ВВЕДЕНИЕ В ПРОГРАММНУЮ ИНЖЕНЕРИЮ

Программная инженерия (технология программирования) -дисциплина, целью которой является сокращение стоимости и сроков разработки программ.

Каждый этап развития программной инженерии связан с появлением (или осознанием) очередной проблемы и нахождением путей и способов решения этой проблемы

- Проблема: высокая стоимость программ была связана с разработкой одинаковых (или похожих) фрагментов кода в различных программах.
- Модульное программирование. Главный принцип состоял в выделении таких фрагментов и оформлении их в виде модулей. Каждый модуль снабжался описанием, в котором устанавливались правила его использования – интерфейс модуля.

- Проблема: переход от разработки относительно простых программ к разработке сложных программных комплексов (системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д.) Сложность таких комплексов оценивалась следующими показателями: большой объем кода (миллионы строк), большое количество связей между элементами кода, разработчиков (сотни человек), количество пользователей (сотни и тысячи), длительное время использования.
- Для таких сложных программ оказалось, что основная часть их стоимости приходится не на создание программ, а на их внедрение и эксплуатацию
- Структурное программирование. Этап сопровождения программного комплекса включал действия по исправлению ошибок в работе программы и внесению изменений в соответствии с изменившимися требованиями пользователей.

Основные принципы технологии структурного проектирования и кодирования:

- ✔ Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т.д. (принцип «разделяй и властвую»)
- ✔ Применение специальных языков проектирования и средств автоматизации использования этих языков
- ✔ Дисциплина проектирования и разработки: планирование и документирование проекта, поддержка соответствие кода проектной документации

- Проблема. Изменение требований к программе стали возникать не только на стадии сопровождения, но и на стадии проектирования – проблема заказчика, который не знает, что он хочет. Возник вопрос как проектировать и писать программы, чтобы обеспечить возможность внесений изменений в программу, не меняя ранее написанного кода.
- Объектно-ориентированное программирование: использование объектно-ориентированного проектирования и программированием. Суть подхода состоит в том, что вводится понятие класса как развитие понятия модуля с определенными свойствами и поведением, характеризующими обязанностями класса. Каждый класс может порождать объекты – экземпляры данного класса. При этом работают основные принципы (парадигмы) ООП:

✔ Инкапсуляция – объединение в классе данных (свойств) и методов (процедур обработки).

✔ Наследование – возможность вывода нового класса из старого с частичным изменением свойств и методов

США тратит ежегодно более $200 млрд. на более чем 170 тыс. проектов разработки ПО в сфере IT;

31,1% из них закрываются, так и не завершившись; 52,7% проектов завершаются с превышением первоначальных оценок бюджета/сроков и ограниченной функциональностью; потери от недополученного эффекта внедрения ПО измеряются триллионами.

Причины:

Нереалистичные временные рамки

Недостаток количества исполнителей

Недостаток средств

Нехватка квалифицированных кадров

- **Инженерия программного обеспечения** - инженерная дисциплина, охватывающая все аспекты разработки программного обеспечения
- Программное обеспечение это набор компьютерных программ, процедур и связанной с ними документации и данных (ISO/IEC 12207).
- В зависимости от того, для кого разрабатываются программные продукты (конкретного заказчика или рынка, программные продукты бывают двух типов:
- коробочные продукты (generic products – общие продукты или shrink-wrapped software – упакованное ПО)
- заказные продукты (bespoke – сделанный на заказ или customized products – настроенный продукт). В первом случае ставит задачу (определяет, или специфицирует требования сами разработчики на основе анализа рынка (маркетинга). Во втором – заказчик.

- Основной принцип программной инженерии состоит в том, что программы создаются в результате выполнения нескольких взаимосвязанных этапов (анализ требований, проектирование, разработка, внедрение, сопровождение), составляющих жизненный цикл программного продукта.
- Программная инженерия занимается не только техническими вопросами производства ПО (специфицирование требований, проектирование, кодирование,…), но и управлением программными проектами, включая вопросы планирования, финансирования, управления коллективом и т.д. Кроме того, задачей программной инженерии является разработка средств, методов и теорий для поддержки процесса производства ПО.
- Информатика (computer science) занимается теорией и методами вычислительных и программных систем, в то время как программная инженерия занимается практическими проблемами создания ПО. Информатика составляет теоретические основы программной инженерии.

Основные проблемы, стоящие перед специалистами по программному обеспечению

- *Проблема наследования ранее созданного ПО.* Многие большие программные системы, эксплуатируемые в настоящее время, созданы много лет назад, но до сих пор выполняют свои функции надлежащим образом. Проблема наследования означает поддержку и модернизацию таких систем, причем при минимальных финансовых и временных затратах.
- *Проблема все возрастающей разнородности программных систем.* В настоящее время программное обеспечение должно быть способно работать в качестве систем, распределенных в компьютерных сетях, состоящих из компьютеров разных типов и использующих различные операционные системы. Проблема возрастающей разнородности программных систем состоит в том, что необходимо разрабатывать надежные программные системы, способные работать совместно с ПО разных типов.
- *Проблема, порожденная требованием уменьшения времени на создание ПО.* Многие традиционные технологии создания качественного программного обеспечения требуют больших временных затрат. Вместе с тем сегодня запросы рынка ПО и требования к программным системам меняются очень быстро. Поэтому и ПО должно меняться с соответствующей скоростью. Проблема, порожденная требованием уменьшения времени на создание ПО, заключается в том, чтобы сократить время на разработку больших и сложных программных систем без снижения их качества.

Процесс создания программного обеспечения

Создание ПО — это совокупность процессов, приводящих к созданию программного продукта. Существует четыре фундаментальных процесса которые присущи любому проекту создания ПО.

Разработка спецификации требований на программное обеспечение. Требования определяют функциональные характеристики системы и обязательны для выполнения.

Создание программного обеспечения. Разработка и создание ПО согласно спецификации на него.

Аттестация программного обеспечения. Созданное ПО должно пройти аттестацию для подтверждения соответствия требованиям заказчика.

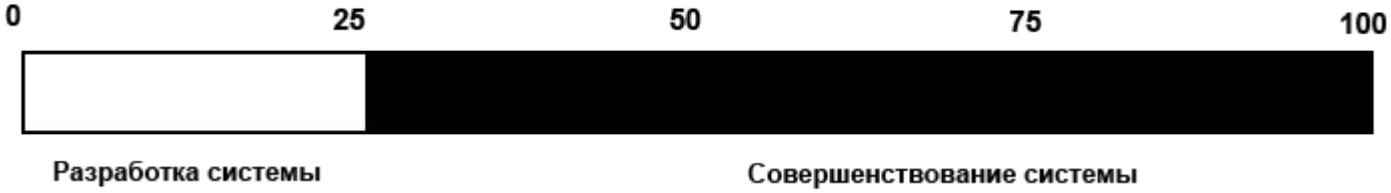
Совершенствование (модернизация) программного обеспечения. ПО должно быть таким, чтобы его можно было модернизировать согласно измененным требованиям потребителя.

При выполнении разнообразных программных проектов эти процессы могут быть организованы различными способами и описаны на разных уровнях детализации. Длительность реализации этих процессов также далеко не всегда одинакова. Если использовать неподходящий процесс, это может привести к снижению качества и функциональности разрабатываемого программного продукта.

Структура затрат на создание ПО

Структура затрат на создание программного обеспечения существенно зависит от процессов, используемых при разработке ПО, а также от типа разрабатываемого программного продукта. Если принять общую стоимость создания ПО за 100 единиц, то распределение стоимостей отдельных этапов производства может иметь такой вид

Стоимость создания ПО также могут включаться затраты на его модернизацию после начала эксплуатации программного продукта. Для многих программных систем затраты на совершенствование системы могут превышать стоимость разработки в 3 или 4 раза



- Стоимость программы – это стоимость ее разработки
  - Стоимость коробочных продуктов «размазывается» по копиям
  - Стоимость заказных продуктов (массово не копируемых) остается высокой
- Тестирование продукта – это единственный способ убедиться в его качестве. Именно поэтому стоимость тестирования составляет существенную стоимость ПО.

