

Технологии программирования

2011

Лекции по технологиям программирования запечатаны в 2011 году.

Романова Татьяна Николаевна.
Вишневская Татьяна Ивановна.

11 мая 2011
Михаил Никифоров
melvin-studios@mail.ru

Оглавление

| | |
|---|----|
| 08.02.2011..... | 3 |
| Основные требования к методологии проектирования программного обеспечения | 3 |
| Основные требования к методикам и методам проектирования ПО | 3 |
| 15.02.2011..... | 5 |
| Сложная система. | 5 |
| Разработка технического задания. | 5 |
| Диаграмма Use-Case (диаграмма прецедентов) | 9 |
| Методология быстрой разработки приложений – RAD. | 10 |
| 22.02.2011..... | 11 |
| Структурный подход в проектировании..... | 11 |
| Методология Гейна/Сарсона..... | 11 |
| Нотация Йордена/Де Марка | 12 |
| Синтаксис графического языка IDEF0..... | 13 |
| Модель IDF3 | 15 |
| 01.03.2011..... | 16 |
| Типы отношений..... | 16 |
| Диаграмма атрибутов | 17 |
| Диаграмма потоков данных (ДПД, DFD)..... | 18 |
| 15.03.2011..... | 19 |
| Методология RUP и пример ее использования на простом примере о торговой фирме..... | 19 |
| Спецификации прецедента «обновить данные из внешней системы». | 22 |
| Возможные отношения между сценариями..... | 23 |
| UML, разновидности предметов, существующих в UML. | 24 |
| 22.03.2011..... | 26 |
| Операции..... | 26 |
| Группировка | 26 |
| Множественность. Как ее обозначить? | 27 |
| Реализация..... | 28 |
| 29.03.2011..... | 29 |
| Деревья наследования | 29 |
| Автомат..... | 30 |
| Диаграмма деятельности. | 32 |
| Защелкивание, выбор вариантов и циклы. | 35 |
| 05.04.2011..... | 37 |
| Оценка производительности распределенных информационных систем на этапе проектирования..... | 37 |
| Модели реализации..... | 37 |
| Компонентная диаграмма | 38 |
| 12.04.2011..... | 39 |
| Computer-Aided Software/System Engineering (CASE) | 39 |
| Критерии классификации CASE-систем. | 40 |
| Тестирование ПО | 40 |
| Этапы тестирования. | 41 |
| Нагрузочное и предельное тестирование..... | 42 |
| Интеграционное тестирование при структурном подходе к программированию. | 43 |
| Тестирование при объектно-ориентированном подходе. | 43 |
| Сложность тестирования интеграционного класса. | 44 |
| 19.04.2011..... | 46 |
| Структурное тестирование программного обеспечения | 46 |
| Способ тестирования базового пути..... | 46 |
| Потоковый граф | 47 |
| Графы и отношения | 48 |

| | |
|--|----|
| Отношения (симметричность, транзитивность) | 49 |
| Тестирование циклов | 49 |
| 26.04.2011..... | 50 |
| Тестирование очередей и потоков данных. | 50 |
| Как тестировать очереди. | 50 |
| Тестирование потока транзакций. | 52 |
| Тестируем декларацию..... | 53 |
| Лекция 03.05.2011 | 54 |
| Международные стандарты на разработку ПО..... | 54 |
| Стандарты, регламентирующие документирование программных средств и баз данных. | 54 |
| Профиль стандартов документирования объектов | 54 |
| Эксплуатационная документация..... | 55 |
| Исследовательская документация | 55 |
| Пользовательская документация | 56 |
| Лекция 10.05.2011 | 58 |
| Основные качественные характеристики программных средств и их атрибутов. | 58 |
| Пример требований к количественным характеристикам качества программного средства..... | 59 |
| Характеристики качества баз данных..... | 60 |
| Жизненный цикл профилей стандартов | 60 |

08.02.2011

Основные требования к методологии проектирования программного обеспечения

Технология – сложный комплекс, в основе которого лежит применение различных орудий, инструментов и аппаратов, использующий наработанные человечеством знания и умения.

Первая информационная революция – письменность.

Вторая – создание печатного станка.

Третья – разработка ЭВМ, компьютеров.

(Новая) информационная технология – система методов и способов сбора, получения, накопления, хранения, обработки, анализа и передачи информации с использованием средств вычислительной техники.

Мы будем работать именно с информационной технологией.

Методология – совокупность механизмов, применяемых при разработке программных систем и объединённых единым философским подходом.

Что значит философский подход. Мы иногда «подход» называем парадигмой, т.е. это некое направление, в котором мы будем развивать наши идеи. Какие парадигмы мы знаем? ООП, Структурное, субъектно-ориентированное, функциональное, процессорное и т.д. Подход выбирается от класса задач, которые требуется реализовать.

Мы будем говорить о структурном и ООП.

Метод – концептуальное описание правил построения моделей системы, представляющих разные взгляды на проект с использованием специальных графических нотаций, которая определяет изобразительные средства и состав документации по проекту.

Основные требования к методикам и методам проектирования ПО

1. Метод должен отражать специфику подхода (парадигму программирования) – структурный, ОО, ориентированный на процессе, субъектно-ориентированный.
2. Метод должен быть освоен всеми участниками проекта и должен быть наглядным с точки зрения полученных результатов.
3. Должны существовать формальные переходы от этапов анализа к этапу проектирования и обратно.
4. Должны существовать инструментальные средства, поддерживающие все эти методы.

ISO/IEE 12207:1995 Information Technology – software life cycle processes (IT – процессы жизненного цикла).

Этот стандарт описывает структуру жизненного цикла программного обеспечения и его процесс.

Процесс жизненного цикла – совокупность взаимосвязанных действий, преобразующих некоторое входное в выходные.

Каждый процесс характеризуется некоторыми задачами и методами их выполнения.

Модели жизненного цикла

1. Каскадная (есть основные процессы).

Состоит из последовательных шагов:

- Анализ предметной области →
- Проектирование →
- Кодирование →
- Тестирование →
- Внедрение →
- И сопровождение.

В ТЗ профиль стандартов должен быть прописан.

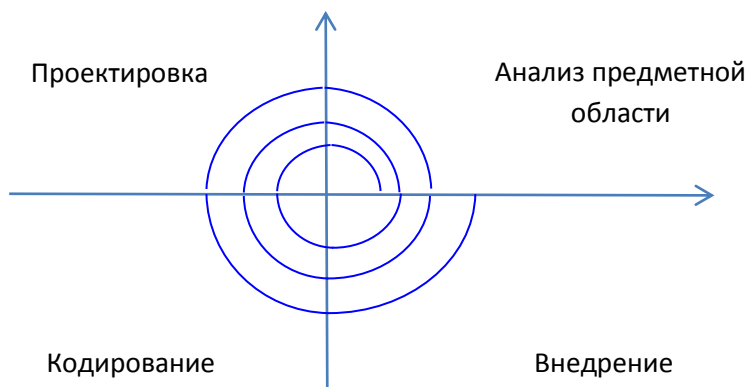
15.02.2011

Сложная система.

1. Такая система, которая разрабатывается не одним человеком, а группой разработчиков (более 5 человек).
2. Если количество строк исходного кода исчисляется сотнями тысяч или даже миллионами, то это тоже показатель сложной системы.
3. Если сложную задачу можно декомпозировать на более простые задачи (которые будут реализованы более мелкими подсистемами).
4. Если в требованиях встречаются взаимнопротиворечивые требования, то такая система будет однозначно сложной. Например, нужно обработать огромные информационные потоки, и время отклика должно быть кратчайшим. Следует найти компромисс.

Это самые важные и самые общие требования, а остальные из них вытекают.

Несколько слов об этапах жизненного цикла.



На каждом новом витке этой спирали могут добавляться новые наработки. Недостаток в том, что граница (напр., между Анализ и проектировка) очень расплывчата. В каскадной модели такого нет.

На каждом этапе я должен выдать документацию.

Каскадная модель работает при структурном подходе в программировании. Напр., при разработке крупных военных систем, где каждый этап регламентирован.

Но в бизнес-системах применяется спиралевидная. Её применяют для систем не критичных к ошибкам. В ООП.

Разработка технического задания.

Основные эксплуатационные требования к программным продуктам, а также выбор архитектуры.

Техническое задание и ГОСТы, регламентирующие этапы разработки.

ГОСТ 19.201-78: «Единая система программной документации», подраздел «техническое задание». ЕСПД.ТЗ: «Требования к содержанию и оформление».

В соответствии с этим стандартом ТЗ содержит:

1. Введение.

Здесь следует писать краткую характеристику предметной области (медицина, торговля, промышленность). Можно даже нарисовать в виде картинки. Есть торговое предприятие. Есть поставщики и заказчики. Тогда на введении, прямо на странице рисуют схематично всё это. Условно обозначаю, кто соучаствует в процессе.

И, конечно, какая проблема стоит, какие решения есть (существующие разработки), какие у них недостатки. Напр., устарела и т.п. Т.е. почему возникла необходимость создания нового ПО, в самом общем виде – в том виде, в каком описывает заказчик. Например «большая доля ручного труда, следует автоматизировать, чтобы снизить человеческий фактор ошибок и повысить производительность».
2. Основания для разработки. «На основании приказа такого-то по корпорации», или «на основании учебного плана кафедры ИУ7, разработать техническое задание и программное обеспечение в рамках учебного курса распределённые системы».
3. Назначение разработки. Необходимо веско аргументировать назначение программы. Всё это обосновать детально. «Аналогичные разработки существуют, но они обладают рядом недостатков», «проанализировав предметную область, я выявил слабые места:
 1. Ручной труд,
 2. Неэффективно работает существующая система,
 3. Избыточная функциональность существующей системы,
 4. 5. 6. ... другие недостатки,
7. Необходимо добавить конкретную функциональность в рамках сущ.системы».
4. Требования к программе (к программному изделию, комплексу).
 - 4.1. Сначала высокоуровневые требования – это требования, которые касаются абсолютно всех подсистем, которые я интегрирую. Здесь можно сослаться на политические, юридические или финансовые документы, на основе которых ПО будет функционировать.

Режим:. Напр., хочу разработать систему, которая должна будет работать круглый год 24 часа в сутки. Пишут: «режим функционирования системы 24/7/365». Или «ежедневно», «ежедневно», «только по ночам» и т.д.
 - 4.2. Функциональные требования.
 - 4.2.1. Ф.Т с точки зрения пользователя. Требования будущих функций системы с точки зрения пользователя. Напр., «система должна предоставить возможность пользователю просмотреть список заказов», и т.п. Короче, всё, что может сделать система для пользователя.
 - 4.2.2. Ф.Т. с точки зрения администратора. У него полномочий больше. Администратор может не только читать, но и, в отличие от пользователя, но и изменять и т.п. Перечисляем всё, что может сделать администратор в рамках системы. Админ и юзер – разные роли. Если есть другие роли (напр., менеджер) – то и его описываю.
 - 4.3. Требования к функциональным характеристикам. Это «какие показатели» она будет выдавать с точки зрения реактивности системы – время реакции на запрос пользователя. Напр., «одновременно может работать NN клиентов. Если количество клиентов больше, то время отклика увеличивается».
 - 4.4. Надёжность. Следует указать уровень надёжности, который обязан быть у системы, и время восстановления системы после сбоя. Здесь следует описать, как производится

контроль входной и выходной информации. Иногда этим занимается специальный блок контроля. Создание резервных копий.

4.5. Условия эксплуатации. «Температура, влажность» и т.п. Напр., «система должна эксплуатироваться в нестандартных условиях (особые условия, экстремальные):». Т.е. это те технические характеристики, которые должна учитывать программа в своей работе. Этот пункт часто не пишут. Пишут его только в программно-техническом комплексе.

4.6. К составу и параметрам технических средств. Здесь студенты часто делают ошибки. Мы отрабатываем программу на своём ПК (с его конфигурацией). А потом студенты заявляют что-то вроде «прога работает на всём!!», но ведь не факт! Она и медленнее работать будет. Или не совместима с чем-то. Надо протестировать, и только протестировав, мы можем написать «система может работать под.... (и перечисляю всё то, под чем тестировал)». Обычно пишут, указывая минимальные требования и рекомендуемые. «Минимальные требования к технической платформе и операционному окружению». «Техническая платформа: (прописываю всё детально)». «Операционное окружение: (какое ПО, какие платформы, какие версии и т.п.)». Рекомендуемые – это те требования, на которых будет выполняться время отклика и т.п., т.е. программа будет работать наилучшим образом.

4.7. Требования к информационной и программной совместимости. Предположим, что-то надо оптимизировать, и делаем это через Matlab. Тогда надо указать, что мы не сами модуль делали, а использовали «сторонний». Предполагаемые методы решения, язык и среду программирования, а также другие программные средства, которые должны взаимодействовать (и каким образом) с нашими программами. С чем программа не может сосуществовать (конфликты), и т.д.

ТЗ – это язык промежуточный между профессиональными программистами и заказчиками. Поэтому не должно быть никаких профессионально-жаргонных слов. «Осуществляем back-up» – не годится. Если их не избежать, то следует создать раздел «Глоссарий», в котором требуется описать слова и их определения. Там следует указывать всю терминологию предметной области.

