Развитие системы обусловливается добавлением в каждом цикле разработки новых функций. По завершении предварительного этапа планирования система продолжает разрабатываться на этапе построения в течение нескольких циклов разработки.

В каждом цикле решаются задачи удовлетворения небольшого количества требований к системе в процессе анализа, проектирования, построения и тестирования. При выполнении каждого из таких циклов система постепенно совершенствуется.

Такая стратегия разработки противоречит классическому пониманию жизненного цикла, в котором каждый из видов деятельности (анализ,

проектирование и т.д.) выполняется лишь один раз с целью удовлетворения всего набора требований к системе.

Можно выделить следующие преимущества итеративного процесса разработки.

- Эффективность преодоления сложности поставленной задачи
- Быстрая обратная связь, обеспечиваемая коротким временем реализации каждого из элементарных циклов

Полезной стратегией при выполнении каждого цикла разработки является выделение для него определенных временных рамок — краткого фиксированного времени (к примеру, четыре недели). В течение этого временного интервала должна быть выполнена вся работа по реализации цикла. В качестве временных рамок следует выбирать диапазон от двух недель до двух месяцев. Более краткий период не позволит выполнить сложные задачи, а при выборе более длительных интервалов удлиняется период обратной **связи,** что затрудняет преодоление сложности поставленной задачи.

Прецеденты и итеративные циклы разработки

Прецедент – это словесное описание некоторого процесса из предметной области, например заказ книг из библиотеки.

Последовательные циклы разработки определяются требованиями прецедентов

Цикл разработки призван реализовать один или несколько прецедентов либо упрощенных версий прецедентов. Некоторые циклы разработки (особенно на ранних этапах) должны быть посвящены выполнению не только очевидных, но и неочевидных требований, таких как создание служб поддержки (надежного функционирования, безопасности и т.д.)

Прецеденты необходимо ранжировать и реализовывать на более ранних стадиях разработки прецеденты с более высоким приоритетом.

Этап планирования

На этапе планирования проекта предусматриваются выделение основных понятий, исследование альтернативных возможностей, планирование, спецификация требований и т.д.

На этом этапе могут генерироваться следующие артефакты.

- **План —** график выполнения, ресурсы» бюджет и т.д.
- **Предварительный отчет** — обоснование альтернативы, экономические потребности
- **Спецификация требований —** декларация основных требований
- **Словарь** — список терминов (понятий) и связанной с **ними** информации, на пример правил и ограничений
- **Прототип** ~ прототип системы, создаваемый для более глубокого понимания сущности проблемы, основных рисков и требований
- **Прецеденты** — словесные описания процессов предметной области
- **Диаграммы прецедентов** — иллюстрация всех прецедентов и их взаимоотношений
- **Приблизительная концептуальная модель** — предварительная концептуальная модель, предназначенная для более глубокого понимания словаря предметной области, прецедентов и требований.

Этап построения: циклы разработки

Этап построения проекта включает несколько циклов разработки (возможно, помещенных во временные рамки), в процессе которых происходит расширение системы. Основное назначение этого этапа — создать работающую программную систему, корректно удовлетворяющую **всем** требованиям.

Основными шагами одного цикла разработки являются анализ и проектирование. Подробное описание этих шагов приводится ниже. Артефакты не всегда создаются последовательно. Некоторые артефакты могут создаваться параллельно, например, при создании концептуальной модели и словаря; создании диаграммы взаимодействий и диаграммы классов.

Требования к системе

Чтобы успешно реализовать проект, необходимо корректно сформулировать требования к системе. Для этого нужны специальные знания, детальное изложение которых не входит в наши планы. Наша задача — освоить процесс объектно-ориентированного анализа и проектирования. Однако формулировка требований к системе — очень важный этап, поэтому вкратце рассмотрим процесс на примере торгового приложения.

Основное назначение данного раздела — научить читателя формулировать требования к системе, а не сделать его экспертом в предметной области торговых предприятии и терминальных торговых систем**.** Поэтому перечень функций и атрибутов системы является, скорее, иллюстративным, чем исчерпывающим.

Требования (requirements) -- это описание необходимых или желаемых свойств продукта. Основная задача на этапе определения требований состоит в идентификации и документировании необходимых свойств системы в форме, удобной для

заказчика и команды разработчиков. Очень важно корректно сформулировать требования, чтобы идентифицировать возможные риски и избежать неприятных сюрпризов после окончательной поставки готового продукта.

На стадии определения требований рекомендуется использовать следующие основные артефакты.

- Общая формулировка задачи
- Потребители системы
- Цели
- Функцией системы
- Атрибуты системы

Объектно-ориентированный анализ и проектирование проиллюстрируем на примере системы розничной торговли

Это компьютеризированная система организации товарооборота и обработки платежей, используемая в обычных магазинах. Система розничной торговли включает аппаратные средства (компьютер и устройство считывания штрих-кода), а также программное обеспечение, выполняющее основные задачи системы.

Предположим, требуется создать программный продукт для обслуживания терминала торговой точки. На основе последовательной стратегии разработки мы выполним все необходимые этапы создания системы: формулировку требований, объектно-ориентированный анализ**,** проектирование и реализацию.

Сейчас по предложенному выше плану определим требования к системе.

Общая формулировка задачи

Цель проекта — создание терминальной торговой системы.

Потребители

Компания "Магазин объектов", межнациональный распространитель объектов.

Цели

Основная цель — повышение уровня автоматизации торговли для обеспечения более быстрого, эффективного и дешевого выполнения экономических операций.

- Быстрое обслуживание клиентов
- Быстрый и точный анализ торговой деятельности предприятия
- Автоматическая инвентаризация товаров

Функции системы

Функции системы (system functions) — это ее основное назначение, например авторизация кредитных платежей. Они должны быть определены и систематизированы в логически связанные группы.