- Стоимость модернизации общих программных продуктов (т.е. тех, которые продаются на открытом рынке программ) с трудом поддается оценке. Во многих случаях осуществляется небольшая формальная модернизация. Обычно с началом реализации созданного программного продукта начинается работа с его следующей версией. Но исходя из требований маркетинга предпочтительнее представить новую версию как новый (но совместимый со старой версией) программный продукт, а не как модифицированную версию того продукта, которую пользователь уже купил. Поэтому стоимость модернизации ПО обычно не подсчитывается отдельно, как это делается при модернизации заказных программных продуктов, а просто входит в стоимость разработки следующей версии программной системы.
- Структура затрат на создание систем для электронной коммерции в Internet обычно отличается от того, что описано выше. В таких системах вместо создания программных модулей, управляющих информацией, обычно используют готовое программное обеспечение, а основные затраты приходятся на разработку пользовательских интерфейсов.

Основные показатели качественного программного обеспечения

- **Удобство сопровождения**

ПО должно быть таким, чтобы существовала возможность его усовершенствования в ответ на измененные требования заказчика или пользователя. Это определяющий показатель, поскольку любое ПО неминуемо подвергается модернизации вследствие изменений, происходящих в реальном мире

- **Надежность**

Определяется рядом характеристик, таких как безотказность, защищенность и безопасность. Надежность ПО означает, что возможные сбои в работе системы не приведут к физическому или экономическому ущербу

- **Эффективность**

Работа ПО не должна приводить к расточительному расходованию таких системных ресурсов, как память или время занятости процессора. Поэтому эффективность ПО описывается следующими характеристиками: скорость выполнения, используемое процессорное время, объем требуемой памяти и т.п.

- **Удобство в использовании**

ПО должно быть удобным в эксплуатации и не требовать чрезмерного напряжения усилий пользователя того уровня, на которого оно рассчитано. Это означает, что программная система должна обладать соответствующим пользовательским интерфейсом и необходимой документацией

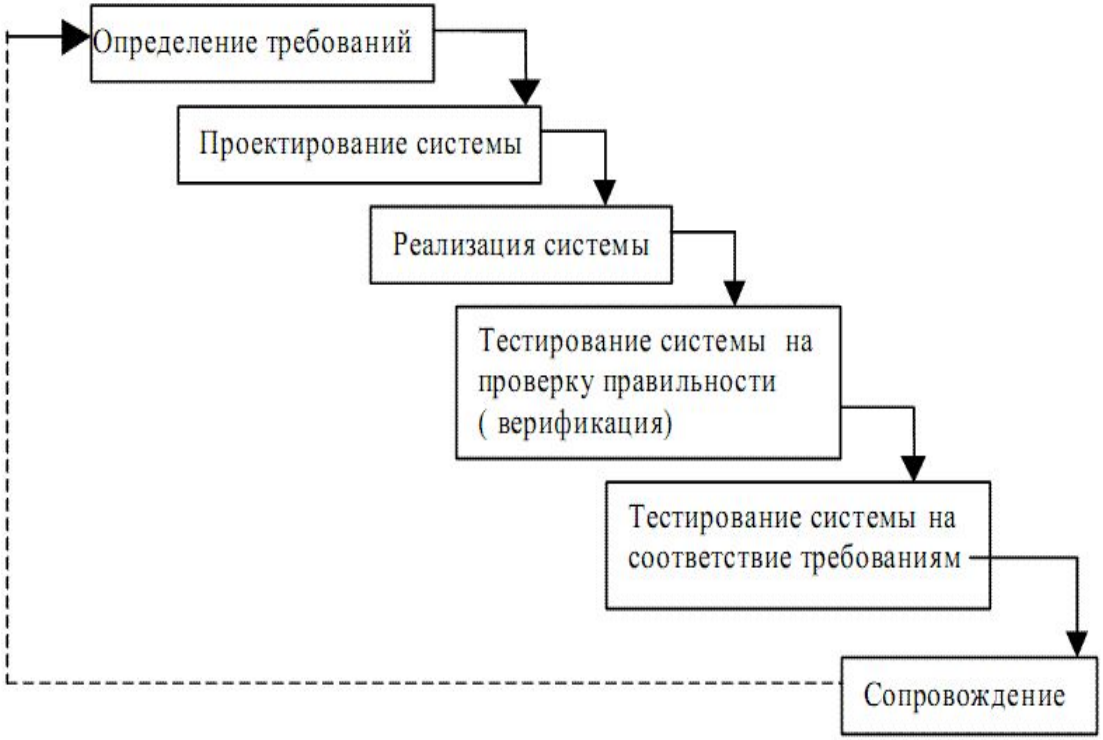
- Одним из основных понятий программной инженерии является понятие жизненного цикла программного продукта и программного процесса.
  Жизненный цикл – непрерывный процесс, начинающийся с момента принятия решения о создании ПО и заканчивающийся снятием его с эксплуатации. Жизненный цикл разбивается на отдельные процессы.
  Процесс – совокупность действий и задач, имеющих целью достижение значимого результата. Основными процессами (иногда называют этапами или фазами) жизненного цикла являются:
  - Разработка спецификации требований (результат – описания требований к программе, которые обязательны для выполнения – описание того, что программа должна делать)
  - Разработка проекта программы (результат – описание того, как программа будет работать)
  - Кодирование (результат – исходный код и файлы конфигурации)
  - Тестирование программы (результат - контроль соответствия программы требованиям)
  - Документирование (результат – документация к программе)

  Кроме основных, существует много дополнительных и вспомогательных процессов, связанных не созданием продукта, а с организацией работ (нефункциональные процессы): создание инфраструктуры, управление конфигурацией, управление качеством, обучение, разрешение противоречий, и т.п.

- Модель программного процесса — это упрощенное описание программного процесса, представленное с некоторой точки зрения. Модель задается в виде практических этапов, необходимых для создания ПО.
- Выбор модели процесса является первым шагом в создании ПО. Правильный выбор модели очень важен, т.к. во многом определяет успех проекта.
- Типы моделей процесса: модели процесса разработки (модели жизненного цикла) и модели организации работ по выполнению разработки

- К первым типам моделей (модели жизненного цикла) относятся модели, в которых описывается порядок выполнения действий по созданию продукта. К наиболее известным моделям относятся: Водопадная (каскадная) модель, Спиральная (циклическая) модель, Эволюционная модель..
- Различия между этими моделями существуют только в теории. Задача программного инженера – подобрать правильную их комбинацию, ориентируясь только на конечный результат.
- Ко второму типу моделей – моделей организации работ относятся:
- Модель потока работ (workflow model) — показывает последовательность действий, выполняемых людьми на различных этапах разработки ПО. Для каждого действия указываются входы, выходы (результаты) и связи по входам и выходам.
- Модель потоков данных (data flow model) — представляет процесс в виде последовательного преобразования данных. Каждое преобразование может выполняться одним или несколькими действиями.
- Ролевая модель — показывает роли людей, участвующих в программном процессе, а также действия и результаты, за которые они отвечают

Каскадная (водопадная) модель ЖЦ



Процесс разбивается на последовательное выполнение стадий; каждая стадия начинается после полного завершения предыдущей, продукт создается завершением последней стадии и должен полностью соответствовать изначально установленным требованиям. Однако вспомогательные и организационные процессы (контроль требований, управление качеством и др.) обычно выполняются параллельно с процессом разработки. В данной модели возвращение к начальному процессу предусматривается после сопровождения и исправления ошибок.