По поводу протоколов. Если их диктует заказчик, то «основания протоколов передачи данных такие-то, по требованию заказчика». Если заказчик выдаёт сценарий работы системы, то пишу: «по требованию заказчика, сценарий работы следующий:». А если я сам что-то определяю, то не пишу, т.к. это не исходные данные, а что-то то, чего разработать следует в ходе выполнения работы.

В этом же разделе при необходимости указывают степень защиты. Т.е. нужно ли шифрование, или нет. Каким образом будет осуществляться шифрование. Весь ли трафик шифруется, или часть. В конце привожу список стандартов, на основе которых осуществляется шифрование. А в самом начале писать: «Данное ТЗ разработано на основе ГОСТа (и полное название госта)».

4.8. Требования к программной документации. Документация бывает технологическая, научная, пользовательская и т.п. Напр., разработали ПО, передаю её, коды и инструкцию. Пользователю передаётся только пользовательская документация, инструкцию по установке, инструкцию по использованию, и описание возможных сбоев с инструкциями по их устранению. Ещё инструкцию для администратора. Все остальные вещи – это моё know-how, передавать это не стоит, за исключением тех случаев, когда они сами хотят дорабатывать его и т.д.

5. Техничко-экономические показатели. Здесь указывается ориентировочная экономическая эффективность, предполагаемую годовую потребность и экономические преимущества.
6. Стадия и этапы разработки и содержание работ, с указанием сроков и исполнителей.
7. Порядок контроля и приёмки. Здесь указывать виды испытаний. В учебных проектах этот пункт писать не стоит.

В ТЗ самое главное – чёткость формулировок. Никаких лишних фраз. Все они должны звучать чётко, однозначно, грамотно. Чем отличается информационная система от пакета программ? Вот, примеры определений.

- Программа – это на языке программирования записанная последовательность операторов, которые должен выполнить компьютер, чтобы реализовать поставленную задачу.
- Пакет программ – совокупность программ, решающих задачи некоторой конкретной области. Напр., пакет графических программ, пакет аэродинамических задач и т.п.
- Программные комплексы – совокупность программ, совместно обеспечивающих решение задач конкретной предметной области.
- Программно-технические комплексы – бывают и такие 😊.
- Программные системы (информационные системы) – представляют собой организованную совокупность программ (подсистем), позволяющих решать широкий класс задач из некоторой прикладной области. Они используют единое хранилище данных. Источник данных у них более-менее консолидирован.
- Многопользовательские программные системы – те, которые взаимодействуют по сети (в т.ч. удалённые).

Основные эксплуатационные требования к программным продуктам.

1. Правильность – функционирование в соответствии с техническим заданием. Есть такой процесс – «валидация», когда есть ТЗ, и их реализация. Независимый эксперт сравнивает, насколько реализация соответствует ТЗ. Это оно и есть.
2. Универсальность – обеспечение правильной работы при любых допустимых данных и защита от неправильных данных.
3. Надёжность (помехозащищённость) – обеспечение полной повторяемости результатов при наличии различного рода сбоев. Тут прописывается, как и куда сохраняю информацию в случае сбоя, как обеспечиваю рестарт и восстановление после сбоя.
4. Проверимость – возможность проверки полученных результатов. Пусть, я разработал БД. И говорю: «СУБД берёт на себя то-то, контролирует то-то». Но в случае сбоя всё-таки возможны какие-то сбои целостности данных. В крупных компаниях есть такая практика – они запускают тестовые данные, благодаря которым идёт проверка целостности данных.
5. Точность результата – это обеспечение необходимой погрешности, не выше заданной.
6. Защищённость – обеспечение конфиденциальности информации. Что и где должно быть защищено, на каком уровне и т.п. По-хорошему, ещё надо это и тестировать.
7. Программная совместимость.
8. Аппаратная совместимость – возможность работы с каким-либо оборудованием.
9. Эффективность – использование минимально возможного количества вычислительных ресурсов. Этот параметр тестируется при сертификации программ. Здесь же – проверка на утечку памяти. Проверяется тестами на эффективное использование памяти.

10. Адаптивность – это этап эксплуатации, один из критериев качества. Говорит о том, на сколько разрабатываемое ПО зависит от установленного ПО и аппаратуры.
11. Повторная входимость. В случае сбоя необходимо любыми способами за *минимальное* время восстановить систему.
12. Реентерабельность – возможность использования несколькими процессорами параллельно. Если есть возможность что-то пустить в отдельный поток – надо пускать.
13. Предпроектное исследование предметной области. Сергей Викторович Горин часто смотрит на этот пункт. Говорят, что надо провести анкетирование. Это один из методов. Анкетирование. Составляю список вопросов (заранее). Прихожу, спрашиваю. Ответил – записываю. Не ответил – спрашиваю, у кого узнать можно.
Цель предпроектных исследований – преобразовать нечёткие понятия о целях в чёткие функциональные требования, которые запишу в ТЗ. Методы – анкетирование, беседа с ключевыми фигурами в этом предприятии. Напр., технический директор.

Существует два варианта неопределённости на этапе анализа предметной области (в предпроектных исследованиях)

1. Неизвестны методы решения – нет информационной постановки. Поэтому часто используют «эскизный проект».
2. Неизвестная структура автоматизированных информационных процессов.

На этапе предпроектного исследования пишем модель As-Is. Что я предлагаю изменить в последовательности в бизнес-процессов, чтобы всё это дело отэфективнить.

Диаграмма Use-Case (диаграмма прецедентов)

Взять систему rational rose – см. диск с инструкцией у преподавателя.

Общие требования к проектированию:

Все участники проекта должны подчиняться единым правилам и соглашениям. К таким правилам относится:

1. Стандарт проектирования системы. Обсуждаем набор необходимых диаграмм моделей на каждой стадии проектирования и степень их детализации.
2. Стандарт оформления проектной документации
3. Стандарт интерфейса пользователя.

Требования к конфигурации, к ОС, к настройкам case-системы.

Правила интеграции подсистем. Т.е. соблюдение регламента и т.п.

Стандарт оформления документации. (дописать).

Стандарт интерфейса пользователя.

1. Договариваюсь о едином оформлении экрана (т.е. пользовательского интерфейса).
2. Правило использования клавиатуры и мыши.
3. Правило оформления текста и помощи.

4. Перечень стандартных сообщений. Некоторые сообщения инициируют какую-то подсистему.
5. Правила обработки реакции пользователя. Эти правила должны быть жёстко зафиксированы.

Методология быстрой разработки приложений – RAD.

Он включает три составляющие:

1. Небольшую группу разработчиков.
2. Короткий чётко проработанный график (2-6 мес).
3. Повторяющийся цикл разработки, При котором разработчики запрашивают изменяющиеся требования и срочно вносят изменения в свои приложения.

Жизненный цикл по методологии RAD состоит из четырёх фаз (см. начало лекции 2).

Этапы RAD:

1. **Анализ и планирование требований.** Здесь определяются функции, и выделяются первоочередные, которые реализовать надо в первую очередь. В ТЗ выделяются жирным, а остальные – курсивом. Описываются информационные потребности (какие инфопотоки будет обрабатывать система), определяются материальные затраты – нужно ли что-то прикупить для реализации проекта (деньги на разработчиков, на ПО). Сроки. Результатом будет список приоритетных функций, предварительная и информационная модель и архитектура системы в общем виде (топология системы).
2. **Фаза проектирования.** Определяю, нужно ли CASE-средство, или нет. Это зависит от класса решаемых задач, от коллектива, от возможностей и т.д. Устанавливаются требования к необходимой документации и пишутся диаграммы, которые необходимы. Сколько нужно времени на реализацию каждой подсистемы. Результатом является полная информационная модель в целом с указанием взаимодействий между подсистемами. Прототипы экранов, прототипы отчётов, выходных форм, диалоговых форм.
3. **Фаза построения.** Разработка кодов. Результатом фазы является готовая система, которую я представляю на beta-тестирование пользователям.

Структурный подход в проектировании.

Методологии структурного проектирования.

Структурный подход – это подход «сверху вниз», когда

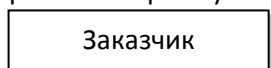
1. «разделяй и властвуй»: мы большую проблему разбиваем на несколько подзадач (осуществление декомпозиций) – и так до тех пор, пока задачи не станут совсем простыми.
2. Выстраиваем эти подзадачи в виде иерархического дерева. Т.е. акцентируем внимание на тех связях, что идут сверху вниз и игнорируем те, которые идут по горизонтали.
3. Принцип абстрагирования: выделение существенных аспектов, и отвлечение от несущественных.
4. Принцип формализации: Необходимость строгого методического подхода к решению проблемы. Т.е. в ключе одних и тех же методологий решаем все задачи. А не то, чтобы одну систему одними методологиями, другую – другими...
5. Принцип непротиворечивости: согласованность и обоснованность всех элементов будущей системы.
6. Принцип структурирования данных: данные должны быть тоже также структурированы и иерархически организованны (хотя, конечно, не всегда: они могут быть и реляционно связаны).
7. Структурные методологии проектирования:
 - 7.1. DFD (Data Flow Diagrams) – диаграммы потоков данных

Методология Гейна/Сарсона.

Основными компонентами диаграмм являются:

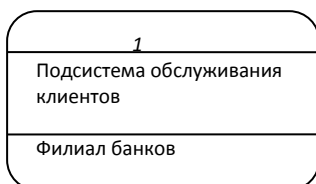
- Внешние сущности – это материальный предмет или физическое лицо, представляющее собой источник или приемник информации (заказчик, поставщик, склад)

Изображается прямоугольником с тенью(!) (по нотации Гейна/Сарсона):



Математический предмет или физическое лицо, предоставляющее собой источник или приемник информации

- Системы/подсистемы

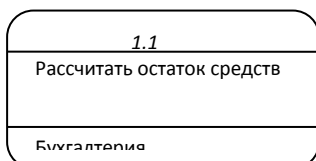


Поле номера

Поле имени
(существительное)

Поле проектировщика

- Процессы – преобразование входных данных в выходные по определенному алгоритму



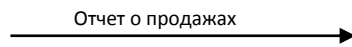
Поле номера

Поле имени (глагол в
неопределенной форме)

- Накопители данных, хранилище данных

| | |
|----|---------------------------|
| D1 | Хранилище данных о счетах |
|----|---------------------------|

- Потоки данных.



Нотация Йордена/Де Марка

- Внешние данные:



- Системы/подсистемы



- Хранилище данных

- Потоки данных:



7.2. SADT (Structured Analysis and Design Technologies) – стандарт IDEF0.

В 1970-1975 ВВС США приняли решение о компьютеризации всей промышленности в единых стандартах. Программа называется: ICAM (integrated computer Aided Manufacturing). Начали ее, и сразу трудность: все работают в своих стандартах. Эта программа породила необходимость создания такого стандарта. Поэтому создали методологию IDEF (ICAM Definition):

- Первая часть этой методологии IDEF0 – функциональная модель системы создается для того, чтобы отобразить структуры и функции системы, а также потоки информации и материальных объектов, связывающих эти функции.

Достоинства (завоевал мир!):

Графический язык, позволяющий точно и однозначно выразить весь спектр производственных деловых, производственных и других процессов предприятия на любом уровне детализации. Этот язык прошел многократную проверку в крупных проектах

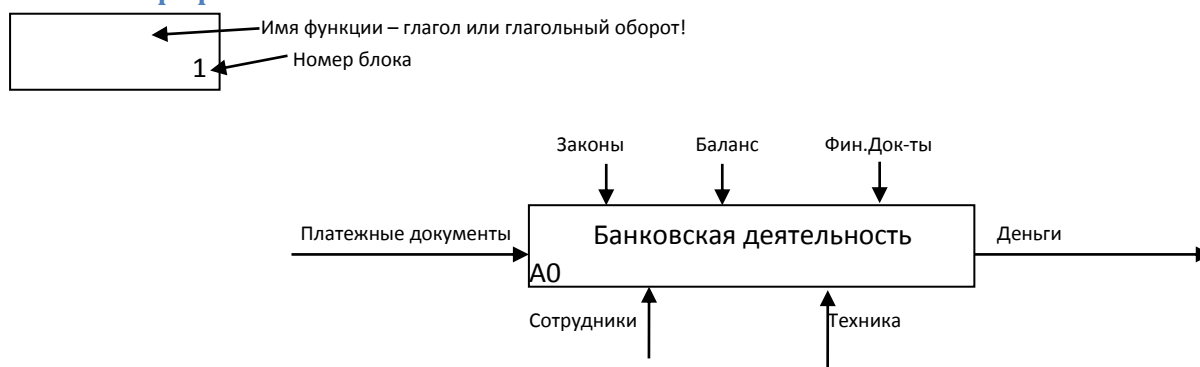
Лёгок и прост в освоении и изучении. Облегчает взаимодействие и взаимопонимание системных аналитиков, разработчиков, обслуживающего персонала и других, кто связан с производством.

- Вторая часть IDEF1 – применяется для построения информационной модели системы, отображающей структуру и содержание информационных потоков, и поддерживать эту систему в работу.
- Третья (забыта) IDEF2 – динамическая модель, отображающая динамику системы. Но, к сожалению, система не прижилась нигде, и не используется. На ее место пришли IDEFx и IDEF3.
- IDEF3 – это бизнес-процессы
- IDEFx – структуры данных.