Если X действительно является функцией системы, то имеет смысл следующее предложение: Система должна выполнять **<X>.** Например, система должна выполнять авторизацию кредитных платежей. *Атрибуты системы* (system attributes) — это нефункциональные качества системы, например простота в использовании. Очень часто атрибуты путают с функциями. Обратите внимание, что атрибут "простота в использовании" не может быть подставлен в тестовое предложение: "Система должна выполнять простоту в использовании". Атрибуты системы не являются частью функциональной спецификации, а определяются в отдельном документе (спецификации атрибутов).

Категории функций

Функции системы, например авторизация кредитных платежей, должны быть систематизированы

с учетом их приоритета. Существуют следующие категории функций.

Категория функций	Значение
Очевидные	Их выполнение очевидно для пользователя
Скрытые	Должны выполняться незаметно для пользователя, Это касается многих базовых технических функции, таких как сохранение информации на постоянном носителе. Скрытые функции зачастую необоснованно упускаются в процессе определения требований к системе
Дополнительные	Необязательные функции, добавление которых не приведет к существенному удорожанию проекта и не повлияет на выполнение остальных функций

Атрибуты системы

Атрибуты системы – это нефункциональные характеристики системы, например: простота в использовании, отказоустойчивость, стоимость.

Атрибуты системы могут относиться ко всем функциям одновременно, либо к одной или нескольким функциям. Атрибуты системы характеризуются допустимым набором значений и могут принимать дискретные, нечеткие или смысловые значения.

Другие артефакты стадии формулировки требований

В этой главе изложены лишь краткие сведения о требованиях к системе. На самом деле данному вопросу можно посвятить целую книгу**.** Описание системных функций и атрибутов — это лишь

минимальный набор документов, создаваемых на этапе определения требований к системе. Однако для более глубокой проработки вопроса и снижения риска разработки необходимо учитывать и другие важные артефакты.

- *Группы разработчиков требований* — перечень участников, вовлеченных в процесс формулировки требований, определения функций и атрибутов системы, выполнения обзоров и проведения интервьюирования.
- *Группы участников разработки* — перечень групп, принимающих участие в разработке или развертывании системы
- *Предположения* — сделанные допущения
- *Риски* ~ факторы, которые могут повлиять на успешную реализацию проекта
- *Зависимости* — компании, системы или продукты, от которых зависит выполнение проекта
- *Словарь терминов* — определение основных терминов
- *Прецеденты* - описания процессов, происходящих в предметной области (обсуждаются в последующих главах)
- *Приблизительная концептуальная модель* - модель основных понятий и их взаимосвязей

Описание процессов: прецеденты

Для того чтобы определить требования, лучше всего создать прецеденты — описания происходящих в предметной области процессов. В этой главе содержится введение в концепцию прецедентов, а также иллюстрируется се использование на примере системы розничной торговли.

Прецеденты и диаграммы прецедентов входят в состав языка UML

Виды деятельности и зависимости

Прецеденты оказываются зависимыми как минимум от частичного понимания требований к системе, а в идеале отражаются в документах, содержащих спецификацию требований.

Прецеденты

Прецедент (use case) представляет собой документ, описывающий последовательность событий, связанных с исполнителем (внешним агентом), который для завершения требуемого процесса использует создаваемую систему. Прецеденты являются описанием или вариантами использования системы. Они не совпадают с требованиями или функциональными спецификациями, однако, в то же время предоставляют описание и иллюстрируют требования. Для обозначения прецедента используется пиктограмма, изображенная на рисунке

Пиктограмма языка UML, используемая для обозначения прецедента

Пример прецедента высокого уровня: Buy Items

Следующий прецедент высокого уровня сжато описывает процесс приобретения покупок в магазине, где установлен терминал розничной торговли.

Прецедент	Buy Items (Приобретение товаров)
Исполнители	Покупатель, кассир
Тип	Главный (рассматривается ниже)
Описание	Покупатель подходит к кассе с товарами, которые он желает приобрести. Кассир регистрирует их и вычисляет общую цену. В завершение покупатель покидает магазин с приобретенными товарами

Заголовки и структура прецедента являются типичными. Тем не менее, язык UML не определяет какой-либо жесткий формат. Может быть выбрана и другая структура в зависимости от требований к документации, что может значительно облегчить дальнейшую работу.

Для того чтобы быстрее познакомиться с основными процессами предметной области, лучше всего начать с прецедентов высокого уровня.

Пример развернутого прецедента: приобретение товаров за наличные

Развернутые прецеденты (expanded use case) предоставляют более подробное описание, чем прецеденты верхнего уровня. Они оказываются полезными для углубленного понимания происходящих процессов и требований. Зачастую развернутые прецеденты имеют форму диалога между исполнителями и системой. Вот пример развернутого прецедента, описывающего процесс приобретения товаров. В данном случае рассматривается торговля только за наличные, а управление инвентаризацией игнорируется (это сделано для обеспечения простоты первого примера).

Прецедент	Buy Items with Cash {Приобретение товаров за наличные)
Исполнители	Покупатель (инициатор), кассир
Цель	Ведение торговой сделки и связанных с ней платежей
Краткое описание	Покупатель подходит к кассе с товарами, которые он желает приобрести. Кассир регистрирует их и вычисляет общую цену. В завершение покупатель покидает магазин с приобретенными товарами
Тип	Главный и идеальный
Ссылки	Соответствующие функции системы

Типичный ход событий	
Действия исполнителя	Отклик системы
1. Покупатель подходит к кассе системы розничной торговли с товарами, которые он желает приобрести.	
2.Кассир регистрирует идентификатор каждой единицы товара. Если покупатель приобретает несколько единиц одного и того же товара, кассир может также внести их количество.	3. Определяет цену единицы товара и добавляет информацию, требуемую для выполнения транзакции, которая связана с данной продажей. Отображает описание и цену товара.
4. После завершения ввода информации о товаре кассир уведомляет об этом систему розничной торговли.	5. Вычисляет и отображает общую стоимость товара.
6. Кассир сообщает	

покупателю общую стоимость товара.	
7, Покупатель передает кассиру деньги, количество которых, возможно, превышает общую стоимость.	
8, Кассир вводит полученную сумму.	9. Отображает сумму, которую требуется, вернуть покупателю. Генерирует товарный чек.
10. Кассир кладет полученные от покупателя деньги в кассу и извлекает причитающуюся ему сдачу, Кассир передает покупателю товарный чек и сдачу.	11. Регистрирует завершенную продажу.
12. Покупатель покидает магазин с приобретенными товарами.	

