Технологии программирования. Компонентный подход

В. В. Кулямин

Лекция 4. Анализ предметной области и требования к ПО

Аннотация

Рассматриваются вопросы, связанные с анализом предметной области и выделением требований к разрабатываемой программной системе, а также основные графические модели, используемые в этих деятельностях — диаграммы потоков данных и вариантов использования.

Ключевые слова

Анализ предметной области, схема Захмана, модели предметной области, диаграммы потоков данных, диаграммы сущностей и связей, функции ПО, требования к ПО, варианты использования, действующие лица, диаграммы вариантов использования.

Текст лекции

Анализ предметной области

Для того, чтобы разработать программную систему, приносящую реальные выгоды определенным пользователям, необходимо сначала выяснить, какие же задачи она должна решать для этих людей и какими свойствами обладать.

Требования к ПО определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц. Однако сформулировать требования к сложной системе не так легко. В большинстве случаев будущие пользователи могут перечислить набор свойств, который они хотели бы видеть, но никто не даст гарантий, что это — исчерпывающий список. Кроме того, часто сама формулировка этих свойств будет непонятна большинству программистов — могут прозвучать фразы типа «должно использоваться и частотное, и временное уплотнение каналов», «передача клиента должна быть мягкой», «для обычных швов отмечайте бригаду, а для доверительных — конкретных сварщиков», и это еще не самые тяжелые для понимания примеры.

Чтобы ПО было действительно полезным, важно, чтобы оно удовлетворяло реальные потребности людей и организаций, которые часто отличаются от непосредственно выражаемых пользователями желаний. Для выявления этих потребностей, а также для выяснения смысла высказанных требований приходится проводить достаточно большую дополнительную работу, которая называется ***анализом предметной области*** или ***бизнес-моделированием,*** если речь идет о потребностях коммерческой организации. В результате этой деятельности разработчики должны научиться понимать язык, на котором говорят пользователи и заказчики, выявить цели их деятельности, определить набор задач, решаемых ими. В дополнение стоит выяснить, какие вообще задачи нужно уметь решать для достижения этих целей, выяснить свойства результатов, которые хотелось бы получить, а также определить набор сущностей, с которыми приходится иметь дело при решении этих задач. Кроме того, анализ предметной области позволяет выявить места возможных улучшений и оценить последствия принимаемых решений о реализации тех или иных функций.

После этого можно определять область ответственности будущей программной системы — какие именно из выявленных задач будут ею решаться, при решении каких задач она может оказать существенную помощь, и чем именно. А определив задачи ПО в

рамках общей системы задач и деятельностей пользователей, можно уже более точно сформулировать требования к нему.

Анализом предметной области занимаются ***системные аналитики*** или ***бизнес-аналитики***, которые передают полученные ими знания другим членам проектной команды, сформулировав их на более понятном разработчикам языке. Для передачи этих знаний обычно служит некоторый набор моделей, в виде графических схем и текстовых документов.

Анализ деятельности крупной организации, такой как банк с сетью региональных отделений, нефтеперерабатывающий завод или компания, производящая автомобили, дает огромные объемы информации. Из этой информации надо уметь отбирать существенную, а также надо уметь находить в ней пробелы — области деятельности, полученной по которым информации недостаточно для четкого представления о решаемых задачах. Значит, всю получаемую информацию надо каким-то образом систематизировать. Для систематизации сбора информации о больших организациях и дальнейшей разработки систем, поддерживающих их деятельность, применяется *схема Захмана* (автор — John Zachman, [1,2]) или *архитектурная схема предприятия (enterprise architecture framework)*.

	Мотивация	Люди	Данные	Функции	Место	Время
Контекст	Цели и стратегия бизнеса	Важные для бизнеса организации	Вещи, значимые для бизнеса	Основные бизнес-процессы	География бизнеса	События и периоды, важные для бизнеса
Модель бизнеса	Бизнес план, частные цели и стратегии	Модели потоков работ	Семантические модели Бизнес-сущности и их связи	Модели бизнес-процессов	Система логистики	Базовый график работ
Системная модель	Модель бизнес-правил	Архитектура пользовательского интерфейса	Концептуальная модель данных	Архитектура приложений	Архитектура распределенной системы	Структура обработки событий
Технологическая модель	Модель правил обработки событий	Архитектура представления	Физическая модель данных	Архитектура программно-аппаратной системы	Технологическая архитектура	Структура циклов управления
Детальное представление	Спецификации правил работы системы	Спецификации ролей и прав доступа	Спецификации форматов данных	Код программных компонентов	Спецификации архитектуры сети	Спецификации обработки событий и прерываний
Работающая организация	Стратегия и тактика	Структура организации	Данные	Выполняемые функции	Географическое расположение и сети	Планы

Таблица 1. Схема Захмана. Приведены примеры моделей для отдельных клеток.

В основе схемы Захмана лежит следующая идея: деятельность даже очень большой организации можно описать, используя ответы на простые вопросы — зачем, кто, что, как, где и когда, и разные уровни рассмотрения. Обозначенные 6 вопросов определяют 6 аспектов рассмотрения.

- Цели организации и базовые правила, по которым она работает.
- Персонал, подразделения и другие элементы организационной структуры, связи между ними.
- Сущности и данные, с которыми имеет дело организация.

- Выполняемые организацией и различными ее подразделениями функции и операции над данными.
- Географическое распределение элементов организации и связи между географически разделенными ее структурами.
- Временные характеристики и ограничения на деятельность организации, значимые для ее деятельности события.