Основными принципами каскадной модели являются:

- Каждая последующая фаза начинается лишь тогда, когда полностью завершено выполнение предыдущей фазы
- Каждая фаза имеет определенные критерии входа и выхода: входные и выходные данные.
- Каждая фаза полностью документируется
- Переход от одной фазы к другой осуществляется посредством формального обзора с участием заказчика
- Основа модели – сформулированные требования (ТЗ), которые меняться не должны. Критерий качества результата – соответствие продукта установленным требованиям.

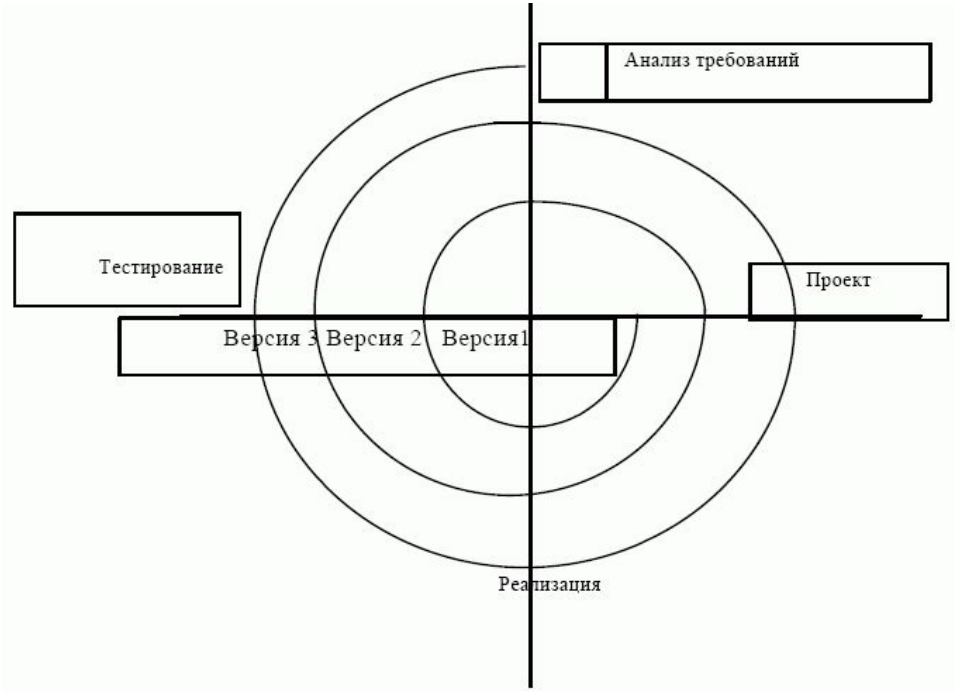
Каскадная модель имеет следующие преимущества: ·

- Проста и понятна заказчикам
- Проста и удобна в применении:
- способствует осуществлению строгого контроля менеджмента проекта;
- Каждую стадию могут выполнять независимые команды (все документировано). ·
- Позволяет достаточно точно планировать сроки и затраты.

Недостатки этой модели:

- процесс создания ПС не всегда укладывается в такую жесткую форму и последовательность действий;
- не учитываются изменившиеся потребности пользователей, изменения во внешней среде, которые вызовут изменения требований к системе в ходе ее разработки;
- большой разрыв между временем внесения ошибки (например, на этапе проектирования) и временем ее обнаружения (при сопровождении), что приводит к большой переделке ПС.

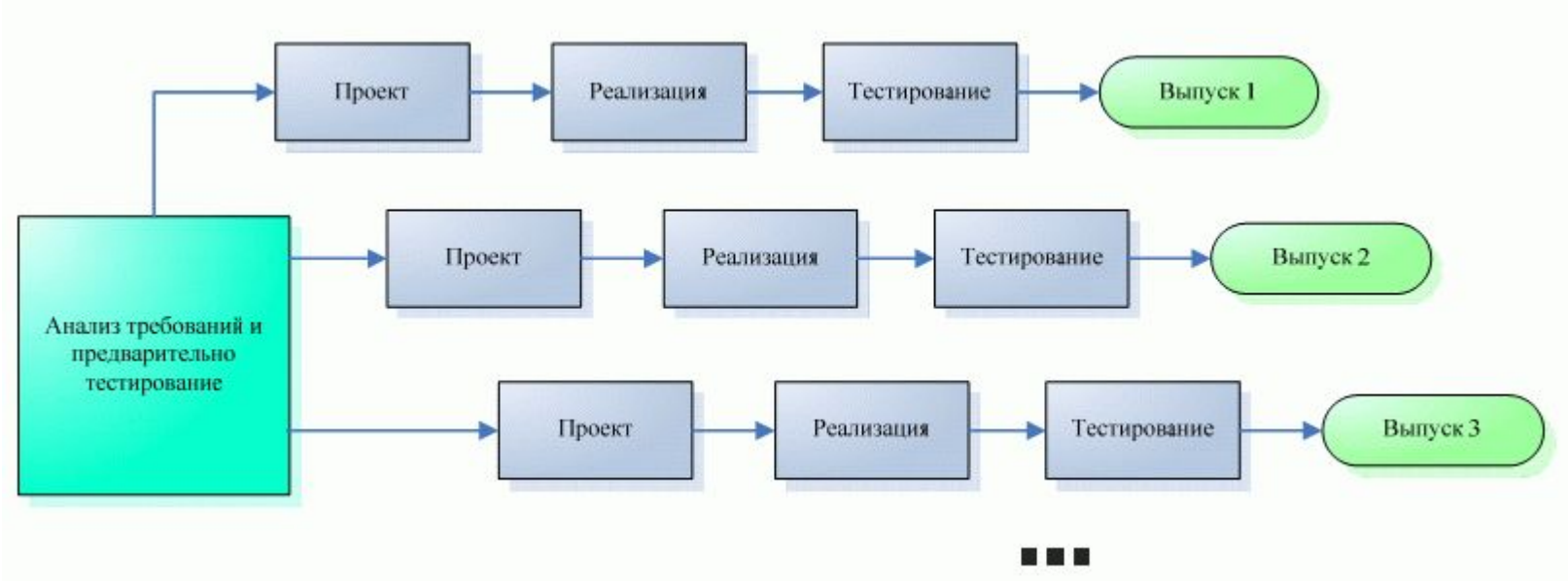
Спиральная (циклическая) модель



- спиральная модель ЖЦ делает упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

- Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.
- Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Инкрементная модель ЖЦ



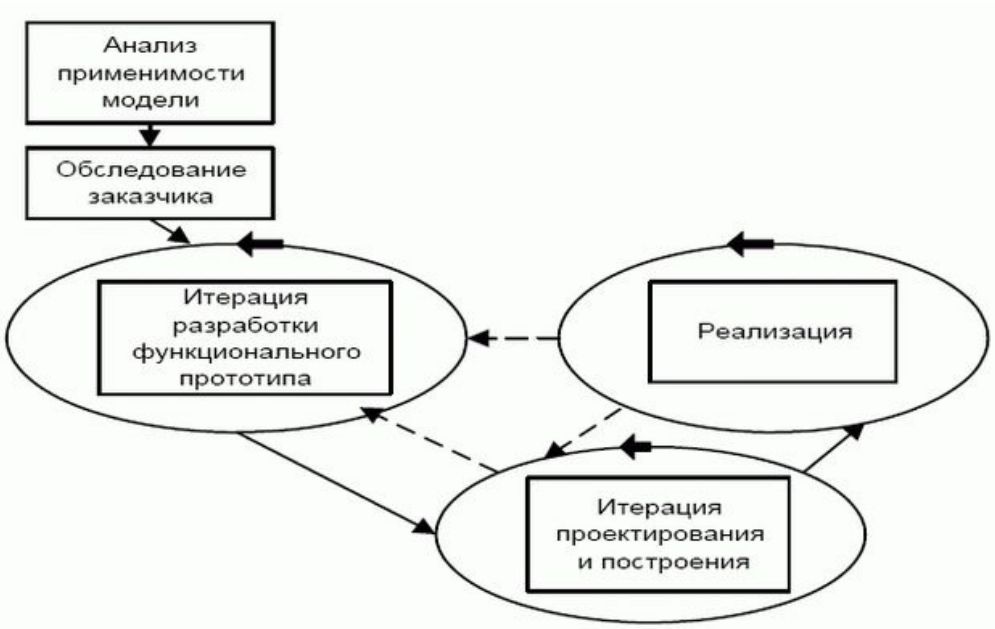
- Первая создаваемая промежуточная версия системы (выпуск 1) реализует часть требований, в последующую версию (выпуск 2) добавляют дополнительные требования и так до тех пор, пока не будут окончательно выполнены все требования и решены задачи разработки системы. Для каждой промежуточной версии на этапах ЖЦ выполняются необходимые процессы, работы и задачи, в том числе, анализ требований и создание новой архитектуры, которые могут быть выполнены одновременно.