Альтернативы

Строка 2. Введен некорректный идентификатор. Генерируется сообщение об ошибке.

Строка 7. **У** покупателя отсутствует требуемая сумма. Отмена транзакции, связанной с данной продажей.

Анализ развернутого формата

В верхней части развернутой формы содержится обобщенная информация.

Прецедент	Имя прецедента

Исполнители	Перечень исполнителей (внешних агентов), а также тех из них, кто инициирует данный прецедент
Цель	Цель прецедента
Краткое описание	Копия содержимого прецедента высокого уровня или некоторая аналогичная обобщенная информация
Тип	1. Главный, второстепенный или дополнительный (рассматриваются ниже) 2. Идеальный или, реальный (рассматриваются ниже) .
Ссылки	Связанные прецеденты и функции системы

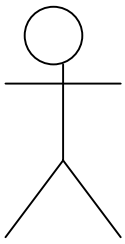
Средний раздел, "Типичный ход событий", является основной частью развернутого формата. В нем представляется содержание подробного, диалога между исполнителями и системой. Важность этого раздела заключается в том, что в нем описывается наиболее стандартная, или типичная, последовательность событий— нечто среднее между видами деятельности и успешным завершением процесса. Альтернативные ситуации в типичную последовательность не включаются.

Типичный ход событий	
Действия исполнителя	Отклик системы
Пронумерованные действия исполнителей	Пронумерованные описания откликов системы

Завершающий раздел, "Альтернативный ход событий", должен содержать важные альтернативы или исключения, которые могут возникать в ходе типичной последовательности. Если эти исключения слишком сложны, они могут быть развернуты в собственные прецеденты.

Исполнители

Исполнитель (actor) является внешним по отношению к системе понятием, которое определенным образом участвует в процессе, описываемом прецедентом. Обычно с помощью входных событий исполнитель стимулирует систему или получает от нее некоторую информацию. Исполнители представляются ролью, которую они играют в прецеденте, например Покупатель, Кассир и т.д. Настоятельно рекомендуется использовать в прецеденте исполнителей, поскольку в этом случае значительно упрощается процесс идентификации



Покупатель

Пиктограмма языка UML, используемая для обозначения исполнителя прецедента

В прецеденте имеется один *инициирующий исполнитель* (initiator actor), который генерирует начальное воздействие, и, возможно, несколько *участвующих исполнителей* (participating actor). Иногда полезно указать, какой из исполнителей является инициирующим.

Обычно исполнители представляются ролями, которые играют люди, однако они могут быть и разновидностью какой-либо системы, такой как внешняя компьютеризованная банковская система. Исполнители могут быть следующих разновидностей.

- Роли, исполняемые людьми
- Компьютерные системы
- Электрические или механические устройства

Стандартные ошибки, возникающие при использовании прецедентов

Стандартной ошибкой является представление в виде прецедентов отдельных шагов, операций или транзакций. Например, анализируя предметную область, связанную с терминалом розничной торговли, можно совершенно неуместно определить прецедент Printing the Receipt **(**Печать товарного чека), тогда как фактически операци**я** печати представляет собой лишь один шаг в большем крупном процессе Buy Items (Покупка товара).

Прецедент является описанием относительно большого, завершенного процесса, в который обычно входит много шагов или транзакций. Как правило, отдельные шаги или виды деятельности в виде прецедента не представляются.

Определение прецедентов

Каждый из последующих шагов определения прецедентов подразумевает использование "атаки мозгового штурма" и анализа имеющихся документов, в которых содержатся спецификации требований.

Один метод, используемый для идентификации прецедентов, основан на анализе исполнителей.

1. Идентифицируйте исполнителей, связанных с системой или организацией.
2. Для каждого исполнителя определите процессы, которые они инициируют или в которых участвуют.

Другой метод основан на анализе событий.

1. Идентифицируйте внешние события, на которые должна реагировать система.
2. Свяжите события с исполнителями и прецедентами.

Для приложения терминала розничной торговли к некоторым возможным исполнителям и

инициируемым ими процессам можно отнести следующие.

Кассир	Регистрация.
	Работа с деньгами
Покупатель	Покупка товаров,
	Возврат товаров

Прецеденты и процессы предметной области

С помощью прецедента описывается некоторый процесс, например обработка деловой информации. *Процесс* (process) от начала и до конца описывает последовательность событий, действий и транзакций, требуемых для достижения какого-либо результата или предоставления некоторого значения организации или исполнителю.

В качестве примера можно привести следующие процессы-

- Получение денег из банкомата.
- Заказ продукции
- Регистрация учебных курсов в школе
- Проверка орфографии в документе, созданном в текстовом процессоре
- Обработка телефонного звонка