Также выделены несколько уровней рассмотрения, из которых при бизнес-моделировании особенно важны три верхних.

- Самый крупный — уровень организации в целом, рассматриваемой в ее развитии совместно с окружением, уровень планирования ее деятельности в целом. Этот уровень содержит долговременные цели и задачи организации как целого, основные связи организации с внешним миром и основные виды ее деятельности.
- Уровень бизнеса, на котором организация рассматривается во всех аспектах как отдельная сущность, имеющая определенную структуру, соответствующую ее основным задачам.
- Системный уровень, на котором определяются концептуальные модели всех аспектов организации, без привязки к конкретным их воплощениям и реализациям, например, логическая модель данных в виде набора сущностей и связей между ними, логическая архитектура системы автоматизации в виде набора узлов, с привязанными к ним функциями и пр.

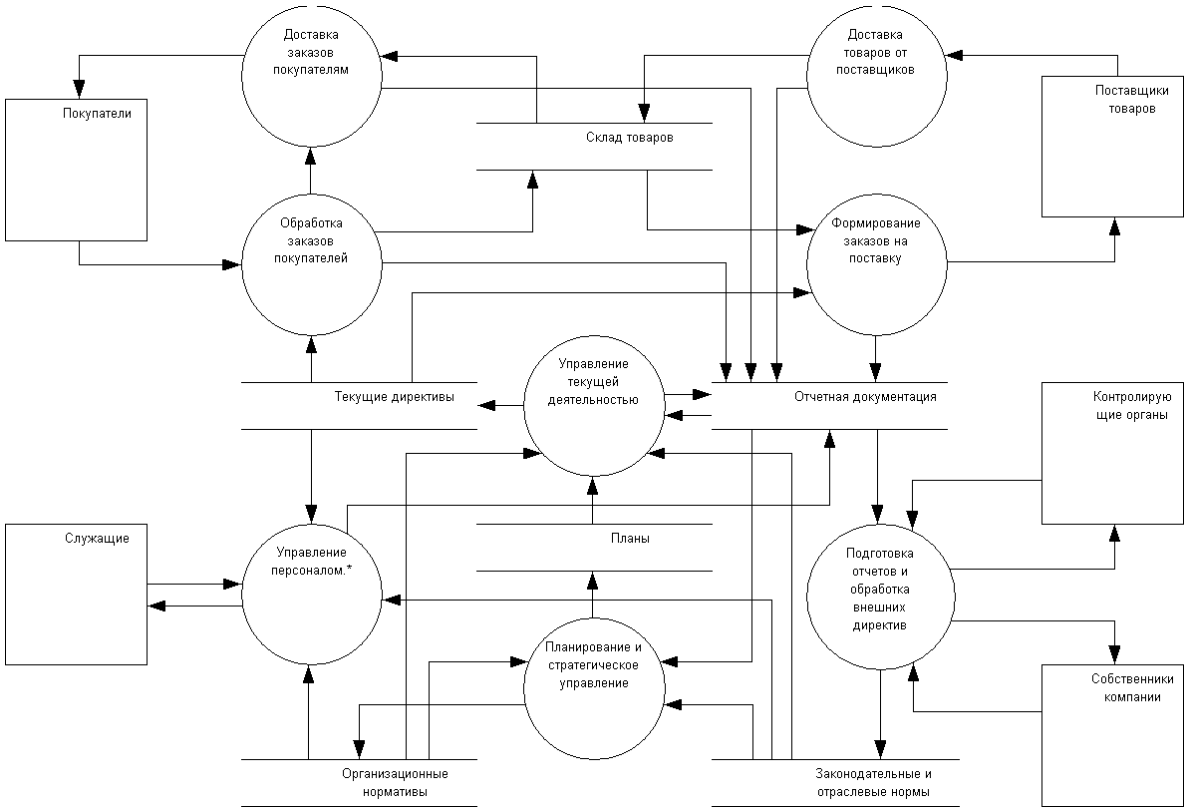

Рисунок 1. Схема деятельности компании, управляющей магазином, в нотации Йордана-ДеМарко.

Наиболее удобной формой представления информации при анализе предметной области являются графические диаграммы различного рода. Они позволяют достаточно быстро зафиксировать полученные знания (набросать рисунок из прямоугольников и связывающих их стрелок обычно можно гораздо быстрее, чем записать соответствующий

объем информации), быстро восстановить их в памяти (на рисунке за один взгляд видно гораздо больше, чем в тексте) и успешно объясняться с заказчиками и другими заинтересованными лицами (хотя изредка встречаются люди, лучше ориентирующиеся в текстах и более адекватно их понимающие, чаще рисунки все же более удобны для пояснения мыслей и объяснения сложных вещей).

Часто для описания поведения сложных систем и деятельности крупных организаций используются ***диаграммы потоков данных (data flow diagrams)***. Эти диаграммы содержат 4 вида графических элементов: *процессы*, представляющие собой любые трансформации данных в рамках описываемой системы, *хранилища данных*, *внешние* по отношению к системе *сущности* и *потоки данных* между элементами трех предыдущих видов.