- Инкрементная разработка представляет собой процесс частичной реализации всей системы и медленного наращивания функциональных возможностей. С помощью этой модели ускоряется процесс создания функционирующей системы. Этому способствует применяемый принцип компоновки из стандартных блоков, благодаря которому обеспечивается контроль над процессом разработки изменяющихся требований.
- В начале работы над проектом определяются все основные требования к системе, после чего выполняется ее разработка в виде последовательности версий. При этом каждая версия является законченным и работоспособным продуктом. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система.

- Данная модель жизненного цикла характерна при разработке сложных и комплексных систем, для которых имеется четкое видение (как со стороны заказчика, так и со стороны разработчика) того, что собой должен представлять конечный результат (информационная система). Разработка версиями ведется в силу разного рода причин:

- отсутствия у заказчика возможности сразу профинансировать весь дорогостоящий проект;
- отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;
- требований поэтапного внедрения и освоения продукта конечными пользователями. Внедрение всей системы сразу может вызвать у ее пользователей неприятие и только «затормозить» процесс перехода на новые технологии.

Эволюционная модель



Эта модель основана на следующей идее: разрабатывается первоначальная версия программного продукта, которая передается на испытание пользователям, затем она дорабатывается с учетом мнения пользователей, получается промежуточная версия продукта, которая также проходит "испытание пользователем", снова дорабатывается и так несколько раз, пока не будет получен необходимый программный продукт

- В случае эволюционной модели система разрабатывается в виде последовательности блоков структур (конструкций). В отличие от инкрементной модели ЖЦ подразумевается, что требования устанавливаются частично и уточняются в каждом последующем промежуточном блоке структуры системы.
- Использование эволюционной модели предполагает проведение исследования предметной области для изучения потребностей заказчика проекта и анализа возможности применения этой модели для реализации. Модель применяется для разработки несложных и не критических систем, для которых главным требованием является реализация функций системы. При этом требования не могут быть определены сразу и полностью. Тогда разработка системы проводится итерационно путем ее эволюционного развития с получением некоторого варианта системы - прототипа, на котором проверяется реализация требований.Такой процесс является итерационным, с повторяющимися этапами разработки, начиная от измененных требований и до получения готового продукта.

Быстрая разработка приложений (RAD)

Под RAD-разработкой обычно понимается процесс разработки, содержащий 3 элемента:

- небольшую команду программистов (до 10 человек);
- короткий, но тщательно проработанный производственный график (от 2 до 6 месяцев);
- повторяющийся цикл, при котором разработчики по мере того, как приложение начинает обретать форму, реализуют в продукте требования, полученные через взаимодействие с заказчиком.

методология RAD подразумевает использование на каждой итерации:

- Case-средства (набор инструментов автоматизации) для формирования и анализа требований, проектирования системы, автоматической генерации кода программ и структуры БД, а также автоматического тестирования программного обеспечения;
- инструментальных средств, обеспечивающих визуальную разработку (программирование) системы. Среда разработки приложений позволяет без написания кода программы создавать («рисовать») сложные графические интерфейсы пользователя, состав и структуру БД, запросы к БД, а также связывать данные с элементами интерфейса (переключателями, полями ввода, таблицами и т. д.);
- инструментальных средств, поддерживающих объектно-ориентированный подход. Эти средства позволяют создать описание предметной области в виде совокупности объектов – сущностей реального мира, характеризуемых свойствами (атрибутами) и поведением (методами);
- инструментальных средств, обеспечивающих событийное программирование. Каждый объект, входящий в состав приложения, может генерировать события и реагировать на события, генерируемые другими объектами;
- шаблонов и библиотек готовых решений как собственной разработки, так и сторонних производителей.

Условия применения методологии RAD:

- применима для относительно небольших проектов, разрабатываемых под конкретного заказчика;
- неприменима для построения сложных расчетных программ, операционных систем или программ управления космическими кораблями, т.е. программ, требующих написания большого объема (сотни тысяч строк) уникального кода;
- неприменима для разработки приложений, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени);
- неприменима для разработки приложений, от которых зависит безопасность людей (например, управление самолетом или атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий, скорее всего не будут полностью работоспособны, что в данном случае исключается.

Экстремальное программирование **XP-процесс**

Данный подход ориентирован на разработку информационных систем группами малого и среднего размера в условиях неопределенных или быстро изменяющихся требований.

Отличительными особенностями XP-разработки являются:

- частая смена версий и модификаций
- непрерывная связь с заказчиком – в группе все время находится квалифицированный представитель заказчика.
- простое проектирование – при разработке всегда выбирается наиболее простое решение.
- простой дизайн – система должна быть спроектирована настолько просто, насколько это возможно на каждый момент времени. Чем интерфейс проще, тем быстрее и качественнее идет освоение системы пользователями. Это не означает, что интерфейс «командной строки» самый предпочтительный. Как раз наоборот, «дружественный» и простой интерфейс должен быть интуитивно-понятным для пользователя; в нем должны отсутствовать бесполезные элементы, основная цель которых – произвести визуальное впечатление на пользователя;

- коллективное владение кодом – любой, кто видит возможность улучшить какую-то часть кода, может сделать это в любой момент времени. Это подразумевает применение одинаковых стандартов и правил оформления кода с исчерпывающими комментариями, а также и ведение общедоступной истории развития системы;
- программирование в парах – на пару программистов приходится один компьютер. Пока один из них непосредственно программирует, другой обдумывает вопросы реализации требований (функций, БД и т.п.). Смысл положения заключается не в экономии на материально-техническом обеспечении команды разработчиков, а в более разумном сочетании разных видов деятельности каждого из них. Как показывает опыт, при непосредственном программировании, исправлении ошибок и т.п. программист начинает думать односторонне и все более узкими категориями. Именно поэтому во время «отдыха от компьютера» приходят наиболее удачные решения;
- непрерывное и пересекающееся проектирование, разработка, интеграция и тестирование системы.

Анализ предметной области и требования к ПО

- 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.
- Описание функциональных возможностей и ограничений, накладываемых на программную систему, называется *требованиями* к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – *разработкой требований* (requirements engineering).
- Требования к ПО определяют, какие свойства и характеристики оно должно иметь для удовлетворения потребностей пользователей и других заинтересованных лиц

Цикл работы с требованиями

- Выделение требований (requirements elicitation), нацеленное на выявление всех возможных источников требований и ограничений на работу системы и извлечение требований из этих источников.
- Анализ требований (requirements analysis), целью которого является обнаружение и устранение противоречий и неоднозначностей в требованиях, их уточнение и систематизация.
- Описание требований (requirements specification). В результате этой деятельности требования должны быть оформлены в виде структурированного набора документов и моделей.
- Валидация требований (requirements validation), которая решает задачу оценки понятности сформулированных требований и их характеристик

Виды требований

1. *Пользовательские требования* – описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на нее.
2. *Системные требования* – детализированное описание системных функций и ограничений, которое иногда называют функциональной спецификацией. Она служит основой для заключения контракта между покупателем системы и разработчиками ПО.
3. *Проектная системная спецификация* – обобщенное описание структуры программной системы, которое будет основой для более детализированного проектирования системы и ее последующей реализации. Эта спецификация дополняет и детализирует спецификацию системных требований.