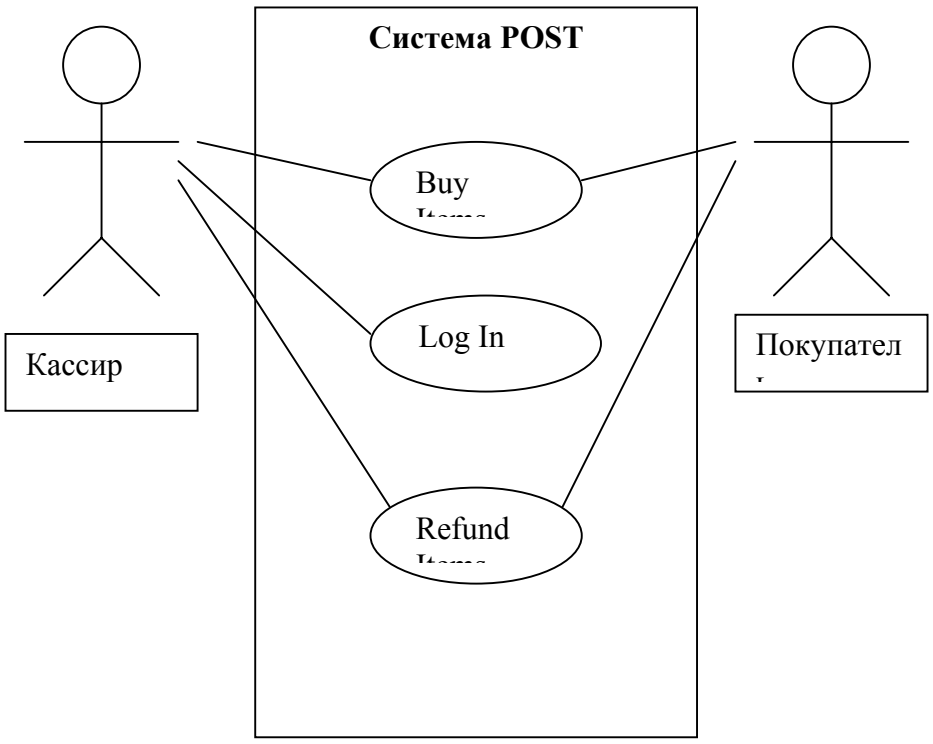
Прецеденты, функции системы и процесс их согласования

При определении функций системы, выполняемом во время формирования предварительных спецификаций требований, все функции должны быть распределены по прецедентам. Это можно проверить, просмотрев раздел *ссылок* (cross references) прецедента. Тем самым между артефактами будет установлена важная связь в терминах. В конечном счете, все функции системы и прецеденты

будут согласованы между собой, что обеспечит возможность их реализации и тестирования.

Диаграммы прецедентов

Для системы розничной торговли пример диаграммы прецедентов представлен на рисунке



Частичная диаграмма прецедентов

На *диаграмме прецедентов* (use case diagram) иллюстрируется набор прецедентов системы и исполнителей, а также взаимосвязи между ними. Прецеденты представляются овалами, а исполнители — условными обозначениями. Между прецедентами и исполнителями имеются линии взаимодействия. Для отображения потоков данных и влияющих объектов могут использоваться стрелки.

Назначение диаграммы - представить некоторую контекстную диаграмму, позволяющую быстро определить внешних исполнителей системы и ключевые методы их использования.

Форматы прецедентов

На практике, прецеденты могут быть выражены, с различной степенью детализации и связей с проектируемым решением. Другими словами, один и тот же прецедент может быть представлен в различных форматах с разными уровнями детализации. *Прецеденты высокого уровня* описывают процессы очень сжато, обычно в двух или трех предложениях. Такой тип описания удобно использовать на начальном этапе формулирования требований к системе для быстрого осознания степени сложности и функций системы. Прецеденты высокого уровня — это лишь краткое описание, имеющее слабое отношение к конкретным проектным решениям.

Развернутый прецедент описывает процесс более детально, чем прецедент высокого уровня. Основной особенностью развернутого прецедента является наличие раздела "Типичный ход событий", в котором описывается последовательность событий. На этапе формулирования требований в развернутом формате целесообразно представлять лишь наиболее важные и значительные прецеденты, а более подробное описание остальных прецедентов отложить до того цикла разработки, в котором они должны быть реализованы.

Прецедент описывает взаимодействие с системой. К числу типичных ограничений системы относятся следующие.

- Ограничения аппаратных средств и программного обеспечения компьютерной системы
- Отдел организации
- Вся организация

Установив ограничения системы, можно идентифицировать ее внешние и внутренние свойства, а также определить обязанности. Внешняя среда представляется лишь исполнителями.

В качестве примера влияния выбора ограничений системы, рассмотрим отдельный кассовый аппарат и

весь магазин. Если в качестве системы выбрать весь магазин, то исполнителем будет выступать покупатель, а не кассир, поскольку кассир — это внутренний ресурс системы, выполняющий определенные задачи. Однако если в качестве системы выбрать аппаратные средства и программное обеспечение кассового аппарата, то исполнителями будут и покупатель, и кассир.

Выбор границ системы определяется целями исследования. Если планируется разработка программного приложения или устройства, то имеет смысл устанавливать границы системы в соответствии с аппаратным и программным обеспечением. Например, терминал торговой точки и его программное обеспечение составляют систему, а покупатель и продавец являются внешними исполнителями.

Основные, второстепенные и дополнительные прецеденты

Прецеденты необходимо ранжировать по трем категориям: основные, второстепенные и дополнительные. Позднее на основе этой классификации можно будет разделить их по приоритету разработки.

Основные прецеденты (primary use cases) представляют самые общие процессы, например покупку товара.

Второстепенные прецеденты (secondary use cases) представляют менее значительные или более редкие процессы, такие как запрос на ассортимент новых товаров.

Дополнительные прецеденты (optional use cases) описывают процессы, которые могут быть не реализованы в системе.

Идеальные и реальные прецеденты

Идеальные прецеденты

Идеальные прецеденты (essential use cases) [10] — это развернутые прецеденты, выражающие общую сущность процесса без детализации их реализации. Проектные решения, особенно связанные с интерфейсом пользователя, при этом опускаются. Идеальный прецедент описывает процесс в терминах наиболее существенных видов деятельности и обоснований. Степень абстракции идеальных прецедентов может варьироваться, т.е. прецедент может быть идеальным в большей или меньшей степени.

Прецеденты высокого уровня всегда идеальны по своей природе из-за краткости и абстрактности.

Рассмотрим пример прецедента Withdraw Cash (Получение денег из банкомата), выраженного в относительно идеальной форме.

Идеальный прецедент	
Действия исполнителя	Отклик системы
Покупатель идентифицирует себя.	2, Представляет перечень возможностей.
И т.д.	4. И т.д.