Россия тоже стала стандартизировать.

7.3. ERD (Entity-relationship diagrams) – диаграммы сущность-связь.

Синтаксис графического языка IDEF0.

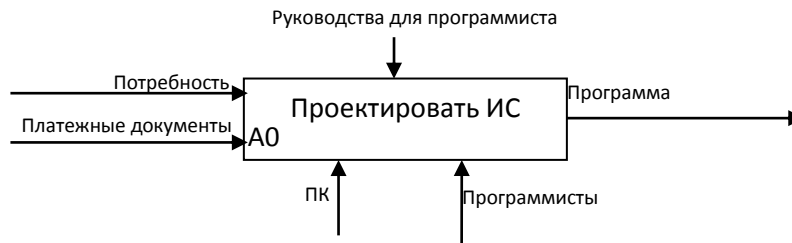


Контекстная диаграмма самого высокого уровня: «Проектирую информационную систему».

Система представляет собой совокупность взаимосвязанных частей, выполняющую полезную работу.

Модель – описывает, что происходит в системе, какие функции она выполняет.

Функция – действия.

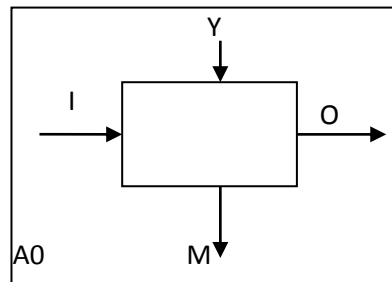


Цель: оценка трудоемкости разрабатываемого проекта, планирование, определение трудоемкости проекта.

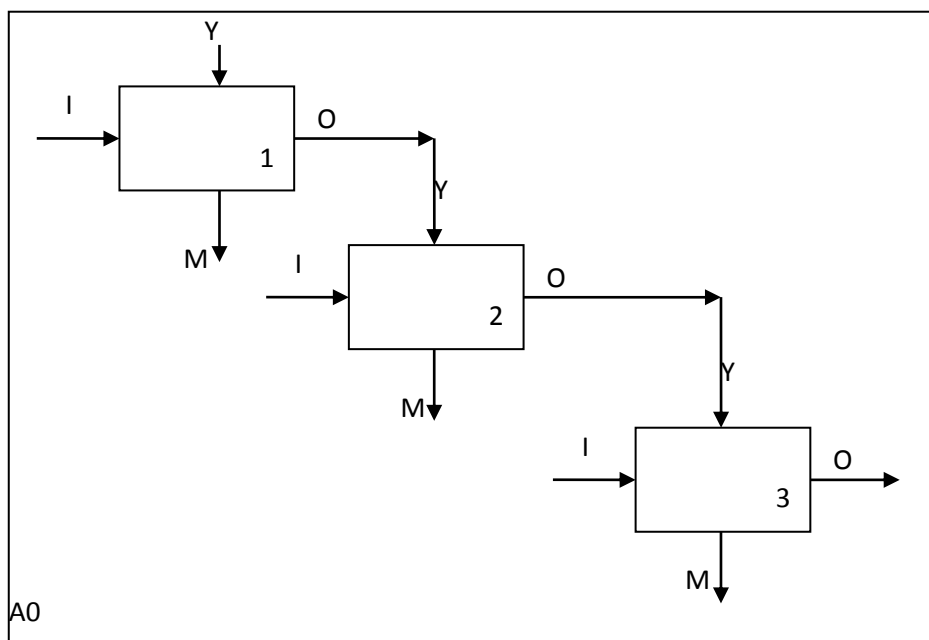
Точка зрения: служба информационной интеграции (служба управления проектами).

Когда проектируем, то становится много точек зрения. И все тут.

Декомпозиция.



Декомпозирую:



Очень важно направление стрелок: поступать может только слева или сверху, а выходить справа или снизу.

На каждой страничке не может быть больше 6, ну, максимум, семи блоков. Иначе просто это нельзя прочитать и осознать! На стрелочках пишутся информационные потоки.

Если все правильно определили, все описали на стрелочках, правильно указали средства и механизмы – то каждый участник производства все правильно поймет (если не дурак).

Цель моделирования должна отвечать на следующие вопросы:

1. Почему этот процесс должен быть смоделирован?
2. Что должна показать эта модель?
3. Что может получить читатель этой модели.

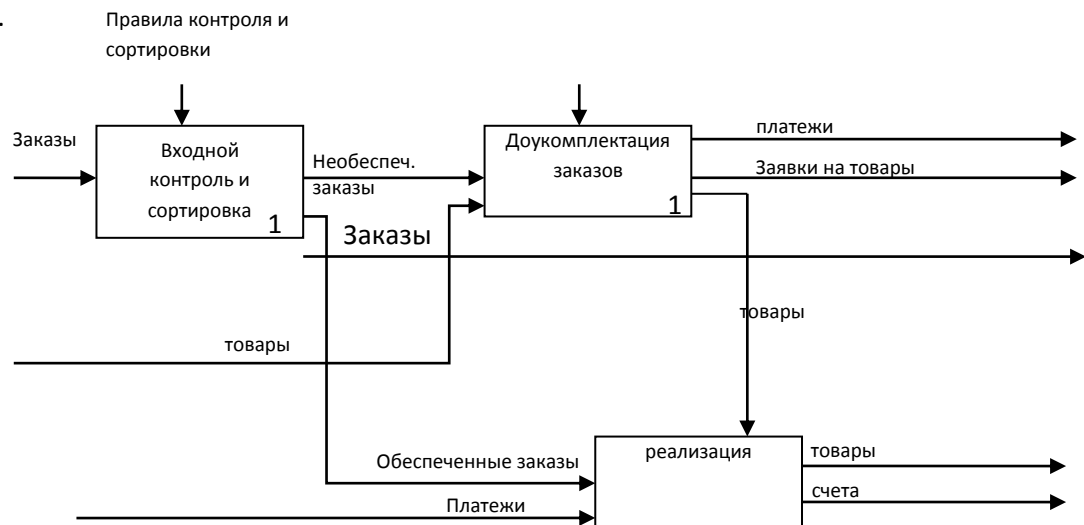
Точка зрения представляет собой взгляд человека, который видит систему в нужном аспекте моделирования (технолог, оператор, руководитель и т.д.).

Модель IDf3

Это метод описания процессов. По идее, должна хорошо отражать динамику. Это система, которая позволяет описать *временные* взаимоотношения между бизнес-процессами на предприятии.



Пример.



Какую методологию выбирать при проектировании? Любую, какая подходит больше. Чаще – комбинировать.

Отношения – связь между двумя или более сущностями.

Отношения именуются глаголами («имеет», «определяет», «владеет»), и они должны быть однозначными, а не типа «редактировать», «модифицировать» - не понятно, что за этими словами стоит.

Какими бывают сущности и отношения в ERD в нотации Петер Пин-Шен Чена.



Независимая сущность – представляет собой независимые данные. Т.е. отношения с другими сущностями могут существовать, а могут и отсутствовать.

Зависимая сущность – представляет собой данные, которые зависят от других сущностей

Ассоциативная сущность – это данные, которые ассоциируются с отношениями между двумя или более сущностями.

Неограниченное (обязательное) отношение – безусловное отношение, которое всегда существует, пока существуют относящиеся к нему сущности.

Ограниченное (необязательное) отношение – отношение, которое существует в некоторых условиях.

Существенно ограниченное отношение используется, когда соответствующие сущности взаимозависимы в системе.

Следующий этап детализации... переходим от отношений к связям...

Типы отношений.

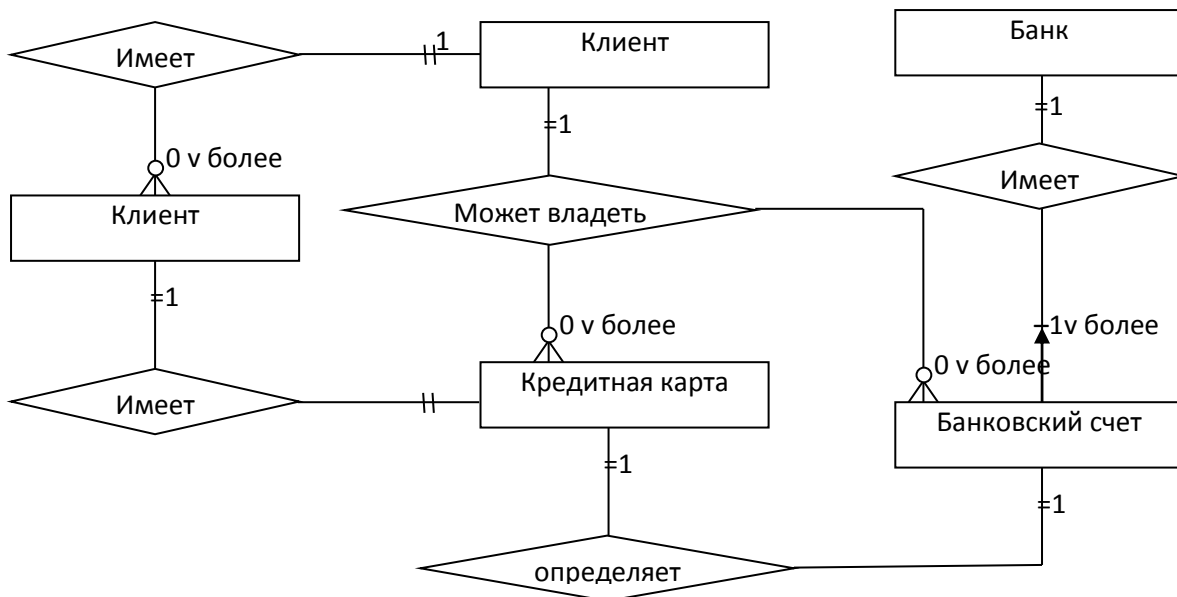
1. 1*1
2. 1*n (т.е. 1* ∞)
3. n*m (т.е. ∞ * ∞)

Первая нормальная форма – это схема без повторяющихся групп.

Вторая нормальная форма – все ее не ключевые атрибуты полностью функционально зависят от (простого) ключа.

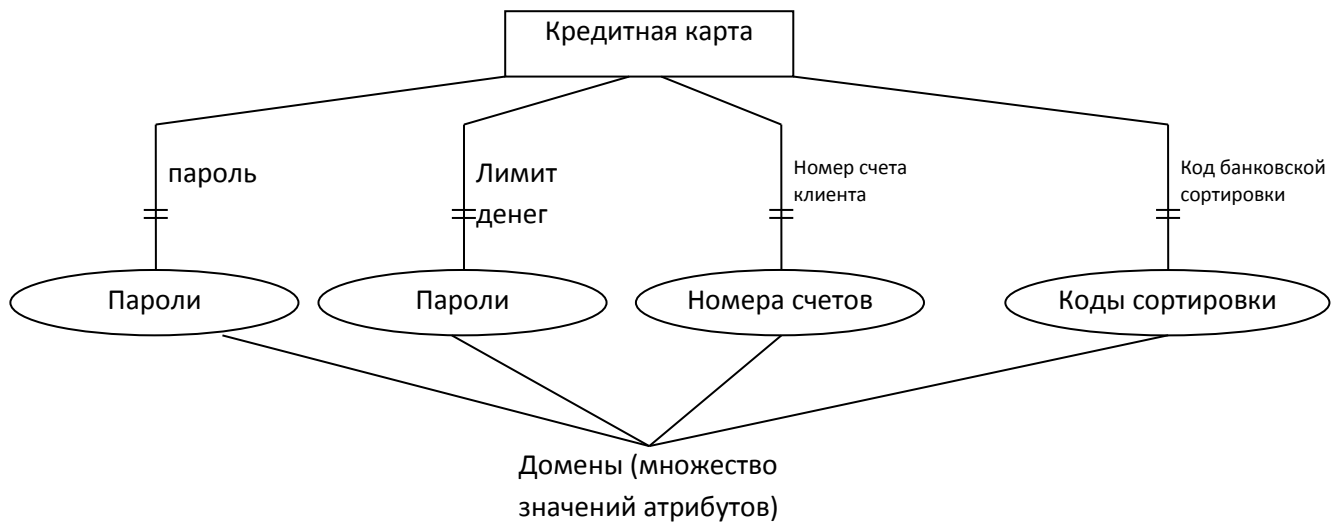
Третья нормальная форма – никакой из неключевых атрибутов не зависим от неключевых.

Пример ERD в нотации Чена.



Никаких атрибутов на этой схеме не обозначать! Для этого есть диаграмма атрибутов.