Используются несколько систем обозначений для перечисленных элементов, наиболее известны нотация Йордана-ДеМарко (Yourdon-DeMarco, [3,4]) и нотация Гэйна-Сарсона (Gane-Sarson, [5]), обе предложенные в 1979 году. Рис. 1 показывает диаграмму потоков данных, описывающую деятельность компании, управляющей небольшим магазином, в нотации Йордана-ДеМарко (процессы изображаются кружками, внешние сущности — прямоугольниками, а хранилища данных — двумя горизонтальными параллельными линиями), на Рис. 2 изображена та же диаграмма в нотации Гейна-Сарсона (на ней процессы — прямоугольники со скругленными углами, внешние сущности — прямоугольники с тенью, а хранилища данных — вытянутые горизонтально прямоугольники без правого ребра).

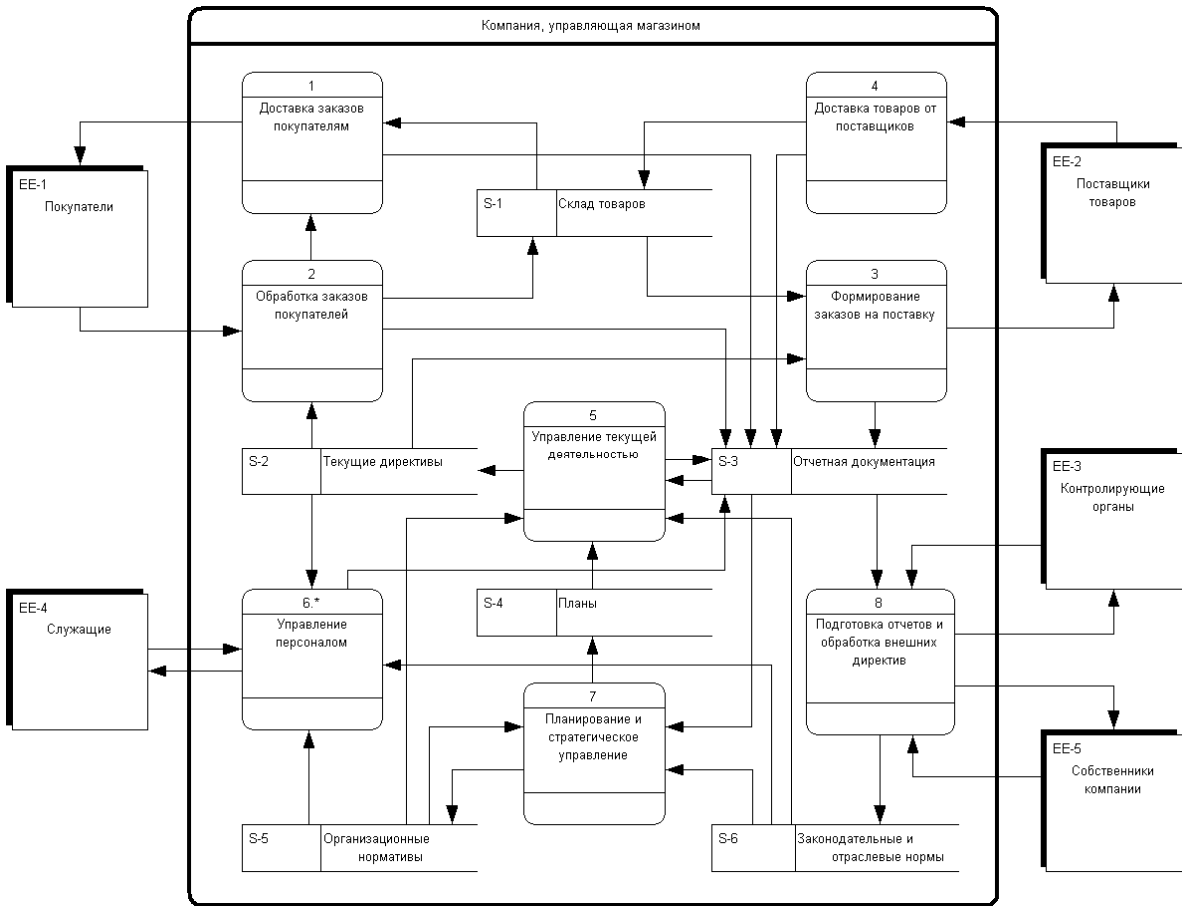

Рисунок 2. Схема деятельности компании, управляющей магазином, в нотации Гэйна-Сарсона.

Процессы на диаграммах потоков данных могут уточняться — если некоторый процесс устроен достаточно сложно, для него можно нарисовать отдельную диаграмму, описывающую потоки данных внутри этого процесса. На ней показываются те элементы, с которыми это процесс связан потоками данных и составляющие его более мелкие

процессы и хранилища. Таким образом, возникает иерархическая структура процессов. Обычно на самом верхнем уровне находится один процесс, представляющий собой систему в целом, и набор внешних сущностей, с которыми она взаимодействует.

На Рис. 3 показана возможная детализация процесса «Управление персоналом».

Диаграммы потоков данных появились как один из первых инструментов представления деятельности сложных систем в рамках разнообразных методик структурного анализа. Для представления структуры данных в рамках тех же подходов используются ***диаграммы сущностей и связей (entity-relationship diagrams, ER diagrams)*** [6], изображающие набор *сущностей* предметной области и *связей* между ними. И сущности, и связи на таких диаграммах могут иметь атрибуты. Пример такой диаграммы представлен на Рис. 4.