Конкретная реализация процесса идентификации (проектное решение) будет выполнена позднее, однако это действие является частью абстрактного процесса идентификации личности.

Идеальные прецеденты желательно создавать на ранних стадиях формулирования требовании. Это обеспечит более полное понимание сущности задачи и функций системы. Выделение идеальных прецедентов играет важную роль, поскольку позволяет увидеть суть процессов и понять их основные причины, не перегружая себя деталями проектирования. Кроме того, идеальные прецеденты остаются истинными в течение длительного времени, поскольку они не включают в себя проектные решения и их создание

позволяет глубже понять основные движущие силы происходящих в системе процессов. Организация может повторно использовать одни и те же идеальные прецедента при разработке новых проектов.

Реальные прецеденты

В отличие от идеальных *реальные прецеденты* (real use cases) конкретно описывают процесс в терминах реальных проектных решений, на основе конкретных технологий ввода-вывода информации и т.д. Когда речь идет об интерфейсе пользователя, реальные прецеденты зачастую определяют содержимое диалоговых окон и описывают способы взаимодействия с конкретными устройствами. Вот пример прецедента Withdraw Cash, выраженного в относительно реальной форме.

Реальный прецедент	
Действия исполнителя	Отклик системы
1. Покупатель вставляет свою карточку.	2. Приглашение к вводу информации.
3. Ввод информации с помощью клавиатуры.	4. Отображение пунктов меню.

Абстрактное действие, «Покупатель идентифицирует себя» из идеального прецедента конкретно реализовано в последовательности действий, начиная с «Покупатель вставляет свою карточку» и т.д.

В идеале реальные прецеденты должны создаваться на стадии проектирования, поскольку они составляют артефакты проектирования. В некоторых проектах на ранних стадиях разработки приходится

создавать конкретные проектные решения, относящиеся к интерфейсу пользователя. В таких случаях реальные прецеденты должны создаваться уже на стадии планирования. Однако, реальные прецеденты нежелательно создавать на. стадии планирования, поскольку это может, привести к чрезмерному усложнению системы. Тем не менее, некоторые организации составляют контракты на разработку программного обеспечения на основе спецификации интерфейса пользователя.

Названия прецедентов должны начинаться с глагола или существительного, описывающего процесс, как например, показано ниже**.**

- Buy Items **(**Покупка

  товара**)**

- Enter an Order **(**Ввести заказ**)**

Развернутые прецеденты

Описания развернутых прецедентов следует начинать по следующей схеме.

1. Этот прецедент начинается, когда <Исполнитель> **<**инициирует событие>. Например,
2. Этот прецедент начинается, когда покупатель подходит к системе POST с выбранными товарами

Такой подход позволяет четко идентифицировать исполнителя и событие.

Моменты принятия решений и ветвления

Прецедент может содержать точки принятия решения. Например, в прецеденте Buy Items покупатель может выбрать способ платежа в момент оплаты: наличными, по кредитной карточке или чеком. Если одно из этих решений является типичным, а остальные -редкими, необычными или исключительными, то лишь данный типичный случай

должен быть описан в разделе "Типичный ход событий", а остальные — в разделе "Альтернативы":

Однако в момент принятия решения могут появляться относительно равнозначные и равновероятные альтернативы. Это например, относится к типам платежей: наличными, по кредитной карточке или чеком. В таком случае используется следующая структура описания.

В основном разделе "Типичный ход событий" указываются возможные подразделы. Каждый подраздел описывается как типичный ход событий. Нумерация событий в каждом разделе начинается с единицы. Если подразделы также содержат точки принятия решений, то возможные варианты записываются в разделе "Альтернативы" каждого подраздела.

Главный раздел

Типичный ход событий	
Действия исполнителя	Отклик системы
1. Покупатель подходит к кассовому аппарату с системой POST с выбранными товарами.	
2. (Промежуточные шаги пропускаются.)	
3, Покупатель выбирает тип платежа. а) Если выбрана оплата наличными, см. раздел "Оплата наличными". б) Если выбрана оплата по кредитной карточке, см. раздел "Оплата по кредитной карточке". в) Если выбрана оплата чеком, см. раздел "Оплата чеком",	

	4. Регистрирует выполненную покупку,
	5. Выдает товарный чек.
6. Кассир выдает чек покупателю,	
7. Покупатель покидает магазин с покупками.	

Раздел "Оплата наличными"

Типичный ход событий	
Действия исполнителя	Отклик системы
1. Покупатель выдает сумму наличными, возможно, превышающую стоимость покупки.	
2. Кассир вводит полученную сумму.	3. Отображает сумму сдачи, которую следует возвратить покупателю.
4. Кассир кладет полученные от покупателя деньги в кассу и извлекает причитающуюся ему сдачу, Кассир передает сдачу покупателю.	

Альтернатива

• Строка 4. В кассе недостаточно денег для выдачи сдачи. Обратиться за наличностью в вышестоящую организацию или попросить покупателя поискать более мелкие купюры.

Раздел "Оплата по кредитной карточке"

В этом разделе приводятся типичный и альтернативный ход событий в процессе оплаты по кредитной карточке.

Раздел "Оплата чеком"

В этом разделе приводится типичный и альтернативный ход событий в процессе оплаты чеком.

Процесс разработки прецедентов

Стадии этапа планирования

1. После перечисления функций системы необходимо сначала определить границы системы, а затем идентифицировать исполнителей и прецеденты.

2. Записать все прецеденты в формате высокого уровня. Разделить их по категориям: основные, второстепенные и дополнительные.

3. Построить диаграмму прецедентов.

4. Определить взаимоотношения между прецедентами на диаграмме (взаимоотношения между прецедентами описываются ниже).

5. Выделить наиболее важные и рискованные прецеденты и записать их в развернутом идеальном формате для более глубокого понимания природы и сложности проблемы. Отложить расширенное описание менее важных прецедентов до соответствующих циклов разработки во избежание дополнительных сложностей.