Диаграмма атрибутов



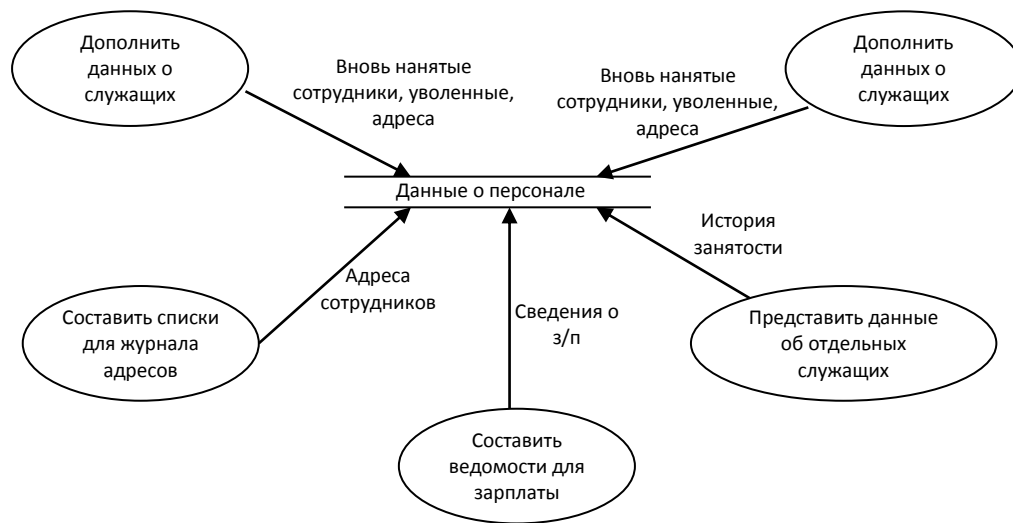
Атрибут – это характеристики. Но Чен ввел математическое понятие атрибута.

Атрибут – это математическая функция, отражающая набор сущностей во множество значений этих сущностей. Либо функция, отражающая набор связей в набор значений связей или даже их декартовых произведений.

$$f : E_i \rightarrow V_i \approx f : R_i \rightarrow V_i (V_1 \times V_2 \times V_3 \times \dots).$$

Диаграмма потоков данных (ДПД, DFD)

ДВД, регулирующая потоки данных фирмы по нотации Йордон/Де Марко



Презентация «Принцип управления распределенной информацией».

Дейт выставил 12 признаков распределенной системы. Среди них есть те, которые практически могут быть реализованы в веке так 25-м 😊

Например: идеальный вывод.

Фирма IBM выставила 4 основные операции, которые реально сейчас можно реализовать и реализовываются:

- Удаленный запрос
- Удаленная единица работы (транзакция)
- Распределенная единица работы
- Распределенный запрос.

15.03.2011.

Сегодня рассмотрим:

Языки моделирования при использовании ООПарадигмы. История развития.

Язык «Унифицированный язык моделирования (UML)».

Методология RUP (Rational unified Process), Unified Modeling Language (UML).

1989 – 1994. Смутные времена в программной инженерии – не было единого языка моделирования. Каждый разработчик предлагал свой синтаксис, семантику и изобразительные средства. Поэтому исторически сложились три поколения языка моделирования. Первое забыто (10 языков насчитывало). Второе насчитывало 50 нотаций. Очевидно, что была война методов. Групповая разработка – это вообще жопа была.

OMG. ER – диаграммы – надо ознакомиться.

Джекопсон ввел диаграммы Use-case. Буч предложил методологию структуры. Нотация Буча вошла в UML. Рамбо предложил диаграммы последовательностей действий, диаграммы взаимодействия внутри объектов, внутри классов.

UML – это не визуальный язык программирования, это язык позволяет генерировать коды на многих популярных языках программирования (C++, oPascal, ADA95, Java etc).

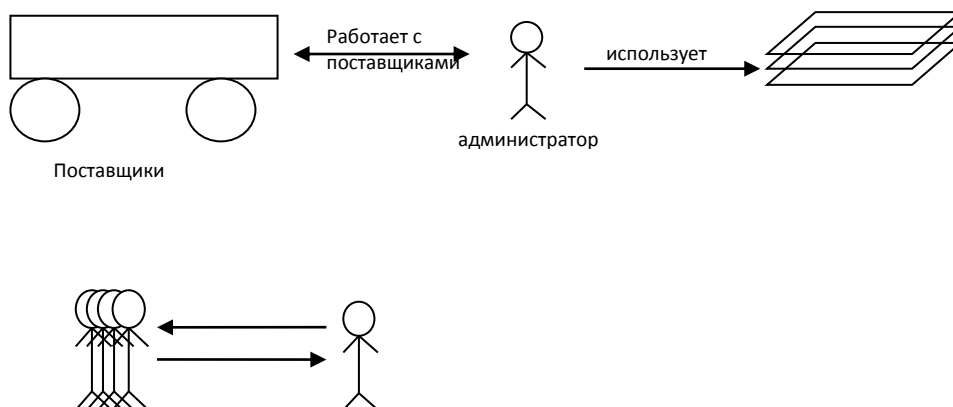
Методология RUP и пример ее использования на простом примере о торговой фирме.

«Методология» - это некая совокупность приемов, которые обозначают и регламентируют, как и что надо сделать, чтобы разработать качественный программный продукт в отведенные сроки с отведенным бюджетом.

Методология RUP отвечает на следующие вопросы: Какие шаги следует предпринять, чтобы уложиться в отведенные сроки с отведенным бюджетом, построить необходимый программный продукт с заявленными требованиями, удовлетворяющими заказчика. Вот, какие это шаги?:

1. Описание предметной области.

Пример. Есть торговая фирма, несколько торговых точек, администратор (работающий с поставщиками)



Это была модель предметной области.

1.2. Сценарий работы данной торговой точки.

Прописываем теперь все четко словами (то, что нарисовали). С высокой степенью детализации:

- У оптовых поставщиков закупается товар.
- Закупленный товар помещается на склад торговой точки.
- Продавец отпускает товар с торговой точки, принимает деньги, выдает товары со склада.
- Продавец фиксирует все продажи в журнале продаж и сдает в конце дня администратору вместе с выручкой.
- Прибыль от работы торговой точки формируется за счет разницы между закупочными и продажными ценами на все товары.

1.3. Что следует автоматизировать в данной коммерческой структуре.

- Накопление журналов продаж, получаемых от продавцов.
- Хранение информации о текущем состоянии на складе (фиксация расходов и доходов).
- Получение отчетов о приходе, расходе, прибыли и выручке.

2. Глоссарий. Это спецификация терминов и понятий предметной области, которые потенциально могут быть затронуты при написании этого проекта. Требования: описать словами основные понятия предметной области развернуто, однозначно и очень точно. Он вводится для того, чтобы закрепить определенный смысл определенных понятий.

Глоссарий создается еще в тот момент, когда система еще только начинает проектироваться, поэтому глоссарий не должен содержать тех понятий, которые касаются реализации. Напр., «метод, который делает то-то и то-то».

Глоссарий (в рамках примера):

| Название термина | Описание термина |
|------------------------|---|
| Товар | Материальная единица, подлежащая продаже на торговой точке. Товар поступает на торговую точку от поставщиков по закупочной цене и продается продавцами по продажной цене. |
| Журнал продаж | Формируется продавцами в течение всего рабочего дня. Содержит в себе информацию о том, сколько, какого товара было продано в течение рабочего дня на торговой точке. |
| Выручка | Суммарная стоимость всех товаров, проданных за день на торговой точке. Присутствует выручка двух типов: фактическая и прогнозируемая. |
| Прогнозируемая выручка | |
| Фактическая выручка | В идеале совпадают. При их несовпадении получается недостача |
| Недостача | |
| Закупочная цена | |
| Продажная цена | |
| И т.д. | |

3. Совладельцы системы (роли).

Совладелец – такой персонаж, который имеет прямое или косвенное отношение к проектируемой системе. В нашей системе – продавец и администратор.

Совладельцем в данной системе является администратор торговой точки. Он же – единственный пользователь автоматизированной системы, принимает товары, смотрит отчеты и т.д.

Вторая роль – продавец. Он не взаимодействует с системой, а значит, не является его пользователем. Он фиксирует в журнале.

4. Границы системы.



5. Функциональные требования к системе. Т.е. что должна делать система. «Система **должна** ...». FR = functional Rub. Нумерация идет через 10, чтобы можно было вписать детализацию.

| Номер | Текст |
|-------|--|
| FR010 | Система должна позволять просматривать информацию о продавцах, зарегистрированных в системе |
| FR013 | Система должна позволять вводить информацию о новых продавцах. (13 – это мы решили сразу детализировать) |
| FR016 | Система должна позволять удалять уволенных продавцов из системы. |
| FR017 | Система должна отображать информацию о состоянии склада (кол-во, список, цены) |
| FR020 | Поиск, Различия товаров, возможность сделать пересчет (напр., скидка и т.п.). |
| | Отчет-приход должен содержать следующую информацию: дата, код, наименование, кол-во, закупочная цена за единицу, отпускная цена. |

6. Технические требования.

TR – technical Rub.

| Номер | Текст |
|-------|--|
| TR010 | Система должна обеспечить хранение информации о следующих объектах: <ul style="list-style-type: none"> • Товар • Продавец • Журнал продаж |
| TR020 | Хранение данной информации должно осуществляться под управлением СУБД: (пишу, какой). |
| | И т.д.: какая СУБД, какой язык, какая ОС, какая платформа. |

Язык UML, диаграмма вариантов использования.

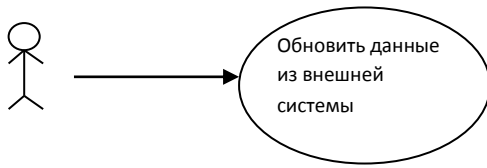


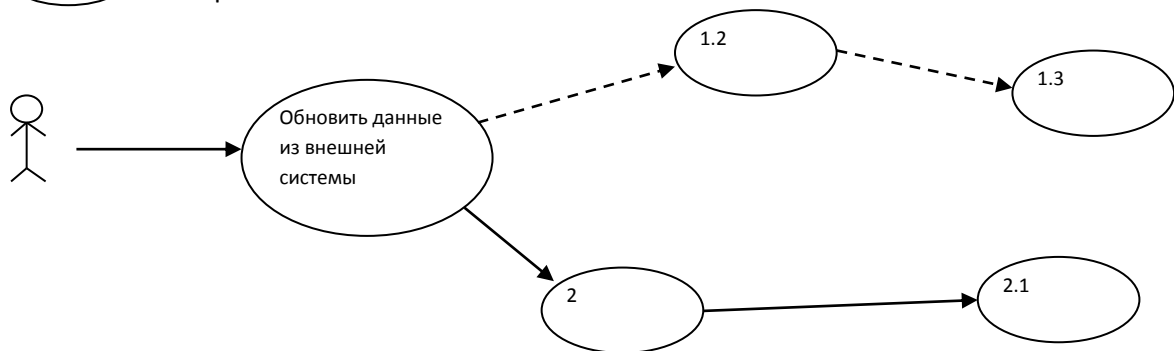


Диаграмма прецедентов Use-Case. Пример.

Диаграмма сценариев и диаграмма прецедентов – не совсем одно и то же!

 Актер – это кто-то или что-то, кто взаимодействует с нашей системой, но к нашей системе не принадлежит.

 - это прецедент. Это действие, которое обеспечивает некую реакцию на действия актера.



Спецификации прецедента «обновить данные из внешней системы».

1. Актер вызывает функцию обновления данных.
2. Система запрашивает подтверждение «обновления данных из внешней системы приведет к потере существующих данных в текущей системе, реально обновить что ли?»
3. Актер подтверждает обновление (а может и не подтвердить, тогда прецедент не совершится).
4. Система обращается к внешней системе (к ее БД, например) с запросом получения данных.
5. Система получает данные из внешней системы и сохраняет их.
6. Система выдает сообщение об успешном завершении обновления данных.
7. Актер подтверждает прочтение сообщения
8. Выполнение сценария заканчивается.

Возможные отношения между сценариями.

1. Расширение (Extend) – обозначает, что один сценарий может расширять поток событий, протекающих в другом сценарии. Обозначается $\leftarrow \ll\text{extend}\gg$

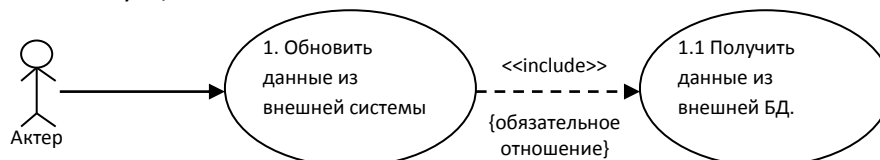
Необязательное отношение говорит о том, что если пользователь даст отказ, то этот сценарий выполняться не будет.