Пример

Пользовательские требования

1. ПО должно предоставить средство доступа к внешним файлам, созданным в других программах.

Спецификация системных требований

1.1. Пользователь должен иметь возможность определять тип внешних файлов.

1.2. Для каждого типа внешнего файла должно иметься соответствующее средство, применимое к этому типу файлов.

1.3. Внешний файл каждого типа должен быть представлен соответствующей пиктограммой на дисплее пользователя.

Пользовательские требования пишутся для заказчика ПО и для лица, заключающего контракт на разработку программной системы, причем они могут не иметь детальных технических знаний по разрабатываемой системе. Спецификация системных требований предназначена для руководящего технического состава компании-разработчика и для менеджеров проекта. Она также необходима заказчику ПО и субподрядчикам по разработке. Эти оба документа также предназначены для конечных пользователей программной системы. Наконец, проектная системная спецификация является документом, который ориентирован на разработчиков ПО.

Спецификация требований к ПО является основным документом, определяющим план разработки ПО. Все требования, указанные в спецификации, делятся на функциональные и нефункциональные.

- **Функциональные** требования определяют действия, которые должна выполнять система, без учета ограничений, связанных с ее реализацией. Тем самым функциональные требования определяют поведение системы в процессе обработки информации.
- **Нефункциональные требования** не определяют поведение системы, но описывают атрибуты системы или атрибуты системного окружения. Можно выделить следующие типы нефункциональных требований:

✔ требования к применению, которые определяют качество пользовательского интерфейса, документации;

✔ требования к производительности, которые накладывают ограничения на функциональные требования, задавая необходимую эффективность использования ресурсов, пропускную способность и время реакции;

✔ требования к реализации, которые предписывают использовать определенные стандарты, языки программирования, операционную среду и ;

✔ требования к надежности, которые определяют допустимую частоту и воздействие сбоев, а также возможности восстановления;

✔ требования к интерфейсу, которые определяют внешние сущности, с которыми может взаимодействовать система, и регламент этого взаимодействия.

Описание требований

ПРИМЕРЫ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

1. Система ATM должна проверять действительность вставленной в банкомат карточки.
2. Система ATM должна проверять достоверность PIN-кода, введенного пользователем.
3. Система ATM должна выдавать по одной ATM-

ПРИМЕРЫ НЕФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

1. Система ATM должна быть написана на C++.
2. Система ATM должна обмениваться информацией с банком, используя 256-разрядную кодировку.
3. Система ATM должна проверять действительность карточки ATM в течение не более трех секунд.
4. Система ATM должна проверять достоверность PIN-кода в течение не

- Разработка требований - это процесс, включающий мероприятия, необходимые *для* создания и утверждения документа, содержащего спецификацию системных требований.
- Различают четыре основных этапа процесса разработки требований:

✔ анализ технической осуществимости создания системы,

✔ формирование и анализ требований,

✔ специфицирование требований и создание соответствующей документации,

✔ аттестация этих требований.

Анализ осуществимости

- Для новых программных систем процесс разработки требований должен начинаться с анализа осуществимости. Началом такого анализа является общее описание системы и ее назначения, а результатом анализа - отчет, в котором должна быть четкая рекомендация, продолжать или нет процесс разработки требований проектируемой системы. Анализ осуществимости должен осветить следующие вопросы.
  1. Отвечает ли система общим и бизнес-целям организации-заказчика и организации-разработчика?
  2. Можно ли реализовать систему, используя существующие на данный момент технологии и не выходя за пределы заданной стоимости?
  3. Можно ли объединить систему с другими системами, которые уже эксплуатируются?
  Критическим является вопрос, будет ли система соответствовать целям бизнеса. Если система не соответствует этим целям, она не представляет никакой ценности для бизнеса. В то же время многие организации разрабатывают системы, не соответствующие их целям, либо не совсем ясно понимая эти цели, либо под влиянием политических или общественных факторов.

- Выполнение анализа осуществимости включает сбор и анализ информации о будущей системе и написание соответствующего отчета. Сначала следует определить, какая именно информация необходима, чтобы ответить на поставленные выше вопросы

Формирование и анализ требований

- После выполнения анализа осуществимости следующим этапом процесса разработки требований является формирование (определение) и анализ требований. На этом этапе команда разработчиков ПО работает с заказчиком и конечными пользователями системы для выяснения области применения, описания системных сервисов, определения режимов работы системы и ее характеристик выполнения, аппаратных ограничений и т.д
- Процесс формирования и анализа требований проходит через ряд этапов.
  1. *Анализ предметной области.* Аналитики должны изучить предметную область, где будет эксплуатироваться система.
  2. *Сбор требований.* Это процесс взаимодействия с лицами, формирующими требования. Во время этого процесса продолжается анализ предметной области.
  3. *Классификация требований.* На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.
  4. *Разрешение противоречий.* требования многочисленных лиц, занятых в процессе формирования требований, будут противоречивыми.
  5. *Назначение приоритетов.* В любом наборе требований одни из них будут более важны, чем другие. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.
  6. *Проверка требований.* На этом этапе определяется их полнота, последовательность и непротиворечивость.

Опорные точки зрения

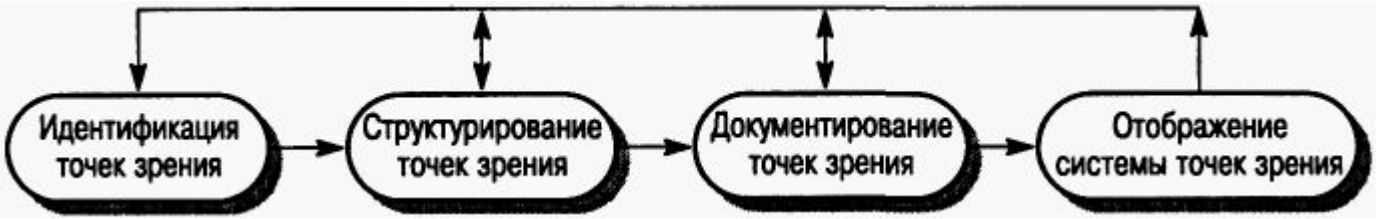
Любая средняя или большая система ПО обычно имеет различные типы конечных пользователей. Многие лица, участвующие в формировании требований, в своих требованиях к системе выражают собственные интересы.

Например, в процессе формировании требований для системы банкоматов участвуют следующие лица.

1. *Обычные клиенты банка,* пользующихся услугами банкоматов.
2. *Представители других банков,* имеющих взаимные соглашения с данным банком о совместном использовании банкоматов.
3. *Менеджеры филиалов банка,* получающих информацию из системы управления банкоматами.
4. *Сотрудники филиалов банка,* вовлеченные в повседневную работу системы банкоматов, обрабатывающие рекламации клиентов и т.д.
5. *Администраторы баз данных,* ответственные за связь банкоматов с базой данных клиентов.
6. *Руководители службы безопасности банка,* обеспечивающей защиту системы банкоматов.
7. *Отдел маркетинга банка,* использующий систему банкоматов как средство маркетинга.
8. *Разработчики аппаратных и программных средств,* ответственные за сопровождение и модернизацию аппаратных и программных средств.