6. В идеале описание реальных прецедентов следует отложить до стадии проектирования цикла разработки, поскольку оно требует принятия проектных решений. Однако иногда необходимо создавать реальные прецеденты на ранних стадиях этапа формулирования требований. Это происходит в следующих случаях.

- Если важное значение имеет конкретное описание процесса
- Если заказчик требует подробную спецификацию на начальном этапе разработки

7. Ранжировать прецеденты

На этапе планирования определяются требования к системе и прецеденты. Кроме того, на данном этапе можно создать приблизительную концептуальную модель и разработать приблизительную архитектуру системы, однако эти виды деятельности будут рассмотрены несколько позже.

Предположим, все необходимые артефакты уже сгенерированы (в том числе определены требования к системе и прецеденты). Следующим шагом должен стать переход к итеративным циклам стадии построения и к началу реализации системы. На этапе построения прецеденты распределяются по многочисленным итеративным циклам разработки.

Рассмотрим вопросы ранжирования прецедентов и составление графика их реализации.

Прецеденты и циклы разработки

Циклы разработки организовываются в соответствии с требованиями прецедентов, т.е. каждый цикл разработки предполагает реализацию одного или нескольких прецедентов либо упрощенных версий прецедентов, если их полные версии являются слишком сложными для реализации в одном цикле.

Прецеденты необходимо ранжировать, чтобы в начальных циклах разработки реализовать наиболее приоритетные прецеденты. Основная стратегия должна заключаться в том, чтобы сначала сконцентрировать внимание на тех прецедентах, которые в значительной мере определяют базовую архитектуру. При этом должны учитываться следующие качества прецедентов.

a) Существенное влияние на архитектуру системы, например добавление многих классов на уровне реализации или необходимость обеспечения постоянно действующих служб.
b) Важная информация и внутренняя структура проекта обеспечиваются за. счет сравнительно небольших усилий

c) Включает рискованные, сложные или срочные функции
d) Требует дополнительного исследования или применения новой и ненадежной технологии
e) Представляет основной экономический процесс
f) Напрямую обеспечивает повышение эффективности или снижение стоимости

В схеме ранжирования можно использовать простые нечеткие оценки, например "высокий", "средний" и "низкий".

Кроме того, при ранжировании можно применять числовые значения (даже взвешенные) и на их основе определять приоритетность прецедента.

Ранжирование прецедентов системы розничной торговли

На основе приведенных выше критериев ранжирования можно провести нечеткое и нестрогое ранжирование прецедентов системы розничной торговли. Данный список не является исчерпывающим.

Ранг.	Прецедент	Обоснование
Высокий	Buy Items (Покупка товара)	Наиболее высокая степень удовлетворения критериям ранжирования
Средний	Add New Users (Добавление новых пользователей)	Влияет на подсистему безопасности
	Log In (Регистрация)	Влияет на подсистему безопасности
	Refund Items (Возврат товара)	Важный процесс; влияет на инвентаризацию
Низкий	Cash Out (Расчет)	Влияние на архитектуру минимальное
.	Start to (Включение)	Определение зависит от других прецедентов

	shut down (Выключение)	Влияние на архитектуру минимально

Начало цикла разработки

Допустим, что этап планирования завершен, прецеденты идентифицированы и ранжированы и график их реализации составлен (по крайней мере, на несколько первых циклов разработки). Теперь необходимо перейти к этапу построения, состоящему in нескольких итеративных циклов разработки. В первом цикле требуется реализовать упрошенную версию прецедента Buy Items (Покупка товара), включающую лишь реализацию оплаты наличными без внесения изменений в базу. данных имеющихся в наличии товаров (без инвентаризации),

Исходные виды деятельности цикла разработки связаны с управлением проектом. В общем случае после этого цикла (а точнее — параллельно с ним) следует дополнение документации (например, диаграмм), созданной в предыдущем цикле разработки, текущей информацией и приведение её в соответствие с текущим состоянием кода, поскольку после написания кода в предыдущем цикле содержание документации и реальное состояние системы существенно различаются.

Затем начинается стадия (фаза) анализа, на которой подробно исследуются задачи текущего цикла разработки. На этой стадии одним из первых видов деятельности является разработка концептуальной модели, которая и будет рассмотрена ниже.

Построение концептуальной модели

Концептуальная модель отображает основные (с точки зрения моделирующего) понятия предметной области. Она является наиболее важным артефактом, создаваемым на этапе объектно-ориентированного анализа.

Основной задачей объектно-ориентированного анализа является идентификация большого количества разнообразных объектов или понятий, а также точная оценка усилий в терминах отдачи на стадиях проектирования и реализации.

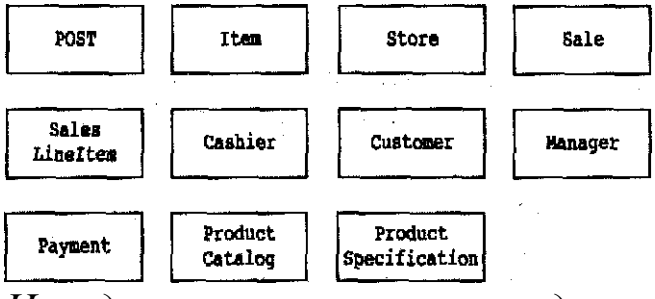
Идентификация понятий — составная часть исследования предметной области.

Важным свойством концептуальных моделей является представление понятий реального мира, а не программных компонентов.

Прецеденты — это важные артефакты этапа анализа требования, но на самом деле они не являются объектно-ориентированными. Они концентрируют внимание разработчиков на процессах, происходящих в предметной области.

Один из первых основных видов деятельности на каждом цикле разработки – это создание концептуальной модели для прецедентов текущего цикла. Процесс создания концептуальной модели может происходить параллельно с разработкой прецедентов.

Представленные выше имена понятий могут быть изображены графически на статической структурной диаграмме в системе обозначений UML.