Напр., мы пришли в ресторан. Там есть завтрак, обед, ужин. Ужин есть всегда в любом ресторане, А мы можем заказать «расширение»: тот же ужин, но в отдельной комнате, при свечах и с цыганами 😊



2. Включение (Include)

А вот в этой ситуации:

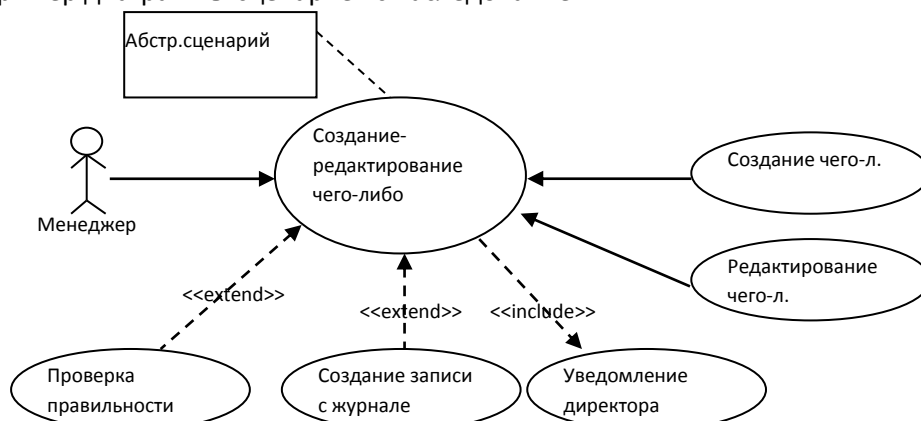


Система спрашивать ничего не будет, а сделает это в обязательном порядке.

Помним, что много расширений на одной диаграмме делать не надо. Лучше разбить на несколько диаграмм.

3. Наследование (generalization)

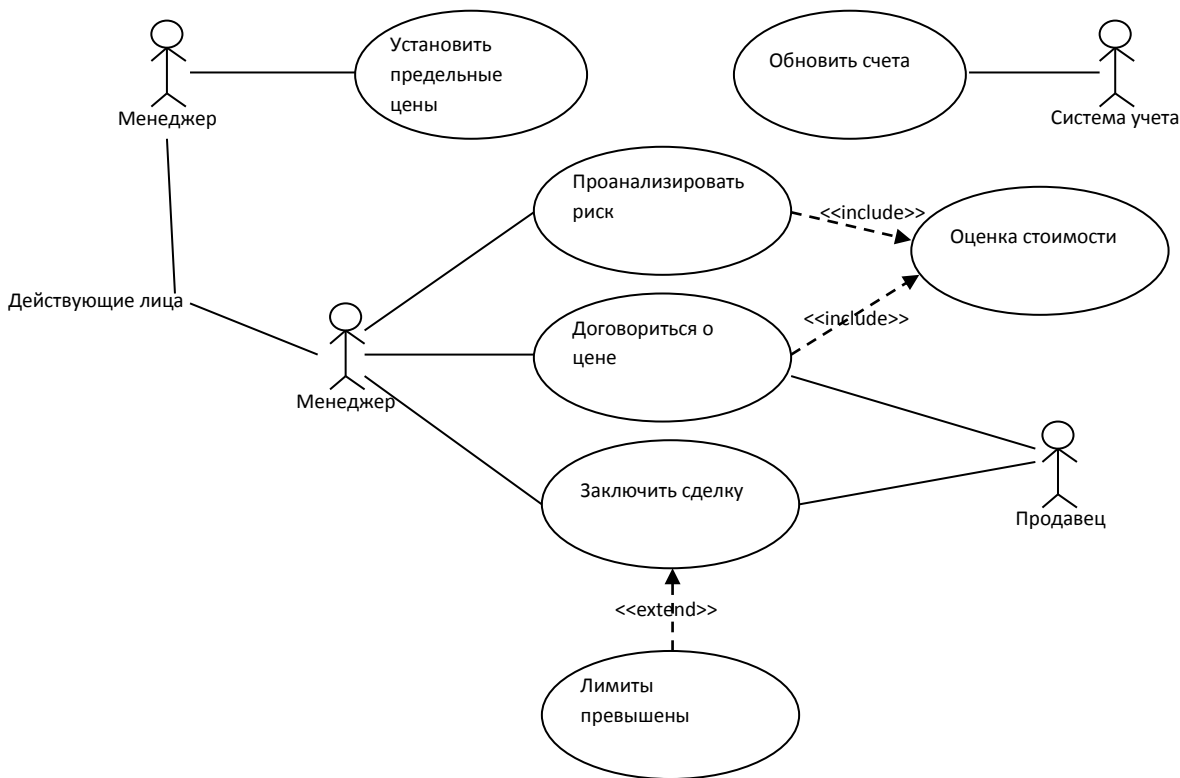
Пример диаграммы сценариев с наследованием:



Еще раз: актер – это роль. Часто роли совпадают в одном человеке. В таком случае это прописываем (под человечком я написал «менеджер» поэтому).

1. Применять связь extend, когда мы хотим описать изменения в нормальном поведении системы. Т.е. прописать альтернативные ходы событий.
2. Применять связь include, когда необходимо избежать повтора в двух или более вариантах использования.
3. Use Case призваны в графическом виде отразить потенциальные функциональные требования к проектируемой системе.

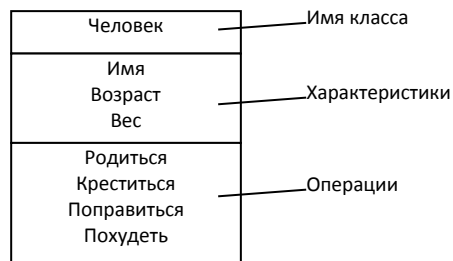
Пример вариантов использования. От менеджера идет не стрелка, А просто линия – это означает двустороннюю связь



UML, разновидности предметов, существующих в UML.

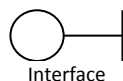
1. Структурные предметы – являются существительными

1.1. Класс – совокупность реаций действительного мира, которых объединяют общие, важные для данного проекта характеристики и поведения.



2.2. Интерфейс – это набор операций, который определяет услуги класса или компонента.

Имя интерфейса часто начинаются с буквы i. Интерфейс часто обозначают так:



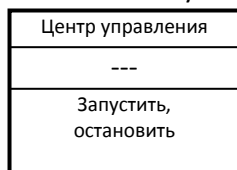
3.3. Кооперация (сотрудничество) – определяет взаимодействия и является совокупностью ролей или других элементов, которые работают вместе для обеспечения коллективного поведения. Пример не привела. Говорит, нету.

3.4. Актер – набор согласованных ролей, которые могут играть пользователи при взаимодействии с системой. Каждая роль требует от системы определенного поведения.

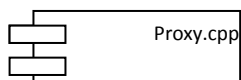


3.5. Прецедент Use Case – описание последовательности действий, или нескольких последовательностей, выполняемых системой в интересах отдельного актера и производящих видимый для актера результат.

3.6. Активный класс – класс, который имеет один или несколько очень важных процессов, и он может запустить



3.7. Компонент



3.8. Узел – техническое устройство, физический элемент, на котором находятся нами разработанные компоненты (платы, компьютеры, процессоры, устройства).

Изображается кубом с надписью на передней стороне.



2. Предметы поведения – динамические части UML.

3. Группирующие предметы

4. Поясняющие предметы и комментарии.

Операции

| |
|---|
| Видимость ИМЯ(Список параметров):ВозвращаемыйТип{характеристики} |
| Записать – только имя +записать – вид и имя Зарегистрировать (И:Имя, Ф:Фамилия) БанансСчета(): Integer Нагревать(){guarded} |

Характеристики операций.

Leaf – конечная операция в цепочке наследований, она не может быть полиморфной, и не может переопределяться.

isQuery – выполнение данной операции не изменяет состояние объекта.

Sequential – в каждый момент времени допустим только поток вызовов, в объект поступает только один вызов операции.

guarded – допускается одновременное поступление в объект нескольких вызовов. Т.е. параллельные потоки управления ставятся в очередь и выполняются последовательно. Тогда помечаем Этой характеристикой.

Concurrent – разрешается параллельное (множественное) выполнение операции.

Если атрибутов слишком много, то есть вариант – использовать многоточие. Иногда используют группировку, а под этими группами ставят многоточие.

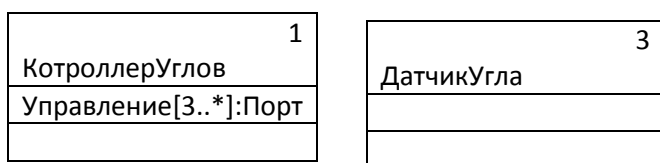
Группировка

Класс «обслуживание заказов»:

| |
|---|
| Обслуживание заказов |
| <<конструктор>> НовыйЗаказ() ... <<процесс>> ОбработатьЗаказ()(О:Заказ) ... <<проверка>> ПроверитьЗаказ() ... |

Множественность. Как ее обозначить?

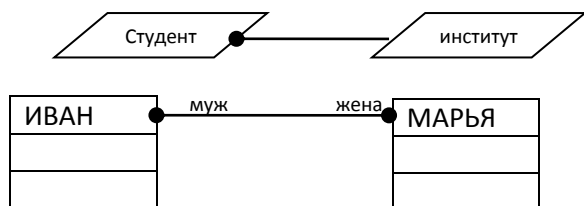
Множественность экземпляров в классе.



Отношение в диаграммах классов.

| | |
|-------------|------------------------|
| Ассоциация | _____ |
| Обобщение | подкласс суперкласс |
| Зависимость | зависимый независимый |
| Реализация | приемник источник |
| Агрегация | часть целое |
| Композиция | часть целое |

Ассоциация – отображают структурные отношения между экземплярами классов, т.е. между экземплярами объектов. Каждая ассоциация имеет метку (имя), которое описывает природу отношения.



Множественность ассоциативной связи:

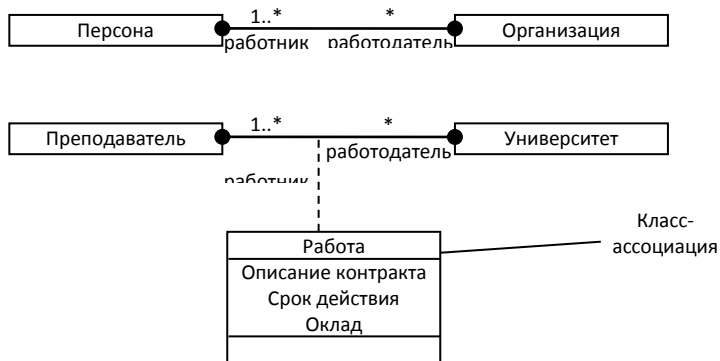
5 – ровно 5

* – неограниченное количество

0-* – ноль или более

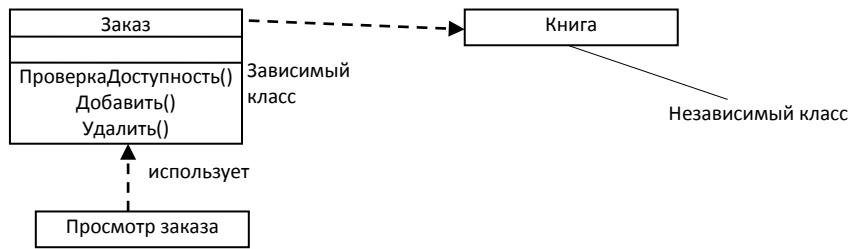
1..* или 3..7 – тип «диапазон»

3..7, 10 – «диапазон и число».

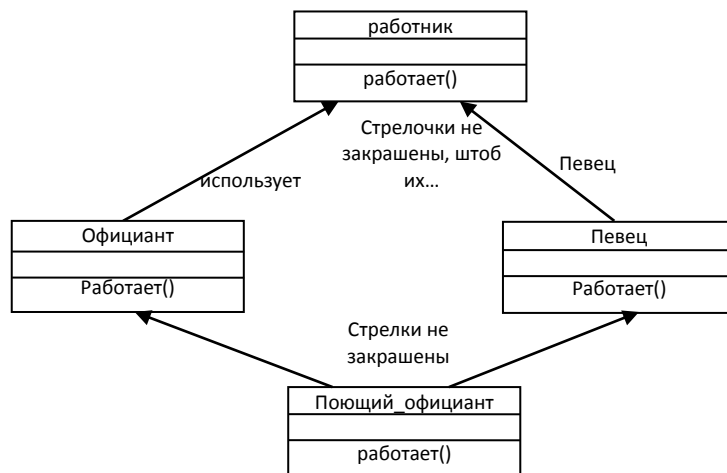


Отношения зависимости.

Зависимость является отношением между клиентом...