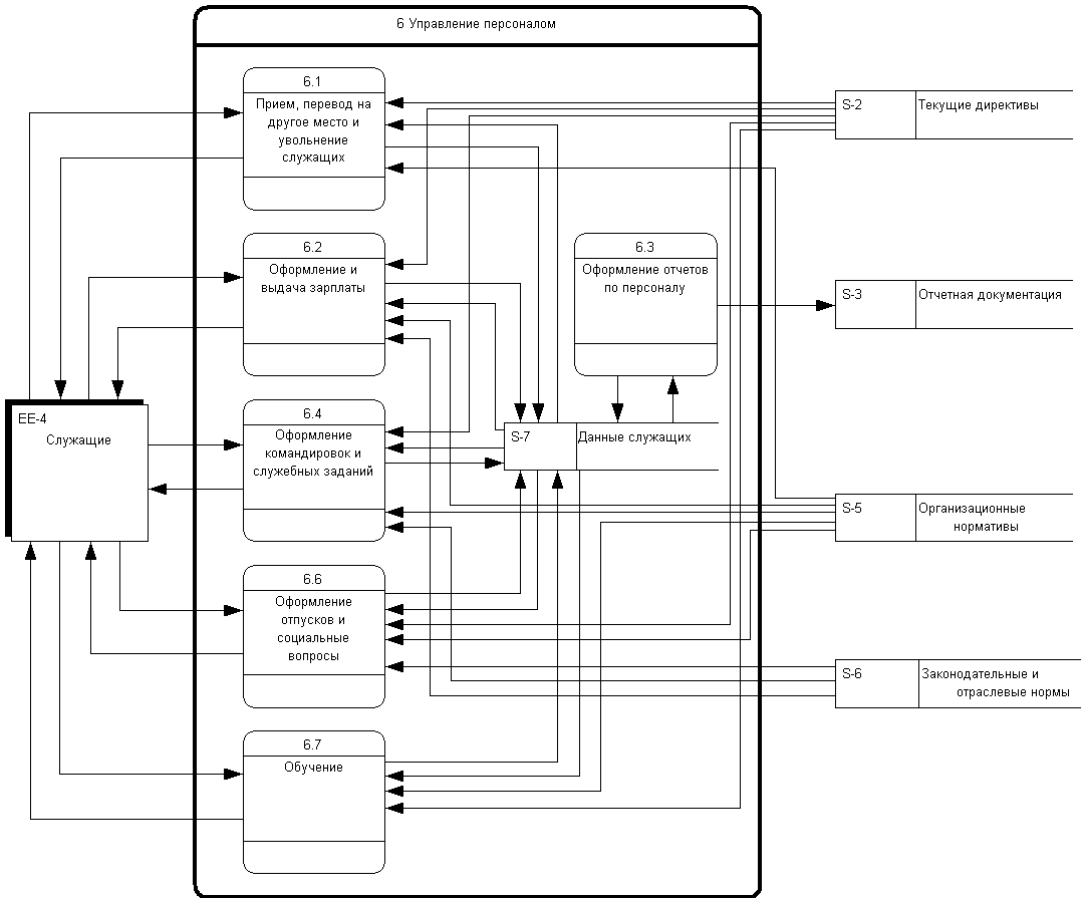

Рисунок 3. Детализация процесса "Управление персоналом".

Хотя методы структурного анализа могут значительно помочь при анализе систем и организаций, дальнейшая разработка ПО, поддерживающего их деятельность, с использованием объектно-ориентированного подхода часто требует дополнительной работы по переводу полученной информации в объектно-ориентированные модели.

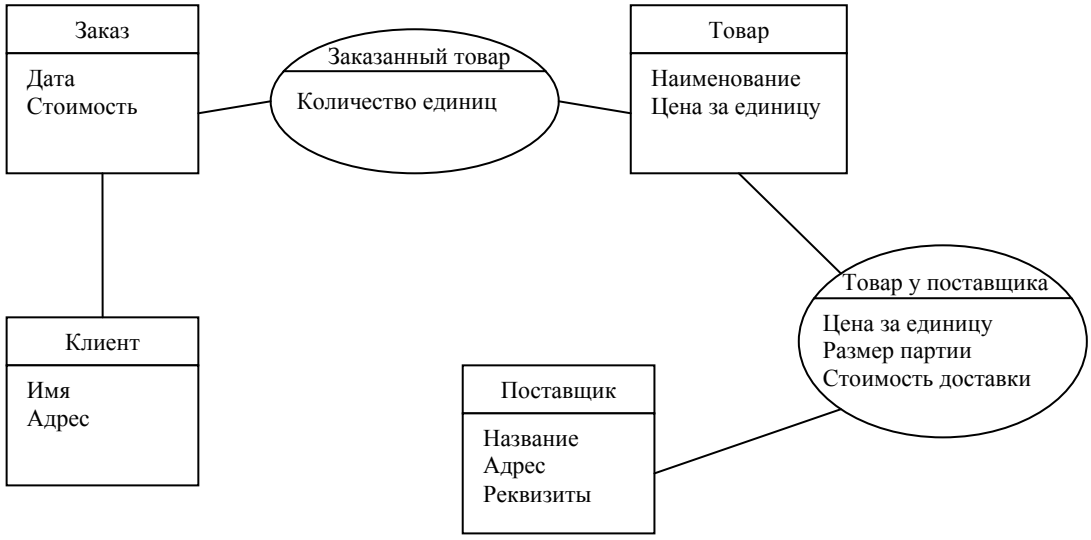

Рисунок 4. Модель сущностей и связей.