Исходная концептуальная модель предметной области системы розничной торговли

Атрибуты и ассоциации, связанные с этой концептуальной моделью, будут рассмотрены далее.

Принципы создания концептуальной модели

Как создать концептуальную модель

Для создания концептуальной модели выполните следующие действия.

1. Составьте список понятий-кандидатов на основе списка категорий и метода анализа текстового описания для текущего цикла разработки.
2. Отобразите их в концептуальной модели.
3. Добавьте необходимые ассоциации, отражающие связи, для которых требуется выделение памяти (обсуждается в следующей главе).
4. Добавьте атрибуты, необходимые для выполнения информационных требований (обсуждается в следующей главе).

Имена и модели: стратегия построения карт

При построении концептуальных моделей применяется та же стратегия, что и при создании карт.

Концептуальную модель следует создавать согласно принципам картографии.

• Использовать применяемые на данной территории названия

• Исключать несущественные детали

• Не добавлять объекты, которые отсутствуют на данной территории

Концептуальная модель — это своеобразная разновидность карты понятий некоторой предметной области. Отсюда следуют аналитическая роль концептуальной модели, а также справедливость следующих замечаний.

• Картографы используют названия, применяемые на данной территории. Они не изменяют названия городов на карте. С точки зрения концептуальной модели это означает необходимость использования словаря предметной области при именовании понятий и атрибутов. Например, при разработке модели библиотеки в качестве понятия, означающего

потребителя, следует выбрать Borrower (Читатель), т.е. использовать терминологию библиотекарей.

• Картограф не наносит на карту объекты, не имеющие отношения к основному ее назначению, например не отображает топографию или состав населения. Аналогично в концептуальной модели не должны содержаться понятия из предметной области, не имеющие отношения к требованиям. Например, из нашей концептуальной модели можно исключить понятия Pen (Ручка) и PapегBag (Папка), поскольку они не имеют отношения к требованиям.

• Картограф не отображает на карте отсутствующие объекты, например горы на карте равнинной местности. Точно так, в концептуальной модели не должны содержаться понятия, не имеющие отношения к рассматриваемой проблеме.

Типичная ошибка при выделении понятий

Возможно, наиболее типичной ошибкой при создании концептуальной модели является отнесение некоторого объекта к атрибутам, в то время как он должен относиться к понятиям. Чтобы избежать этой ошибки, следует придерживаться простого правила.

Если некоторый объект X в реальном мире не является числом или текстом, это, скорее всего, понятие, а не атрибут.

Например, рассмотрим предметную область системы резервирования авиабилетов. Нужно ли рассматривать место назначения как атрибут destination понятия Flight (Полет) или как отдельное понятие Airport (Аэропорт)?

В реальном мире аэропорт не является ни текстом, ни числом: это массивный объект, занимающий определенное пространство. Следовательно, в концептуальной

В реальном мире аэропорт не является ни текстом, ни числом: это массивный объект, занимающий определенное пространство. Следовательно, в концептуальной модели он должен быть представлен понятием Airport.

Если у вас возникают сомнения при разграничении понятий и атрибутов, создавайте отдельное понятие.

Разрешение конфликта сходных понятий: POST или Register

В старые времена, задолго до появления кассовых аппаратов, в каждом магазине вели реестр — книгу, в которой регистрировались все продажи и платежи. Со временем этот процесс был автоматизирован с помощью механических кассовых аппаратов. На сегодняшний день роль реестра выполняет терминальная система розничной торговли.

Таким образом, реестр — это объект, в который записываются сведения о продажах и платежах. Такую же функцию выполняет и терминальная система розничной торговли. Однако термин "реестр" является несколько более абстрактным, чем POST. Так, может быть, в концептуальной модели вместо понятия POST следует использовать понятие Register (Реестр)?

Игрок бросает два кубика. Если сумма очков равна семи, участник считается победителем, в противном случае — проигравшим.

При этом будем использовать систему обозначений UML.

Обратите внимание, что в данном примере представлены не все возможные этапы и диаграммы, а лишь наиболее типичные.

Моделирование "нереального" мира

Предметная область некоторых приложений имеет очень слабое отношение к реальному миру. Примерами таких приложений могут быть системы телекоммуникаций. Для подобных систем также можно создавать концептуальные модели, но для этого требуются высокая степень абстракции и отход от стандартных принципов разработки.

Например, в качестве понятий, связанных с предметной областью системы телекоммуникаций, могут выступать Message (Сообщение), Connection (Соединение), Dialog (Диалог), Road (Маршрут) и Protocol (Протокол).

Спецификация или описание понятий

Примем следующие допущения-

• Термин Item (Товар) представляет физический товар в магазине, а значит, он может иметь серийный номер

• Понятию Item соответствуют описание, цена и универсальный код товара, которые более нигде не записываются

• Каждый сотрудник магазина страдает амнезией

• При каждой продаже реального физического товара удаляется соответствующий программный экземпляр Item