Реализация

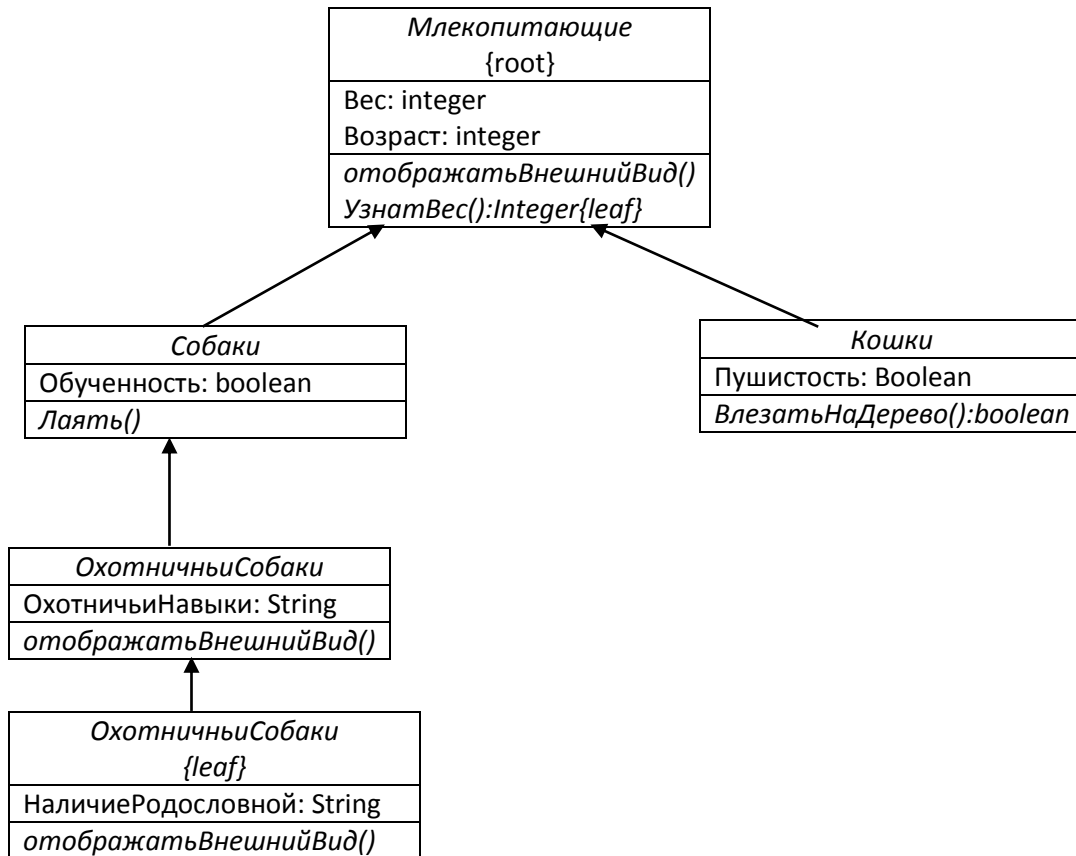


29.03.2011

Деревья наследования

Абстрактный класс.

{root} – тек, который обозначает абстрактный класс.

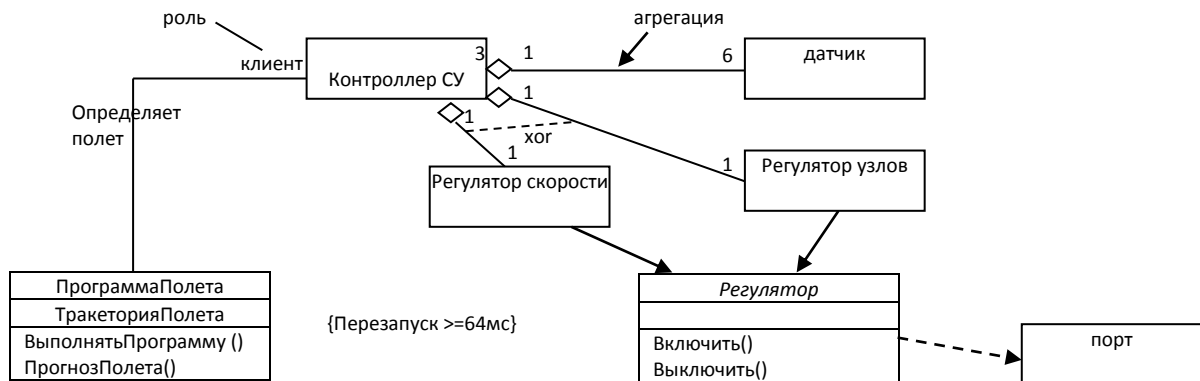


Полиморфная операция. Есть некоторая операция, которую наследуют наследники.

Полиморфизм – возможность с помощью одного имени обозначать операции из различных классов (но относящихся к одному суперклассу).

Диаграмма классов системы управления полетов.

Справа сверху ставится цифра NN – это количество экземпляров «НЕ БОЛЕЕ, ЧЕМ NN». Раньше она говорила, что это «ровно NN», но это она погорячилась.



Динамические модели отображают изменение состояния в процессе работы системы. Поэтому сюда входят автоматы и диаграммы взаимодействий.

Автомат (state-машины) – описывает поведение разрабатываемой системы в терминах последовательности состояний, через которые проходит объект в течение своей жизни.

Взаимодействие – описывает поведение в терминах обмена сообщениями между объектами.

Автомат

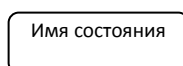
Автомат отображается с помощью:

1. Диаграмм схем состояний
2. Диаграмм деятельности

Взаимодействия (между объектами) отображаются с помощью:

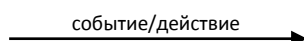
1. Диаграмм сотрудничества (кооперации, collaboration)
2. Диаграммы последовательности действий.

Состояние обозначается прямоугольником со скругленными углами.



Событие – происшествие, вызывающее изменение состояния.

Действие – набор операций, запускаемых этим событием.



Примеры событий:

| | |
|----------------------|---|
| Баланс <0 | – изменение состояния. |
| Помехи | – сигнал «объект с именем таким-то» (имя объекта). Что-то нужно изменить. |
| Уменьшить (давление) | – вызов действия, которое приводит к запуску какого-то другого блока. |
| After (5сек) | – по истечении временного периода что-то сделать. Т.е. каждые 5 секунд происходит какое-то событие. |
| When (time = 17.00) | – наступление абсолютного времени. |

Примеры действий, соответствующих этим событиям

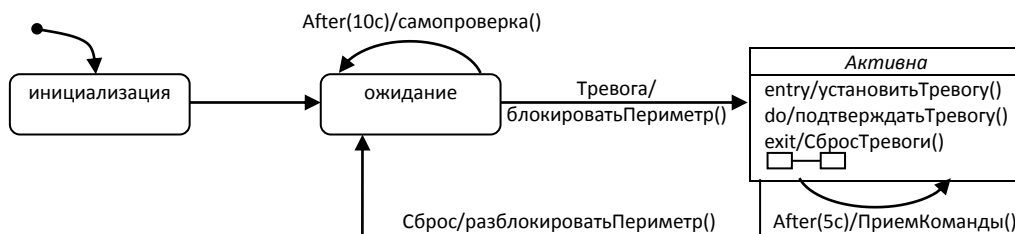
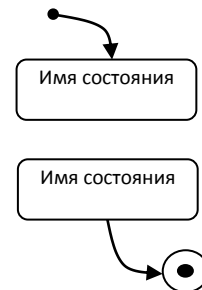
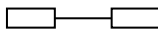
| | |
|--|--|
| Кассир.Прекратить выплаты() | – операция, которая прекращает выплаты, если баланс меньше нуля. |
| fit := new(фильтр); fit.убратьПомехи() | – запускается сразу два события. |
| Send Ник.Привет | – send используется для посылы сигнала. |

Обозначения:

Переход в начальное состояние:

Переход в конечное состояние:

«очки» означают детализацию этого состояния



С одной стороны, система включена круглые сутки. Поэтому нет блока «переход в конечное состояние». Но есть exit. Где его нарисовать в этой схеме – она сомневается. Может быть, где-то внутри «активна».

Пример сохранения истории.

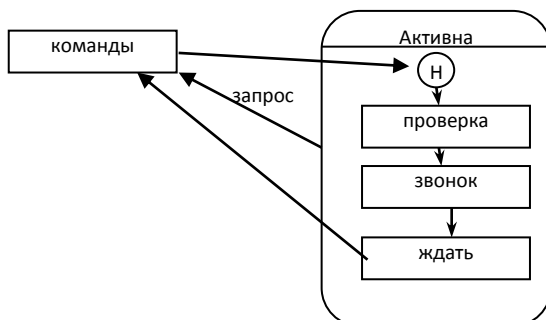


Диаграмма деятельности.

Диаграмма деятельности представляет собой особую форму конечного автомата, в котором показывается процессы вычислений и потоки работ.

Пример диаграммы деятельности.

Черная жирная линия называется линией синхронизации.

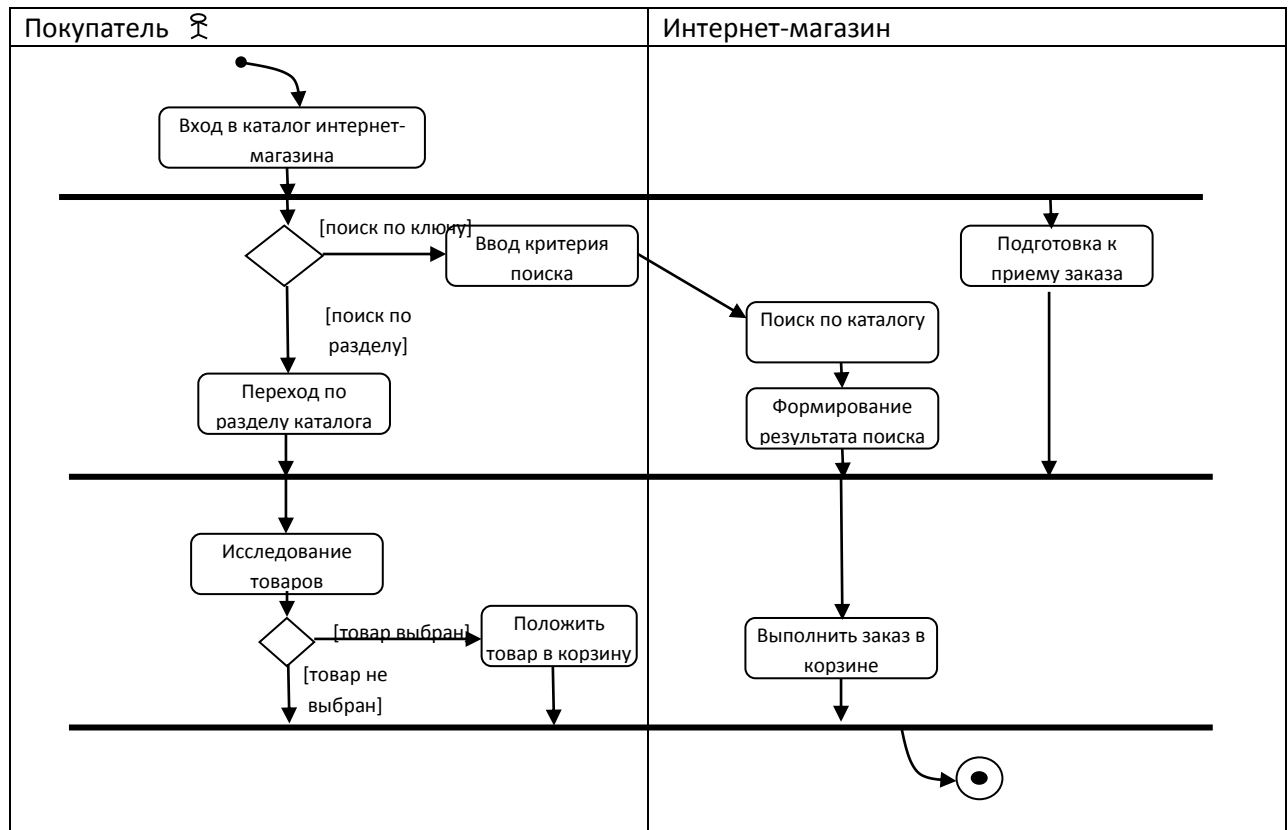
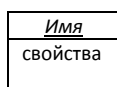


Диаграмма сотрудничества (кооперации).

Синтаксис объекта: ИмяОбъекта:ИмяКласса



- Обозначение объекта.

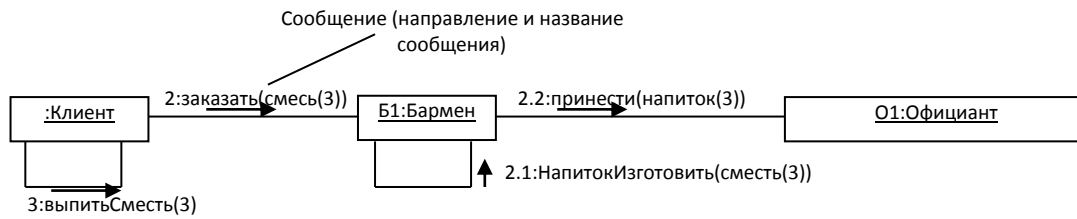
Если не хочу называть объект, то просто пишу: «:ИмяКласса»

Подчеркивание означает конкретный объект. Например:

Адам:Человек
:Пользователь
Агент:
МойКомпьютер:

Пример. Поток синхронных событий.

Имеем: клиент, бармен и официант. Любой объект из класса «Клиент» может выпить.



Система управления полетами.



А теперь то же самое, но диаграммой последовательности действий. В прямоугольниках если не подчеркнуто или не стоит имя класса перед «:» – то это класс. А нужен объект!