Методы объектно-ориентированного анализа предназначены для обеспечения более удобной передачи информации между моделями анализируемых систем и моделями разрабатываемого ПО. В качестве графических моделей в этих методах вместо диаграмм потоков данных используются рассматривавшиеся при обсуждении RUP диаграммы вариантов использования, а вместо диаграмм сущностей и связей — диаграммы классов. Однако диаграммы вариантов использования несут несколько меньше информации по сравнению с соответствующими диаграммами потоков данных — на них процессы и хранилища в соответствии с принципом объединения данных и методов их обработки объединяются в варианты использования, и остаются только связи между вариантами использования и действующими лицами (аналогом внешних сущностей). Для представления остальной информации каждый вариант использования может дополняться набором разнообразных диаграмм UML — диаграммами деятельностей, диаграммами сценариев, и пр. Обо всех этих видах диаграмм будет рассказано в лекции, посвященной архитектуре программного обеспечения.

Выделение и анализ требований

После получения общего представления о деятельности и целях организаций, в которых будет работать будущая программная система, о ее предметной области, можно определить более четко, какие именно задачи система будет решать. Кроме того, важно понимать, какие из задач стоят наиболее остро и обязательно должны быть поддержаны уже в первой версии, а какие могут быть отложены до следующих версий или вообще вынесены за рамки области ответственности системы. Эта информация выявляется при анализе потребностей возможных пользователей и заказчиков.

Потребности определяются на основе наиболее актуальных проблем и задач, которые пользователи и заказчики видят перед собой. При этом требуется аккуратное выявление значимых проблем, определение того, насколько хорошо они решаются при текущем положении дел, и расстановка приоритетов при рассмотрении недостаточно хорошо решаемых, поскольку чаще всего решить сразу все проблемы невозможно.

Формулировка потребностей может быть разбита на следующие этапы.

1. Выделить одну-две-три основных проблемы.
2. Определить причины возникновения проблем, оценить их степень влияния и выделить наиболее существенные из проблем, влекущие появление остальных.
3. Определить ограничения на возможные решения.

Формулировка потребностей не должна накладывать лишних ограничений на возможные решения, удовлетворяющие им. Нужно попытаться сформулировать, что именно является проблемой, а не предлагать сразу возможные решения.

Например, формулировки «система должна использовать СУБД Oracle для хранения данных», «нужно, чтобы при вводе неверных данных раздавался звуковой сигнал» не очень хорошо описывают потребности (за исключением особых случаев, например, если СУБД Oracle уже используется для хранения других данных, которые должны быть интегрированы с рассматриваемыми, при этом ее использование становится внешним ограничением). Соответствующие потребности лучше описать так: «нужно организовать надежное и удобное для интеграции с другими системами хранение данных», «необходимо предотвращать попадание некорректных данных в хранилище».

При выявлении потребностей пользователей анализируются модели деятельности пользователей и организаций, в которых они работают, для выявления проблемных мест. Также используются такие приемы, как анкетирование, демонстрация возможных сеансов работы будущей системы, интерактивные опросы, где пользователям предоставляется возможность самим предложить варианты внешнего вида системы и ее работы или поменять предложенные кем-то другим, демонстрация прототипа системы и др.

После выделения основных потребностей нужно решить вопрос о разграничении области ответственности будущей системы, т.е. определить, какие из потребностей надо пытаться удовлетворить в ее рамках, а какие — нет.

На основе выделенных потребностей пользователей, отнесенных к области ответственности системы, формулируются возможные ***функции*** будущей системы, представляющие собой услуги, предоставляемые системой и удовлетворяющие потребности одной или нескольких групп пользователей (или других заинтересованных лиц). Идеи для определения таких функций можно брать из имеющегося опыта разработчиков (наиболее часто используемый источник) или из результатов мозговых штурмов и других форм выработки идей.

Формулировка функций должна быть достаточно короткой, ясной для пользователей, без лишних деталей. Например:

- Все данные о сделках и клиентах будут сохраняться в базе данных.
- Статус выполнения заказа клиент сможет узнать через Интернет.
- Система будет поддерживать до 10000 одновременно работающих пользователей.
- Расписание проведения ремонтных работ будет строиться автоматически.

Предлагая те или иные функции нужно уметь аккуратно оценивать их влияние на структуру и деятельность организаций, в рамках которых будет использоваться ПО. Это можно сделать, имея полученные при анализе предметной области модели их текущей деятельности.

Имея набор функций, достаточно хорошо поддерживающий решение наиболее существенных задач, с которыми придется работать разрабатываемой системе, можно составлять требования к ней, представляющие собой детализацию работы этих функций. При этом надо учитывать, что часто ПО является частью программно-аппаратной системы, требования к которой надо преобразовать в требования к программной и аппаратной ее составляющим (в последнее время в связи со значительным падением цен на мощное аппаратное обеспечение общего назначения фокус внимания переместился, в основном, на программное обеспечение). Соотношение между проблемами, потребностями, функциями и требованиями изображено на Рис. 5.

Каждое требование раскрывает детали поведения системы при выполнении ею некоторой функции в некоторых обстоятельствах. При этом часть требований происходит из потребностей и пожеланий заинтересованных лиц и решений, удовлетворяющих эти

потребности и пожелания, а часть — из внешних ограничений, накладываемых на систему, например, основными законами той предметной области, в рамках которой системе придется работать, государственным законодательством, корпоративной политикой и пр.

Рисунок 5. Соотношение между проблемами, потребностями, функциями и требованиями.