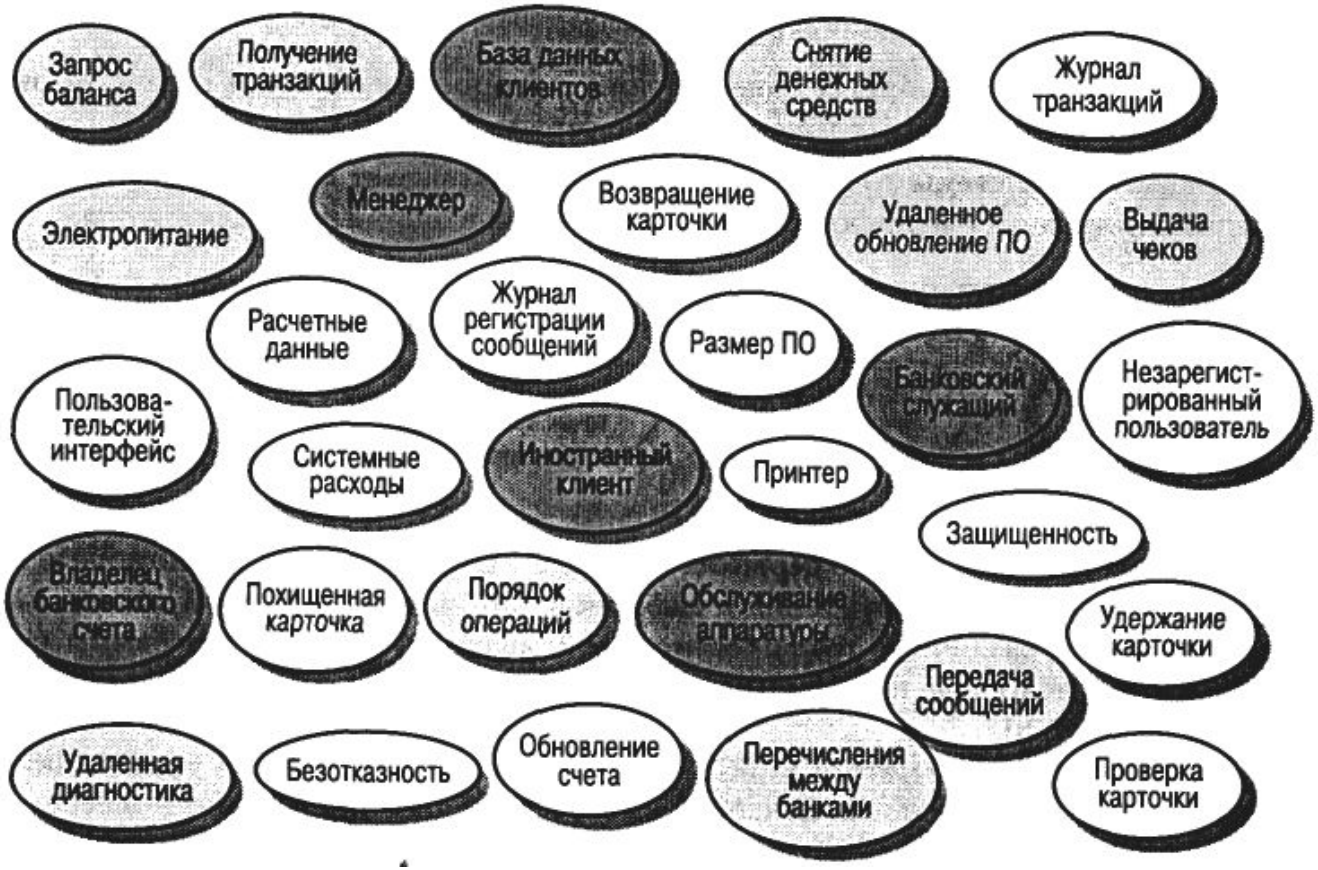
Этот список показывает, что даже для относительно простой системы существует много различных точек зрения, которые должны быть рассмотрены. Различные точки зрения на проблему позволяют увидеть ее с разных сторон. Однако эти взгляды не являются полностью независимыми и обычно перекрывают друг друга, а потому могут служить основой общих требований.

- Подход с использованием различных *опорных* точек зрения к разработке требований признает эти различные (опорные) точки зрения и использует их в качестве основы построения и организации как процесса формирования требований, так и непосредственно самих требований. Сильная сторона анализа, ориентированного на различные опорные точки зрения, в том, что он признает множество взглядов и обеспечивает основу для обнаружения противоречий в требованиях, предложенных различными лицами
- На основе этого подхода разработан метод VORD (Viewpoint-Oriented Requirements Definition – определение требований на основе точек зрения) для формирования и анализа требований. Основные этапы метода VORD
  1. Идентификация точек зрения, получающих системные сервисы, и идентификация сервисов, соответствующих каждой точке зрения.
  2. Структурирование точек зрения – создание иерархии сгруппированных точек зрения. Общесистемные сервисы предоставляются более высоким уровням иерархии и наследуются точками зрения низшего уровня.
  3. Документирование опорных точек зрения, которое заключается в точном описании идентифицированных точек зрения и сервисов.
  4. Отображение системы точек зрения, которая показывает системные объекты, определенные на основе информации, заключенной в опорных точках зрения.



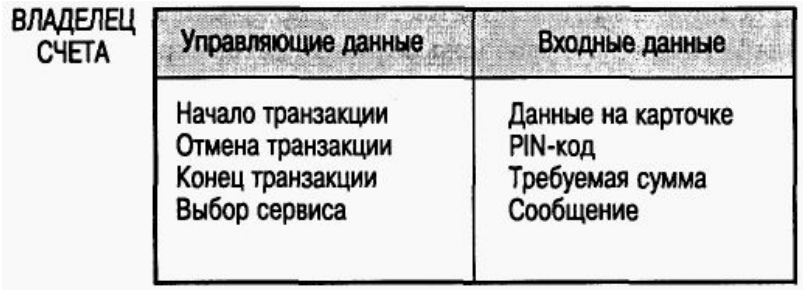
- использование метода VORD на первых трех шагах анализа требований для системы управления банкоматами. Банкомат имеет встроенное программное обеспечение для управления аппаратными средствами и для связи с центральной базой данных банковских счетов.
  Банкомат принимает запросы клиента, выдает наличные деньги, информацию о счете, изменяет данные в базе данных банка и т.д. Банкоматы одного банка могут позволить клиентам других банков использовать какую-то часть своих сервисов (обычно это снятие со счета наличных денег и запрос о текущем балансе счета).
- Первым шагом в формировании требований является идентификация опорных точек зрения. Организуется встреча лиц, участвующих в формировании требований, которые предлагают свои точки зрения. Эти точки зрения представляются в виде диаграммы, состоящей из ряда круговых областей, отображающих возможные точки зрения необходимо идентифицировать потенциальные опорные точки зрения, системные сервисы, входные данные, нефункциональные требования, управляющие события и исключительные ситуации. Источниками информации, которые можно использовать в создании этого начального образа системы, могут служить документы, описывающие назначение системы, знания инженеров-программистов из предыдущих проектов или опыт клиентов банка. Может быть проведен опрос менеджеров банка, обслуживающего персонала, консультантов, инженеров и клиентов.
  Следующей стадией процесса формирования требований будет идентификация опорных точек зрения и сервисов . Сервисы должны соответствовать опорным точкам зрения. Но могут быть сервисы, которые не поставлены им в соответствие. Это означает, что на начальном этапе "мозговой атаки" некоторые опорные точки зрения не были идентифицированы. Например, для сервисов "Удаленное обновление ПО" и "Удаленная диагностика» необходимо иметь точку зрения об обслуживании программного обеспечения
-

.