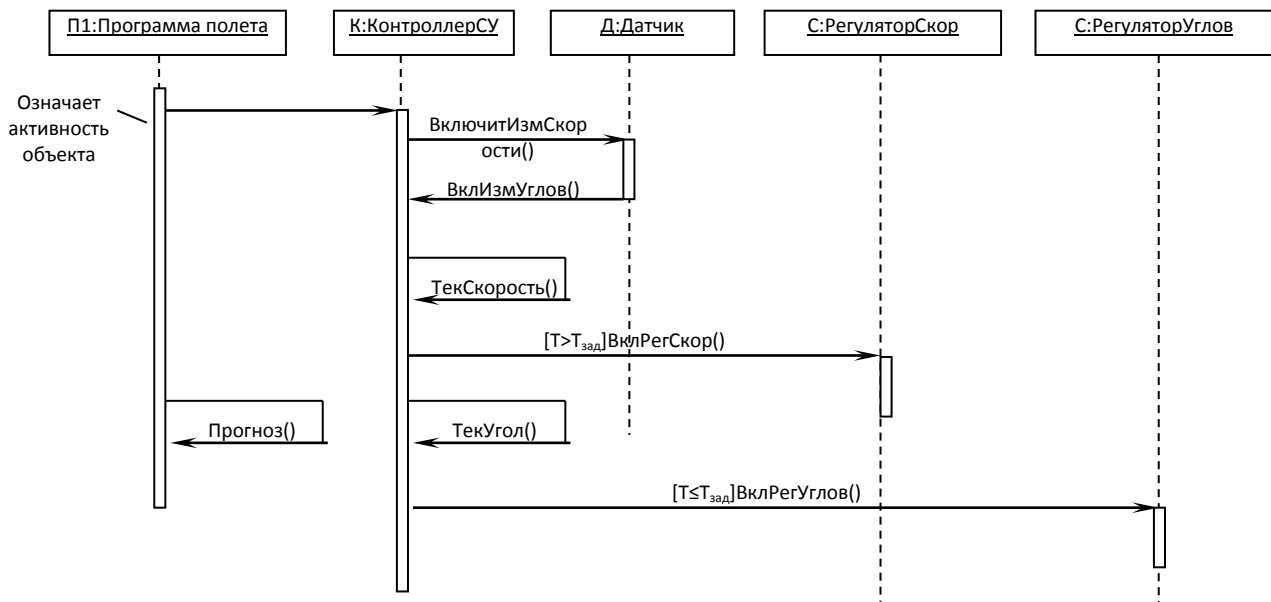


Диаграмма последовательности действий пользователя и автоматизированного кассового аппарата

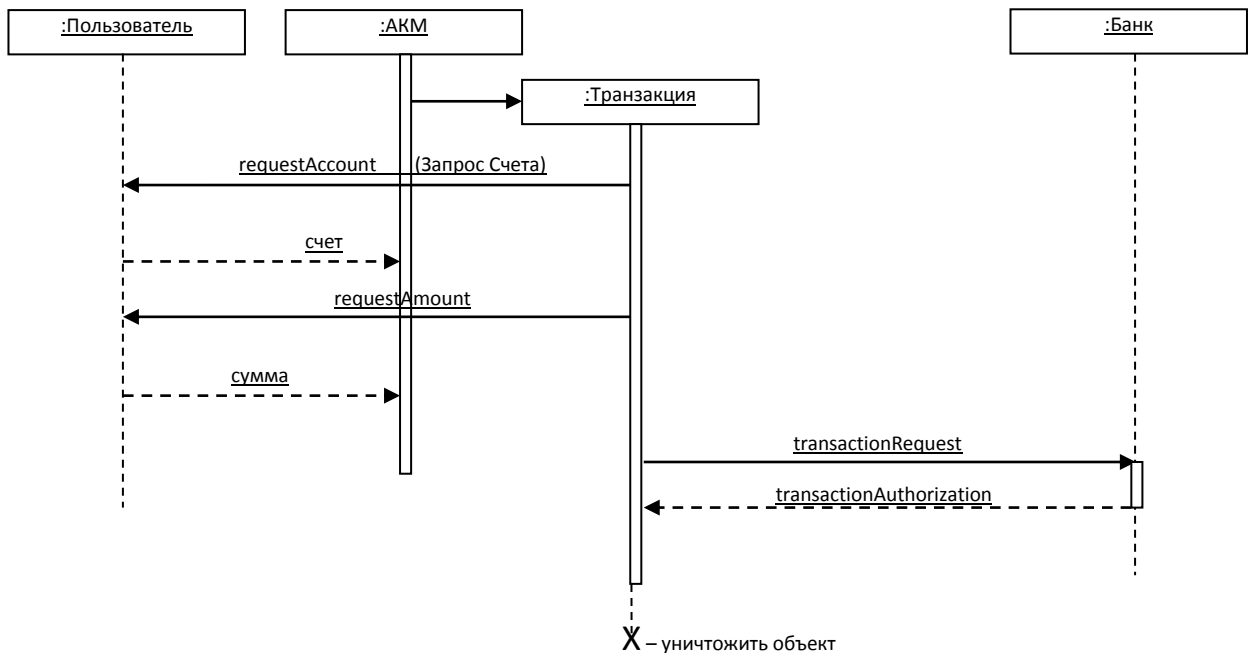
Стрелка со сплошной заливкой наконечника означает процедурную передачу управления.

Стрелка с двухреберным наконечником обозначает непроцедурную передачу управления.

Стрелка с односторонним наконечником обозначает асинхронное взаимодействие между объектами.

Пунктирная стрелка с двухреберным наконечником обозначает возврат из вызова процедуры.

:Пользователь – означает любой объект из класса пользователь.

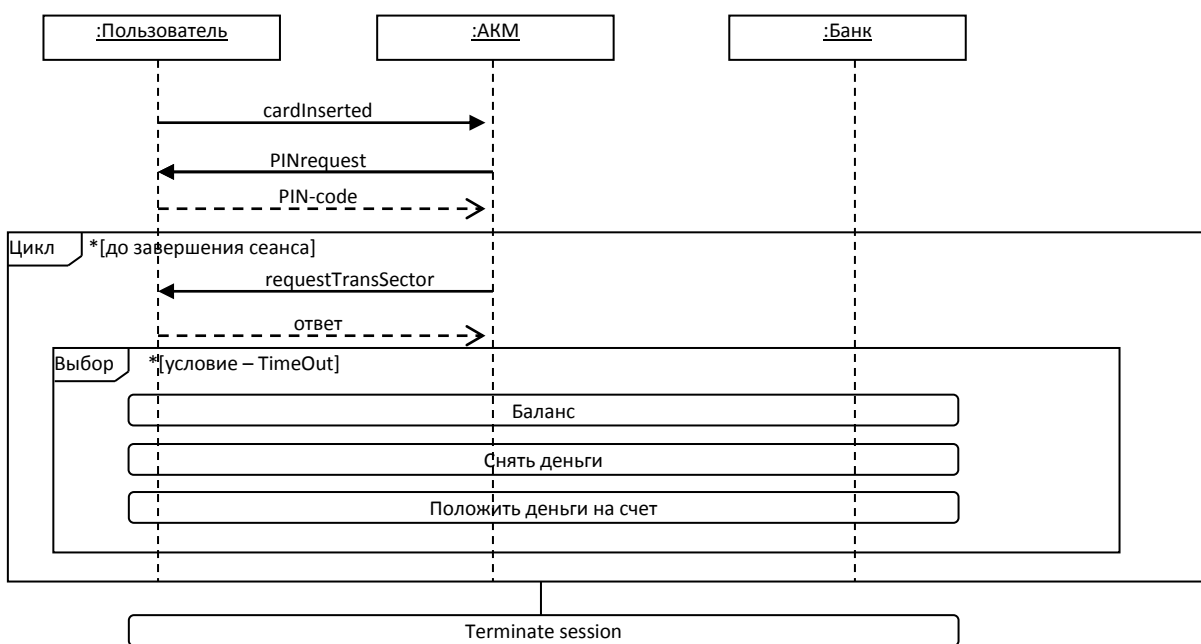


Мы хотим снять деньги. НО, прежде чем снять деньги, мы должны запросить, а есть ли они. Банк – огромная гетерогенная система, к которому мы, вроде, не имеем отношения. Но мы хотим обратиться к хостовому компьютеру этого банка – вот мы его и обозначили выше прямоугольником :БАНК на схеме выше.

Мы создали объект, и мы же его уничтожили на схеме выше.

Зацикливание, выбор вариантов и циклы.

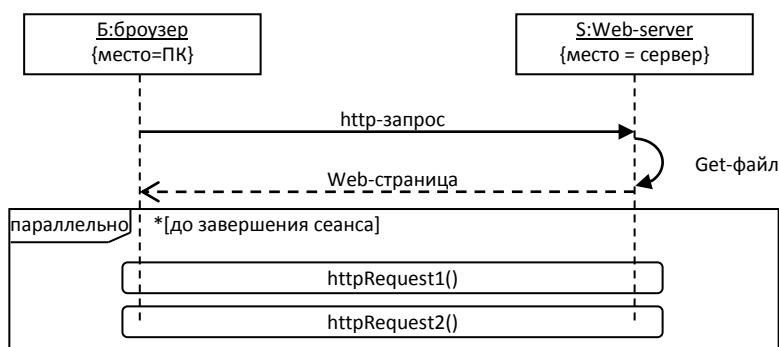
Транзакцию рисовать не будем, т.к. она сидит внутри.



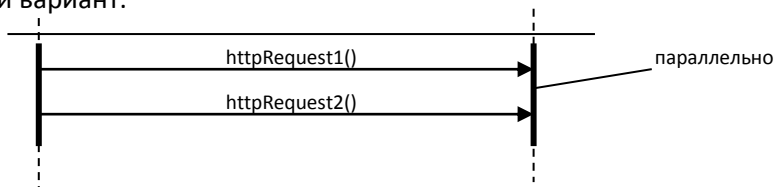
Теперь поговорим о параллельных процессах.

Например, мы осуществляем подкачку данных из БД на наш сайт. Можем распараллелить: пока система спрашивает что-то у пользователя, в этот момент может подгружаться картинка.

В общем, если я вижу, что и как можно распараллелить, то на диаграмме есть возможность это изобразить. Приведем пример на браузере и веб-сервере.



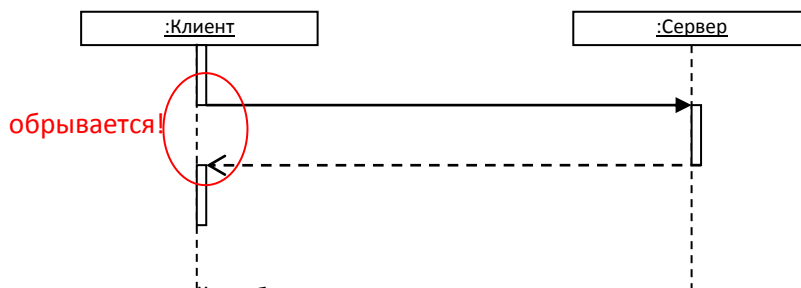
Есть другой вариант:



Синхронный запрос – это ... Он не будет активным, пока не получен ответ!

Асинхронный – это... можно и не дожидаться ответа.

Пример синхронного:

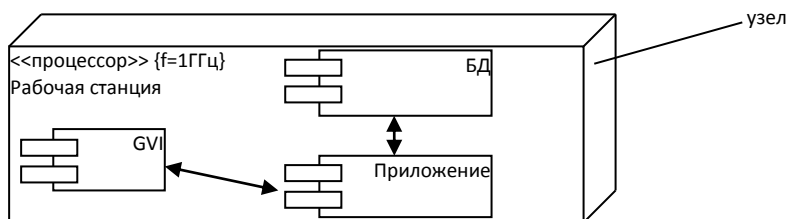


Синхронный с обратным откликом – нарисовала, передумала, стерла. Дала ссылку на презентацию.

Очень важный момент!:

Диаграмма развертывания. Это диаграмма, которая обозначает, На каком узле какие находятся модули, приложения.

Узлы изображаются в виде



Интероперабельность – различные системы (мои и не мои) должны вместе друг с другом взаимодействовать. Интероперабельность нужно отобразить и в ТЗ и на схемах.

Что такое распределенная система? Методичка: «мы обращались к стандарту для разработки автоматизированных систем, то мы разрабатываем автоматизированную систему».

ТИТУЛЬНЫЙ ЛИСТ ДЕЛАТЬ ПО ЕСПД.

Короче... автоматизированная система – это немного не то, что распределенная, а распределенная – это немного не то, что автоматизированная.

Вопросы распределенных единиц работы, итероперабельности – нужно оттенить в ТЗ

05.04.2011

Презентация. И книжка Уильямса.

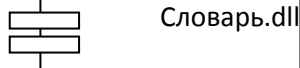
Оценка производительности распределенных информационных систем на этапе проектирования

Чтобы оценить, правильно ли выбраны архитектурные решения, будет ли отвечать заявленной производительности – следует это оценить на этапе тестирования.

Модели реализации.

Компонентная диаграмма и компонент.

Компонент изображается так. Это физическая и заменяемая часть системы, которая соответствует набору интерфейсов и обеспечивает реализацию этого набора интерфейсов.



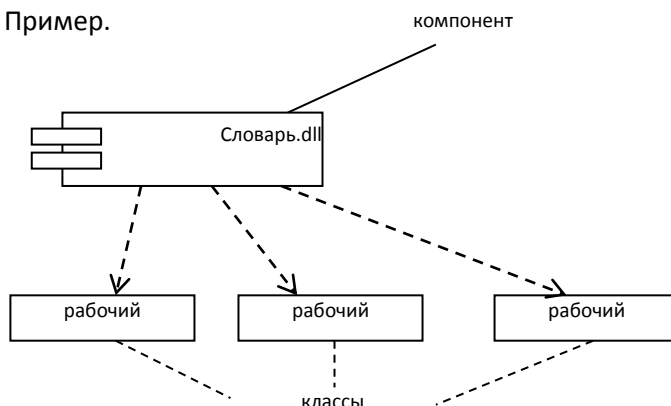
Сходные характеристики компонента и класса.