Еще до перехода от функций к требованиям полезно расставить приоритеты и оценить трудоемкость их реализации и рискованность. Это позволит отказаться от реализации наименее важных и наиболее трудоемких, не соответствующих бюджету проекта функций еще до их детальной проработки, а также выявить возможные проблемные места проекта — наиболее трудоемкие и неясные из вошедших в него функций.

Правила работы с требования к ПО и более общими системными требованиями (к программно-аппаратной системе), определяются следующими двумя стандартами IEEE.

- **IEEE 830-1998 Recommended Practice for Software Requirements Specifications** [7]**.** Описывает структуру документов для фиксации требований к ПО.
  Кроме того, он определяет характеристики, которыми должен обладать правильно составленный набор требований.
  - Корректность или адекватность (соответствие реальным потребностям).
  - Недвусмысленность (однозначность понимания).
  - Полнота (отражение всех выделенных потребностей и всех возможных ситуаций, в которых придется работать системе).
  - Непротиворечивость (согласованность между различными элементами).
  - Упорядоченность по важности и стабильности.
  - Проверяемость (выполнение каждого требования нужно уметь проверять некоторым достаточно эффективным способом — непроверяемые требования должны быть удалены из рассмотрения или сведены к проверяемым вариантам).
  - Модифицируемость (оформление в удобных для внесения изменений структуре и стилях).
  - Прослеживаемость в ходе разработки (возможность увязать требование с подсистемами, модулями и операциям, ответственными за его выполнение, и с тестами, проверяющими его выполнение).
- **IEEE 1233-1998, 2002 Guide for Developing System Requirements Specifications** [8]**.** Описывает правила построения требований для программно-аппаратных систем в целом. Он выделяет следующие необходимые свойства набора требований.
  - Однократное упоминание отдельных требований.

- Отсутствие пересечений между требованиями.
- Явное указание связей между требованиями.
- Полнота.
- Непротиворечивость.
- Определение ограничений, области действия и контекста для каждого требования.
- Модифицируемость.
- Конфигурируемость, удобство поддержки.
- Подходящий для определения системы уровень абстракции.

Кроме того, следующие свойства считаются необходимыми для отдельного требования.

- Абстрактность — формулировка, независимая от возможных реализаций.
- Недвусмысленность.
- Прослеживаемость.
- Проверяемость.

Стандарт предписывает определять следующие атрибуты для каждого требования.

- Уникальный идентификатор.
- Приоритет, важность реализации с точки зрения пользователей.
- Критичность для построения и успешности системы с точки зрения аналитиков.
- Осуществимость с точки зрения готовности пользователей к новой функции, имеющихся технологий и стоимости реализации.
- Риски высокой стоимости, последствий использования для окружающей среды и пользователей, конфликтов со стандартами и законодательством.
- Источник (т.е. кто предложил это требование).
- Тип требования. Возможные типы определяются так (многие из них соответствуют атрибутам качества, рассматриваемым в следующей лекции):
  - Требования на входные данные.
  - Требования на выходные данные.
  - Надежность (reliability, например, среднее время работы между отказами).
  - Работоспособность (availability, например, необходимое отношение времени функционирования к полному времени работы).
  - Удобство сопровождения (maintainability, например, удобство замены компонента).
  - Производительность (performance, например, среднее время ожидания ответа).
  - Доступность (accessibility, например, разные способы доступа для новичков и опытных пользователей).
  - Ограничения окружающей среды (например, максимальный уровень задымленности, при котором гарантируется работоспособность).
  - Эргономичность (ergonomic, например, использование набора цветов, понижающих уставание глаз).
  - Безопасность (safety, например, допустимые уровни электромагнитного излучения различных частот).
  - Защищенность (security, например, ограничения доступа для разных пользователей).

- Требования к оборудованию (например, использование обычной электросети).
- Транспортируемость (transportability, например, ограничения веса).
- Удобство обучения (например, включение обучающих материалов).
- Документированность (например, наличие встроенной документации).
- Внешние интерфейсы (например, поддержка стандартных форматов документов).
- Тестируемость (например, поддержка удаленной диагностики).
- Условия необходимого качества (например, максимально допустимая погрешность производимых измерений).
- Следование корпоративным и законодательным нормам (например, законам об охране труда).
- Совместимость с известными системами.
- Следование стандартам и технологическим нормам.
- Конвертация данных (например, из старой версии системы).
- Возможности роста (например, возможное увеличение числа пользователей).
- Удобство развертывания (например, время, необходимое для приведения в работоспособное состояние).

В дополнение к перечисленному стандарт IEEE 1233 выделяет следующие ошибки, которых необходимо избегать при определении требований.

- Описание возможных решений вместо требований. Эта информация важна, но должна оформляться в отдельных документах.
- Слишком детальные спецификации, описывающие требования к слишком мелким элементам системы или описывающие требования, в точности соответствующие характеристикам определенных систем.
- Слишком сильные ограничения, не вытекающие из реальных потребностей.
- Нечеткие требования, которые могут быть непроверяемыми и субъективными («минимизировать уровень погрешности», «удобный для пользователей интерфейс»), или сформулированы в виде, открытом для пополнения неопределенными элементами (с указанием «и т.д.» или «включая, но не ограничиваясь следующим...»).
- Несформулированные предположения о режимах работы, свойствах окружения, о готовности других систем или принятии законов и стандартов, находящихся в стадии разработки.