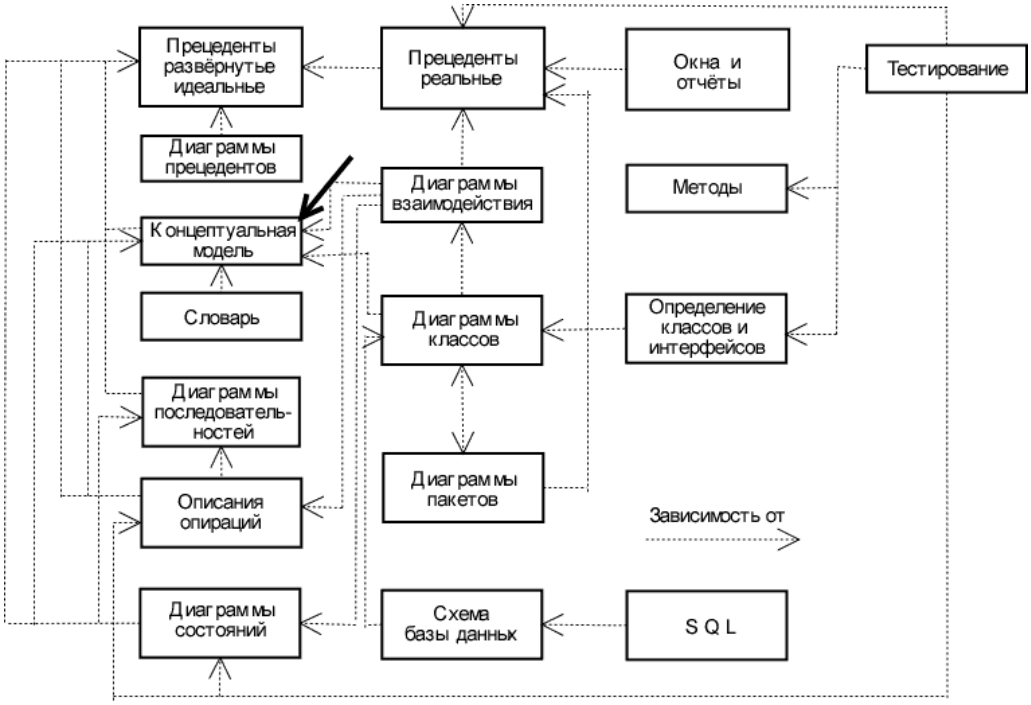
Что произойдет при выполнении следующего сценария при сделанных предположениях?

В магазине повысился спрос на новую разновидность вегетарианских горячих бутербродов ObjectBurger. Идет бойкая торговля, и при продаже каждого объекта ObjectBurger экземпляр Item удаляется из памяти компьютера,

Однако возникает проблема, связанная с тем, что никто не может ответить на вопрос "Сколько стоит ObjectBurger?", поскольку его цена связана с каждым экземпляром, который удаляется из памяти в случае его продажи.

Заметим, что в прежние времена реестр представлял собой лишь одну из возможных реализаций системы записей о проданных товарах. Со временем этот термин приобрел обобщенное значение.

Заметим также, что при программной реализации описанной модели происходит дублирование данных, и память используется неэффективно, поскольку описание, цена и универсальный код товара связаны с каждым экземпляром одного и того же товара.



Зависимости между артефактами на этапе построения.

Концептуальные модели

Основной составляющей объектно-ориентированного анализа или исследования является декомпозиция проблемы на отдельные понятия или объекты. *Концептуальная модель* — это представление понятий в терминах предметной области. На языке UML концептуальная модель представляется в виде набора *статических структурных диаграмм,* на которых не определены никакие операции. Сам термин "концептуальная модель" указывает на строгое соответствие понятиям предметной области, а не программирования.

Концептуальная модель может отображать следующее.

- Понятия
- Ассоциации между понятиями
- Атрибуты понятий

С точки зрения предметной области важными понятиями являются Payment (Платеж) и Sale (Продажа). Эти понятия связаны между собой и понятию Sale соответствуют определенная дата и время.

Концептуальная модель не только предоставляет возможность выполнить декомпозицию проблемы на объекты (понятия), но и помогает сформировать терминологию и

составить словарь терминов предметной области. Этот словарь позволяет разработчиками выделить наиболее важные термины и связи между ними.

Концептуальная модель –это не модель структуры программы

Концептуальная модель — это описание системы в терминах предметной области, а не программных элементов, таких как классы Java или C++ . Следовательно, в концептуальной модели не используются следующие элементы.

Артефакты программирования наподобие окон или базы данных, если только разрабатываемая система не является моделью программного средства, например моделью графического интерфейса пользователя

Обязанности или методы.

Понятия

Понятие — это представление идеи или объекта. Если говорить более строго, то понятие можно рассматривать в терминах символов, содержания и расширения.

- *Символы* (symbol) — слова или образы, представляющие понятия
- *Содержание* (intension) — определение понятия
- *Расширение* (extension) — набор примеров, к которым применимо понятие

Например, рассмотрим понятие события осуществления покупки. Его можно обозначить символом Sale. Содержанием этого понятия является "представление события осуществления покупки в определенный день и определенное время". В качестве расширения можно рассматривать все примеры покупок (другими словами, все множество покупок).

При создании концептуальной модели обычно рассматриваются символьное описание и содержание понятия, поскольку именно они представляют наибольший практический

интерес.

Концептуальные модели и декомпозиция

Программные системы могут быть очень сложными. Поэтому декомпозиция (по принципу "разделяй и властвуй) - это общая стратегия борьбы со сложностью проблемы и се разделение на мелкие составные части. При *структурном подходе* к проектированию систем задача разбивается на процессы или функции, а при объектно-ориентированном — на понятия.

Основное отличие объектно-ориентированного анализа от структурного состоит в декомпозиции проблемы на понятия (объекты), а не на функции.

Следовательно, основной задачей на стадии анализа является идентификация различных понятий из предметной области и. представление результатов в виде концептуальной модели.

Стратегии идентификации понятий

Нашей задачей является создание концептуальной модели, отражающей интересные и важные понятия рассматриваемой предметной области. Поэтому необходимо идентифицировать понятия с использованием двух предлагаемых стратегий.

При идентификации понятий целесообразно руководствоваться следующим принципом.

Лучше излишне детализировать концептуальную модель, чем не доопределить ее.

Не следует думать, что концептуальная модель тем лучше, чем меньше в ней понятий. Истина состоит в обратном.

Зачастую на начальной стадии идентификации некоторые понятия упускаются из виду, а появляются позднее, при рассмотрении атрибутов и ассоциаций, или даже на стадии проектирования. Обнаруженные новые понятия добавляются в концептуальную модель.

Не стоит исключать из словаря понятия только на том основании, что из анализа требований не следует очевидная необходимость его запоминания (этот критерий зачастую применяется при разработке реляционных баз данных, однако

не годится для создания концептуальных моделей) или понятие не имеет атрибутов. Понятия без атрибутов вполне допустимы, как допустимы и понятия, играющие "поведенческую", а не информационную роль в предметной области.