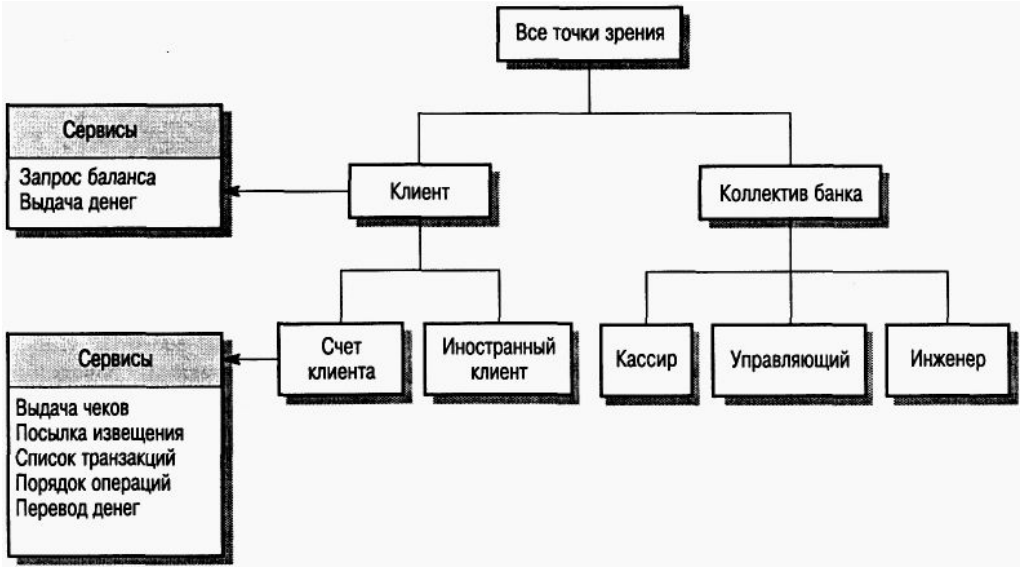


распределение сервисов для некоторых идентифицированных точек зрения

- Точки зрения также определяют входные данные и управляющую информацию для сервисов. Например, банкомат должен определять остаток денег после выдачи наличных. На ранних этапах процесса формирования требований эти данные и управляющая информация идентифицируются просто по имени.



информация, извлеченная из точек зрения, используется для заполнения форм шаблонов точек зрения и организации точек зрения в иерархию наследования. Это позволяет увидеть общие точки зрения и повторно использовать информацию в иерархии наследования. Сервисы, данные и управляющая информация наследуются подмножеством точек зрения.

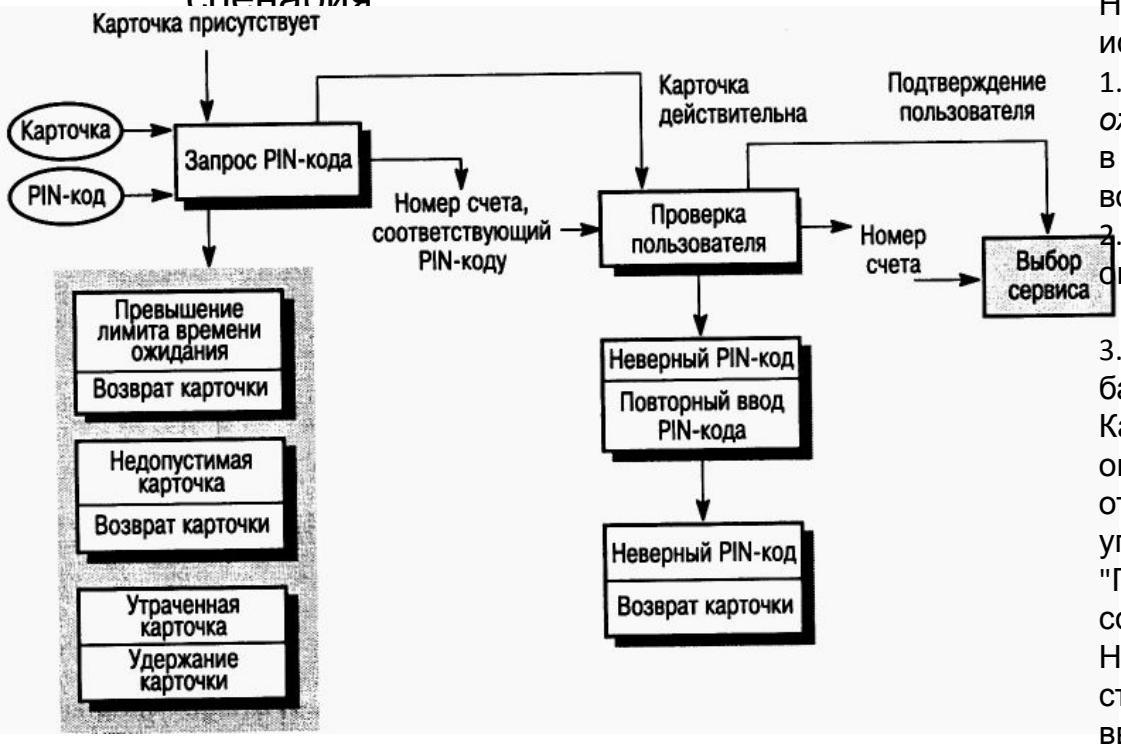


Следующая стадия процесса формирования требований – получение более детальной информации относительно сервисов, используемых сервисами данных, и управляющих данных. Эта информация извлекается из мнений лиц, формирующих требования, связанные с каждой опорной точкой зрения. Для этого используются шаблоны точек зрения и описания сервисов в виде сценариев событий.

Сценарии

- Люди обычно легче воспринимают примеры из реальной жизни, чем абстрактные описания. Специалисты могут использовать информацию, полученную из обсуждения сценариев взаимодействия с системой, для формулирования требований.
  Сценарии особенно полезны для детализации уже сформулированных требований, поскольку описывают последовательность интерактивной работы пользователя с системой. Каждый сценарий описывает одно или несколько возможных взаимодействий. В настоящее время разработаны многочисленные формы сценариев, которые предоставляют различную информацию на разных уровнях детализации системы.
  Сценарий начинается с общего описания, затем постепенно детализируется для создания полного описания взаимодействия пользователя с системой. В большинстве случаев сценарий включает следующее.
  1. Описание состояния системы в начале сценария.
  2. Описание нормального протекания событий.
  3. Описание исключительных ситуаций и способов их обработки.
  4. Информацию относительно других действий, которые можно осуществлять во время выполнения сценария.
  5. Описание состояния системы после завершения сценария.
  Первоначальное описание сценария может быть выполнено неформально в процессе опроса лиц, формирующих требования. Альтернативой может служить структурный подход – разработка сценариев событий или вариантов использования.

- когда карточка вставлена, запрашивается персональный идентификационный номер клиента (PIN-код). Если карточка действительна, она может обрабатываться банкоматом, тогда управление переходит к следующей стадии сценария



На первой стадии сценария возможны три исключительные ситуации.

1. *Превышение лимита времени ожидания.* Клиент может не успеть ввести PIN-код в отведенное для ввода время. Карточка возвращается.

2. *Недопустимая карточка.* Карточка не опознается и возвращается.

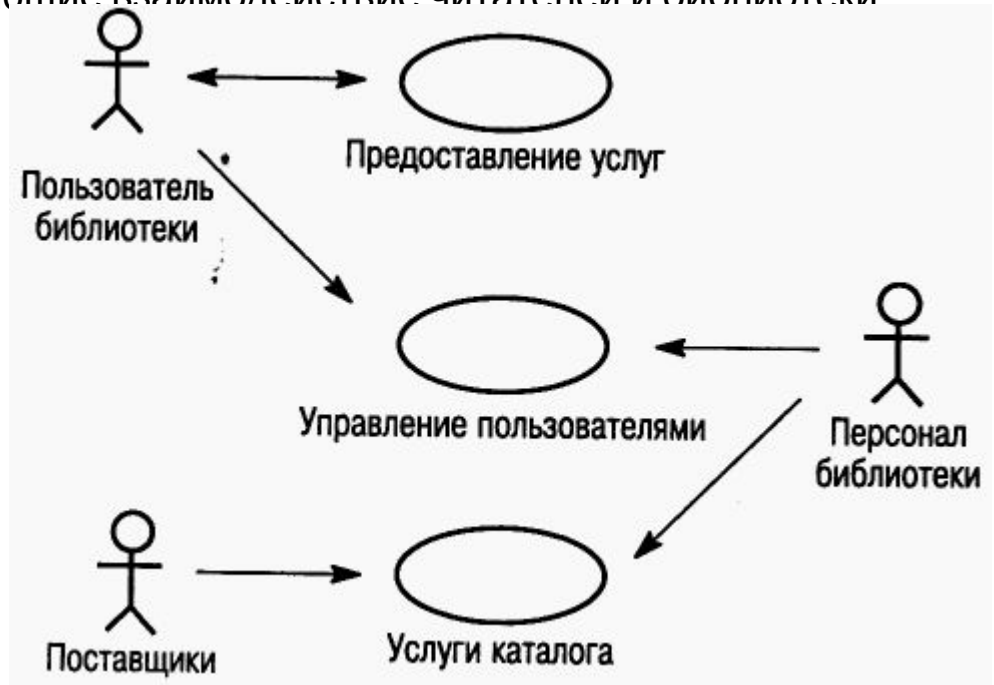
3. *Удержание карточки.* Карточка удерживается банкоматом.