Наиболее широко распространенными техниками фиксации требований в настоящий момент являются структурированные текстовые документы и диаграммы *вариантов использования*, о которых уже заходила речь при обсуждении RUP.

Вариантом использования (use case) называют некоторый сценарий действий системы, который обеспечивает ощутимый и значимый для ее пользователей результат. На практике в виде одного варианта использования оформляется сценарий действий системы, который будет, скорее всего, неоднократно возникать во время ее использования и имеет достаточно четко определенные условия начала выполнения и завершения.

Примеры вариантов использования:

- Покупатель в Интернет-магазине выбирает товар. Для этого он может выбрать категорию товара, фирму-изготовителя или группу таких фирм и отфильтровать оставшиеся товары по цене, габаритам и цвету. Определившись, он выбирает товар, кликая на соответствующем значке мышкой.

- Оператор системы контроля качества газопровода ищет участки газопровода с повышенным риском возникновения аварии. Для этого он выбирает группу ранее случившихся аварий, фильтруя их по дате, нанесенному ущербу, типу аварии и запускает процедуру анализа характеристик соответствующих участков газопровода на совпадение, по крайней мере, двух характеристик (учитываются изготовитель труб и их партия, история хранения труб на складах, землепроходческая бригада, бригада сварщиков, показатели нескольких последних проведенных инспекций, показатели химической активности грунтов, наличие близлежащих предприятий, влияющих на химические и электрические характеристики грунтов). После этого на карте выделяются участки, характеристики которых также попадают под найденный «шаблон аварии».

В языке UML вариант использования изображается в виде овала, помеченного именем представляемого варианта использования. Варианты использования могут быть связаны с участвующими в них ***действующими лицами (actors)***, изображаемыми в виде человечков и представляющими различные роли пользователей системы или внешние системы, взаимодействующие с ней.

Варианты использования могут быть связаны друг с другом тремя видами связей: *обобщением (generalization)*, *расширением (extend relationship)* и *включением (include relationship)*. Действующие лица также могут быть связаны друг с другом с помощью связей *обобщения (generalization)*.

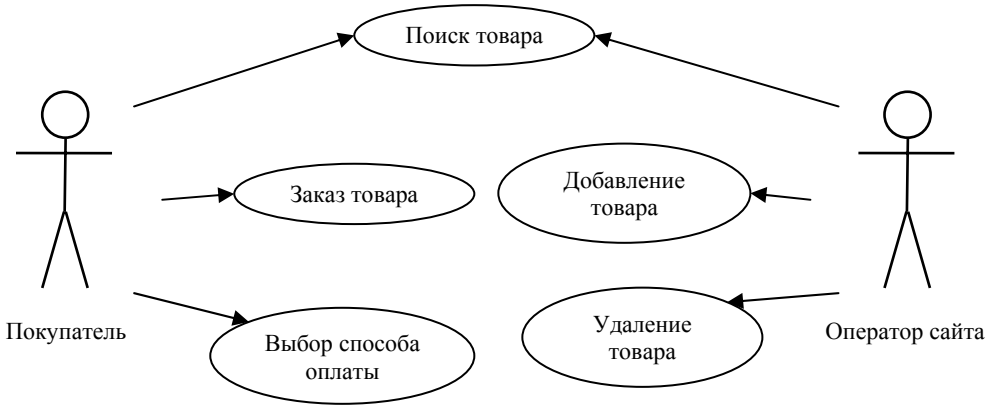

Рисунок 6. Набросок диаграммы вариантов использования, полученных на первых стадиях анализа, для простого Интернет-магазина.

Если несколько вариантов использования имеют много общего в структуре выполняемых в их рамках сценариев и достигаемых целях, можно выделить ***обобщающий*** их вариант использования, содержащий общие части описываемого ими поведения. При этом в общем случае сценарий работы обобщаемого варианта состоит из нескольких кусков — последовательности действий, выполняемых в рамках сценария работы общего варианта использования, перемежаются с последовательностями, специфическими для частного. Например, если система регистрации заказов в магазине позволяет оформить заказ (данные о котором в дальнейшем будут присутствовать в системе) как при помощи сайта магазина, так и по телефону, то варианты использования «Заказ товара через сайт» и «Заказ товара по телефону» могут быть обобщены в варианте «Заказ товара».

Вариант использования A ***расширяет (extends)*** другой вариант использования B, если в ходе сценария работы A при определенных условиях надо включить полный сценарий работы B. Например, оператор сайта магазина может удалить товар, введя его идентификатор; а если идентификатор ему не известен, а известна лишь марка товара и производитель, он должен сначала найти такой товар и определить идентификатор в его

описании, а затем уже удалить товар. Соответственно, вариант использования «Удаление товара» будет расширять вариант использования «Поиск товара».

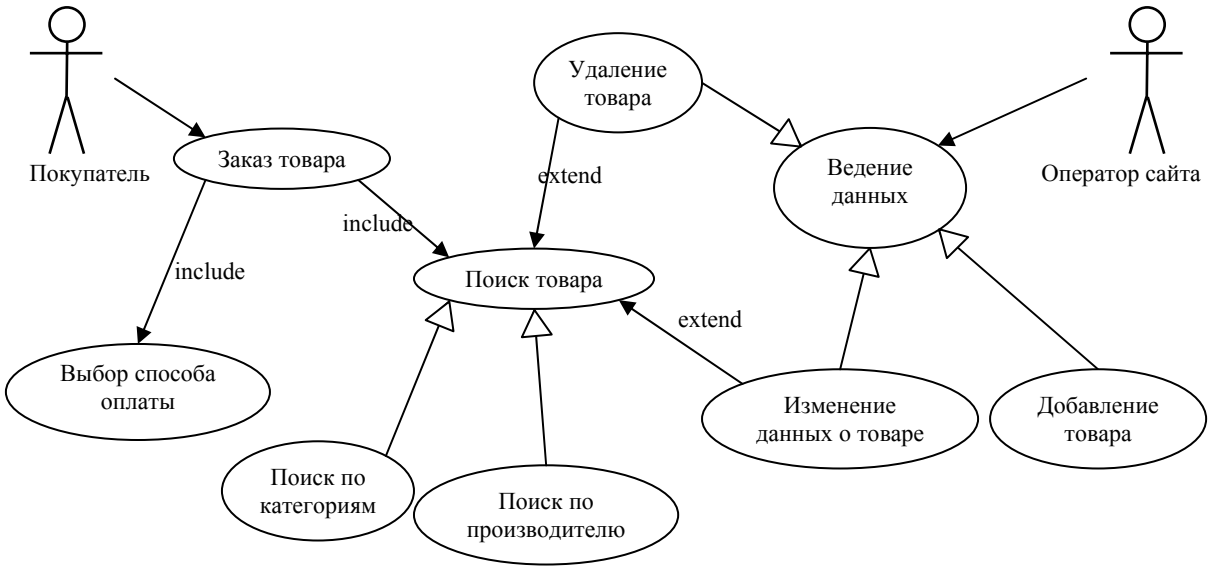

Рисунок 7. Доработанная диаграмма вариантов использования для простого Интернет-магазина.

Вариант использования A ***включает (includes, или использует, uses)*** вариант использования B, если A всегда в некоторый момент включает полностью сценарий работы B. Например, при оформлении заказа покупатель всегда должен определить способ его оплаты. Значит, вариант использования «Заказ товара» включает вариант «Определение способа оплаты».

Обобщение между действующими лицами вводится, если задачи, решаемые одним действующим лицом с помощью данной системы, являются подмножеством задач, решаемых другим действующим лицом. Например, обычный оператор сайта может иметь права только на внесение дополнений и изменений в данные, но не иметь прав на приостановку работы сайта и изменение структуры, которые имеет администратор сайта. В то же время администратор может делать все, что может обычный оператор сайта. Соответственно, администратор сайта является специальным частным случаем оператора.

Хорошо описанный вариант использования имеет следующие атрибуты [9].

1. Имя, ясно говорящее о назначении варианта использования.
2. Описание. Несколько предложений, описывающих этот вариант использования.
3. Частота. Насколько часто данный вариант использования возникает.
4. Предусловия. Все условия запуска варианта использования.
5. Постусловия. Все условия, которые должны быть выполнены после успешного выполнения варианта использования.
6. Основной сценарий работы, который используется в большинстве случаев.
7. Альтернативные сценарии, возникающие иногда. Для каждого альтернативного сценария указываются условия его запуска.
8. (Необязательно) Задействованные действующие лица.
9. (Необязательно) Расширяемые варианты использования.
10. (Необязательно) Включаемые варианты использования.
11. (Необязательно) Статус — в разработке, готов к проверке, в процессе проверки, подтвержден, отвергнут.
12. (Необязательно) Допущения об окружении и ходе работы системы, использованные при разработке варианта использования. В полностью готовом

варианте эти допущения либо должны быть подтверждены и стать ограничениями системы, либо давать начало различным сценариям работы.

Кроме того, варианты использования могут дополняться диаграммами других видов — прежде всего, сценарными диаграммами и диаграммами активностей, описывающими последовательности действий участвующих компонентов, диаграммами состояний и переходов компонентов и диаграммами классов этих компонентов, и др. Все эти виды диаграмм будут рассматриваться в лекции, посвященной архитектуре ПО.

Литература к Лекции 4

[1] J. A. Zachman. A Framework for Information Systems Architecture. IBM Systems Journal, vol. 26, no. 3, pp. 276-292, 1987.

[2] J. F. Sowa and J. A. Zachman. Extending and Formalizing the Framework for Information Systems Architecture. IBM Systems Journal, vol. 31, no. 3, pp. 590-616, 1992.

[3] E. Yourdon. Modern Structured Analysis. Prentice Hall, 1988.

[4] T. DeMarco. Structured Analysis and System Specification. A Yourdon Book, Yourdon Inc., NY, 1979.

[5] C. Sarson, T. Gane. Structured Systems Analysis. Englewood Cliffs, NJ.: Prentice-Hall, 1979.

[6] P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems I (I). March 1976, pp. 8-46.

[7] IEEE 830-1998. Recommended Practice for Software Requirements Specifications. New York: IEEE, 1998.

[8] IEEE 1233-1998. Guide for Developing System Requirements Specifications. New York: IEEE, 1998.

[9] А. Коберн. Современные методы описания требований к системам. М.: Лори, 2002.

[10] И. Соммервилл. Инженерия программного обеспечения. М.: Вильямс, 2002.

[11] Э. Дж. Брауде. Технология разработки программного обеспечения. СПб.: Питер, 2004.

[12] Д. Леффингуэлл, Д. Уидриг. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. М.: Вильямс, 2002.

[13] А. Якобсон, Г. Буч, Дж. Рамбо. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002.

Задания к Лекции 4