Каждую исключительную ситуацию можно определить более подробно, построив отдельные диаграммы потоков данных и управления

"Проверка пользователя" – стадия проверки соответствия PIN-кода номеру счета клиента. Номер счета является выходными данными этой стадии. Возможная исключительная ситуация – ввод неверного PIN-кода, в этом случае PIN-код запрашивается снова. На диаграмме показано, что повторный запрос может опять привести к исключительной ситуации. Если повторно введенный PIN-код снова неверный, карточка возвращается. После события "Подтверждение пользователя" можно переходить к следующей стадии сценария "Выбор сервиса".

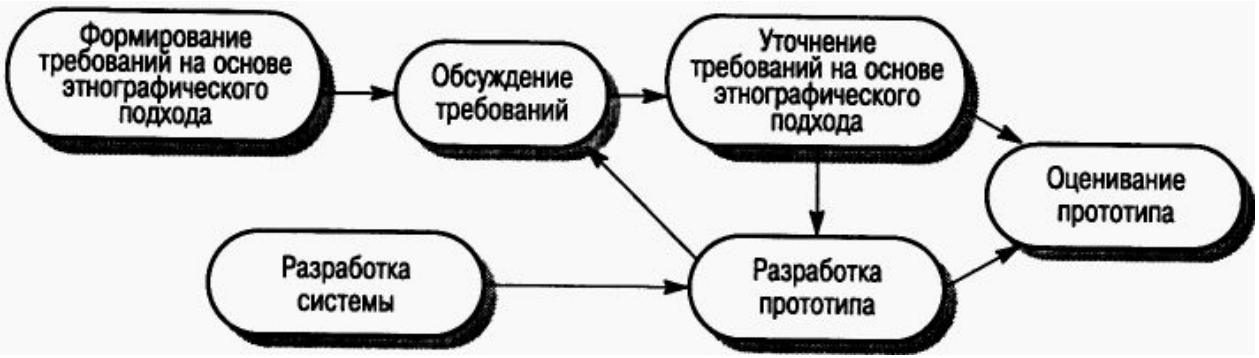
Варианты использования

- Варианты использования (use-case)– это методика формирования требований, основанная на сценариях. Они стали основой нотаций в языке моделирования UML при описании объектных моделей систем. В самой простой форме в варианте использования определены действующие лица (в UML они называются *актерами),* т. е. пользователи, вовлеченные во взаимодействие, и имена типов взаимодействия. Представлен простейший вариант использования, где показаны основные элементы обозначений. Действующие лица (актеры) изображены в виде стилизованных фигурок людей, а каждый класс взаимодействия представлен поименованным эллипсом. Множество вариантов использования охватывает все возможные взаимодействия, которые будут отражены в системных требованиях. показаны варианты использования, описывающие взаимодействие читателей и библиотеки



Этнографический подход

- Системы программного обеспечения не существуют изолированно от окружающего мира. Они эксплуатируются в определенной социальной и организационной среде, поэтому системные требования должны разрабатываться с учетом этой среды. Учет социальных и организационных требований часто имеет большое значение для успеха системы на рынке программных продуктов. Многие представленные на рынке программные системы практически не имеют спроса именно потому, что не учитывают важности социальных и организационных требований.
  Этнографический подход к формированию системных требований используется для понимания и формирования социальных и организационных аспектов эксплуатации системы. Разработчик требований погружается в рабочую среду, где будет использоваться система. Его ежедневная работа связана с наблюдением и протоколированием реальных действий, выполняемых пользователями системы. Значение этнографического подхода заключается в том, что он помогает обнаружить неявные требования к системе, которые отражают реальные аспекты ее эксплуатации, а не формальные умозрительные процессы.
  Обычно людям трудно четко описать все аспекты выполняемой работы, поскольку способ выполнения часто определяется их характером и практическим опытом. Они понимают свою работу, но не могут объяснить ее взаимосвязь с другими видами работ, выполняемых в организации. Социальные и организационные факторы, которые оказывают влияние на работу, но не являются очевидными, могут стать явными, если описаны беспристрастными наблюдателями.
- этнографический подход использован для изучения работы офиса. Показано, что фактическая работа, выполняемая в офисе, более сложна и динамична, чем предусматривает простая модель, принятая для системы автоматизации делопроизводства. Это расхождение между реальной работой и моделью послужило причиной того, что офисные системы автоматизации не показывали той эффективности, на которую были рассчитаны.
  Этнографический подход к формированию требований можно объединить с прототипированием .



- Этнографический подход позволяет получить требования, которые учитываются в разрабатываемом прототипе. Кроме того, этнографический подход используется при решении конкретных проблем прототипирования и при оценивании созданного прототипа
- Этнографический подход позволяет детализировать требования для критических систем, чего не всегда можно добиться другими методами разработки требований. Однако, поскольку этот метод ориентирован на конечного пользователя, он не может охватить все требования предметной области и требования организационного характера. Поэтому он не является всеохватывающим подходом в формировании требований и должен использоваться совместно с такими подходами, как анализ вариантов использования.

Аттестация требований

- Аттестация должна продемонстрировать, что требования действительно определяют ту систему, которую хочет иметь заказчик. Проверка требований важна, так как ошибки в спецификации требований могут привести к переделке системы и большим затратам, если будут обнаружены во время процесса разработки системы или после введения ее в эксплуатацию. Стоимость внесения в систему изменений, необходимых для устранения ошибок в требованиях, намного выше, чем исправление ошибок проектирования или кодирования. Причина в том, что изменение требований обычно влечет за собой значительные изменения в системе, после внесения которых она должна пройти повторное тестирование.

  Во время процесса аттестации должны быть выполнены различные типы проверок документации требований.
  1. *Проверка правильности требований.* Пользователь может считать, что система необходима для выполнения некоторых определенных функций. Однако дальнейшие размышления и анализ могут привести к необходимости введения дополнительных или новых функций. Системы предназначены для разных пользователей с различными потребностями, и поэтому набор требований будет представлять собой некоторый компромисс между требованиями пользователей системы.
  2. *Проверка на непротиворечивость.* Спецификация требований не должна содержать противоречий. Это означает, что в требованиях не должно быть противоречащих друг другу ограничений или различных описаний одной и той же системной функции.
  3. *Проверка на полноту.* Спецификация требований должна содержать требования, которые определяют все системные функции и ограничения, налагаемые на систему.
  4. *Проверка на выполнимость.* На основе знания существующих технологий требования должны быть проверены на возможность их реального выполнения. Здесь также проверяются возможности финансирования и график разработки системы.

Программные документы

Вид программного документа	Содержание программного документа
Спецификация	Состав программы и документации на нее
Текст программы	Запись программы с необходимыми комментариями
Программа и методика испытаний	Требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля
Техническое задание	Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	Схема алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений
Эксплуатационные документы	Сведения для обеспечения функционирования и эксплуатации программы

Виды эксплуатационных документов и их содержание приведены в таблице2

Эксплуатационные документы

Вид эксплуатационного документа	Содержание эксплуатационного документа
Описание применения	Сведения о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств
Руководство системного программиста	Сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения
Руководство программиста	Сведения для эксплуатации программы
Руководство оператора	Сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программы