1. Наличие имени
2. Реализация набора интерфейсов
3. Участие в отношениях зависимости
4. Возможность быть вложенным (для классов – это наследники)
5. Наличие экземпляров.

Различия компонентов и классов

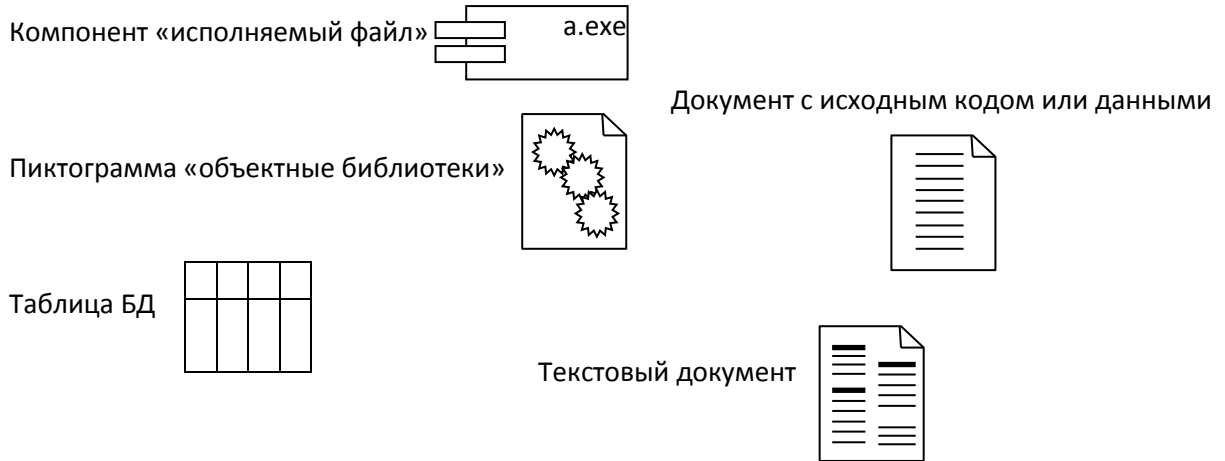
1. Классы – это логические абстракции
Компоненты – физические предметы, которые живут в мире битов. Т.е. физически располагаются на физических узлах (на серверах, рабочих классах, узлах).
2. Компоненты являются физическими упаковками (контейнерами), в которых инкапсулируются различные логические элементы.
3. Классы имеют свойства и операции.
Компоненты имеют только операции, доступные только через их интерфейс.

Пример.

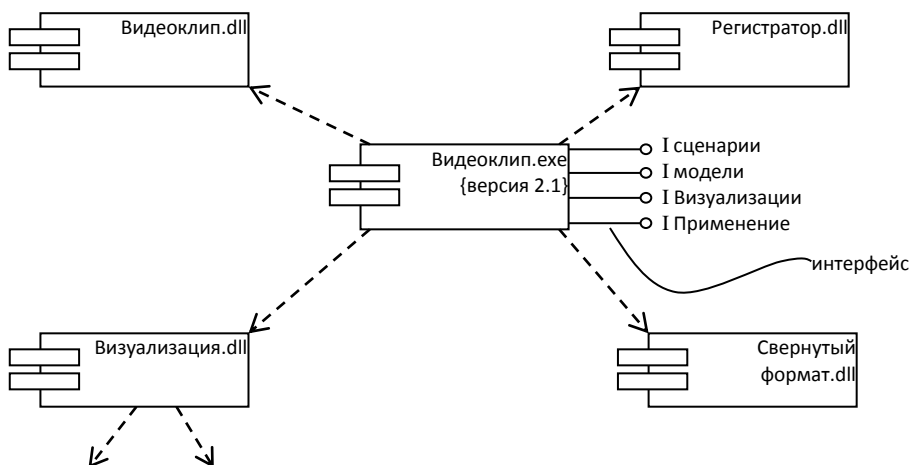


Класс – это некий такой «хрупкий» логический элемент, которому нужен «скафандр» для существования в физическом мире. Этим скафандром является компонент.

Разновидности компонент и их графическое изображение в виде пиктограмм.



Пример.



Компонентная диаграмма

Техническое узлы – это серверы. Изображаются в виде параллелипипеда с затемненными гранями (левым и верхним). На верхней грани пишется: «Сервер, частота столько-то гХЗ». На передней рисуются и пишутся все связи, схематически изображаются диаграммки и т.д.

От этого куба идет ссылка на такой же кубик «клиент». Рисуем «человечков»: «админ», «опер»... оператор то есть.

12.04.2011

Computer-Aided Software/System Engineering (CASE)

Достоинства и преимущества case-систем.

1. Единый графический язык. Это значит, что в графическом виде на одном из стандартов все рисуется.
2. Единая база данных проекта (репозиторий). Репозиторий может сохранять свыше 100 типов проекта, это описание данных, модели данных, исходные коды, меню. Т.е. все информационные объектов проекта.
3. Интеграция средств. Могут интегрироваться разные инструменты – БД, программные пакеты и т.д. И на единой платформе они могут взаимодействовать.
4. Поддержка коллективной разработки и управления проектами. Case-технологии поддерживает групповую работу, работу сети, обмен информацией, контроль работы каждого из участников, осуществлять контроль за работой.
5. Макетирование. Case-система дает возможность быстро построить макеты для будущей системы, показать заказчику, оценить и осуществить необходимые изменения.
6. Генерация документации. Автоматически генерируется документация в нужных стандартах, отражающая ход разработки на данный момент.
7. Верификация проекта. Case-технология обеспечивает автономную верификации проекта на ранних стадиях разработки. На этапе проектирования найти ошибку – это в 100 раз дешевле, чем при внедрении.
8. Автоматическая генерация объектного кода. Case-системы делают это автоматически в различные языки программирования. Но на 85-90%. 100% невозможно.
9. Сопровождение и реинжиниринг. Это значит, что после построения проекта в коллективе, после проверки проекта я нашел ошибки. И тогда после изменения кода делается реинжиниринг, т.е. автоматически применяются все изменения в проекте (переделываются диаграммки и т.п.).

Какими критериями следует руководствоваться при выборе CASE-системы? С первого-то года никакая система не дает никаких положительных эффектов.

1. Выбор зависит от класса решаемых задач. Системы реального времени, коммерческие системы, системы связанные с САПР и т.д. – это все разные классы.
2. Поддержка полного жизненного цикла система. Поддерживает ли case-система полный жизненный цикл и ее модификацию? Есть case-системы, которые поддерживают только один уровень (проектирование, кодирование, тестирование, реинжиниринг). Каждый из перечисленных блоков стоит отдельных денег. Если предстоит крупную систему (и не одну), то стоит закупить систему с полным жизненным циклом.
3. Обеспечение целостности проекта и контроля за его состоянием. Поддерживает ли версиюность, верификации.
4. Независимость от программно-аппаратной платформы и выбранных СУБД.
5. Открытая архитектура. Если case-система обеспечена этим, то это значит, можно подключать другие компиляторы, подключение новых СУБД и т.д.
6. Качество технической поддержке в той стране, где идет работа, стоимость приобретения и поддержки новых версий, опыт успешного использования этой case-системы в данной стране.
7. Простота освоения и использования. Если ее освоить трудно, а квалифицированных кадров немного – то стоит подумать над покупкой более простой.

Пилотный проект – это проект на маленьком релизе для того, чтобы попробовать, как подходит эта система для реализации конкретных поставленных задач. Как правило, большие проекты (IBM и подобные), пробные релизы предоставляют бесплатно.

Критерии классификации CASE-систем.

- По категориям – по следующим признакам:
 - Интегрируемость – оценивается степень интегрированности case-системы с другими средами (средствами) разработки: СУБД, средами визуального проектирования и кодирования, операционными системами.
 - Применяемые методологии организации данных (реляционные, объектно ориентированные модели)
 - Доступным платформам – имеется в виду и техническим в т.ч.
- По типам – она совпадает с компонентным составом case-систем.

По типам:

1. Upper CASE – это case-системы верхнего уровня, средства анализа предметной области. К ним относятся Design/IDEF0, BPWin (бизнес-процесс под windows – позволяет спроектировать бизнес процесс на самом верхнем уровне). Эти системы используются для не очень больших задач.
 2. Middle CASE - case-системы среднего уровня. Включают средства анализа и проектирования. К ним относятся
 - 2.1 CASE-Аналитик – позволяла в нотации Буча или Рамбо осуществить проектирование системы
 - 2.2 Design/2000
 - 2.3 Silveran
 - 2.4 PRO-IV
 3. Case-системы, которые позволяют спроектировать схемы БД и сгенерировать сами БД (SQL). К ним относятся: ERWin, S-Designer (oracle)
 4. CASE-системы, включающие средства разработки приложений. Это языки четвертого поколения (4GL). 4GL – это среды визуального программирования. Это Delphi и прочие.
- <<часть (на 10 минут лекций) пропустил. Следует восстановить>>

Тестирование ПО

См. методичку «тестирование ПО».

Тестирование было искусством, каждый тестировал так, как ему казалось правильно. Сценарии тестирования были засекречены. Но, как это всегда бывает, времена меняются.

Лабораторную следует делать в среде автоматического тестирования.

Карл Поупер – философ: «что нужно сделать, чтобы проверить, что разрабатываемая теория верна». Все думали, что следует искать факты, подтверждающие теорию, а он искал факты, опровергающие.

Теоретически доказано, что невозможно оттестировать до конца ни одну, даже маленькую, программу. Можно говорить только о степени.

Майрс – первый теоретик, опубликовавший книгу «теория тестирования». Написал программу из 10 строк со 150 ошибками.

| «Формула оценки степени тестированности программы» - любимый вопрос.

| Ответ есть в методичке.

Этапы тестирования.

Они совпадают с этапами разработки:

1. Анализ предметной области, стадия планирования, создание спецификаций. На этом этапе планируется проект, выставляются предположительные высокоуровневые требования. После этого вся информация передается независимому тестировщику. Он оценивает риски (финансовые, технические, по квалификации сотрудников), реальность сроков, адекватность требований, совместимые требования. Выполнимые ли данные требования на данной технической платформе. Разумны ли они. Здесь могут приняты решения либо о модернизации, либо об ослаблении требований. Поддаются ли выставленные требования тестированию? Т.е. на этапе проектирования уже задаются методы и средства тестирования: что именно в первую очередь можно тестировать?
2. Стадия проектирования. Здесь идет разработка проекта, более формально описываются все спецификации. Проектируется дизайн пользовательского интерфейса и внутренняя структура программных модулей. Проектируется организация данных и тестируется выбор моделей организации данных. Технологии доступа к данным, модели организации данных. Оценивается правильность выбора технологий. Оценивается и тестируется возможность восстановления базы данных – будут использоваться ЦОДы, зеркальные сервера. Оценивается, полон ли проект – описываются все взаимосвязи, описываются степени реализации каждого модуля.
3. Тестирование на стадии кодирования. Тестировщики работают параллельно с кодировщиками. Сначала модуль тестирует сам разработчик. Используются методы визуальной инспекции за круглым столом. Используется белый (прозрачный) ящик. Тестируются там все условные переходы. Тестирование черного ящика (тестирование функциональности): задаются входные в него потоки, получаем выходные и сравниваем с эталоном, просчитанным теоретически или вычисленным другими ПО. Часто черный работает совместно с белым, особенно если есть разветвленный интерфейс на входе. Используют метод функциональных диаграмм.

Класс эквивалентности – это множество входных значений, каждое из которых имеет одинаковую вероятность обнаружения конкретного типа ошибок.

Как определяют класс эквивалентности? Оценивают правильные исходные данные и «неправильные». Правильные – это те, которые входят в область допустимых значений: «введите значение от 0 до 10».

Класс граничных значений – всегда проверяется: до границы, на границе и после границы. Наибольшее количество ошибок сходится именно вблизи границы. **Это метод граничных значений.**

Метод предположений – используется интуиция программиста. Надо прикинуть, где возможна ошибка (болевая точка) в данном коде. Например, идет удаленный вызов. Надо проверить, что будет в случае обрыва канала связи.

Нагрузочное тестирование.

Регрессионное тестирование – выборочное перетестирование ПО или компонента с целью обнаружения, не появились ли новые ошибки при изменении кода.

Тестирование на дым – «огня нет, а дым остался», то есть след ошибки. Искали и нашли саму ошибку. А теперь остался «дым»: какая-то всплывающая подсказка осталась той, какая была до исправления. Ее надо подкорректировать.

Нагрузочное и предельное тестирование

Нагрузочное тестирование:

Например, идет обработка большого объема информация. При тестировании мы даем не только этот объем, но и сверх этой нагрузки. Вместо 3 файлов, даем 4 или 5. Обработает медленнее? Или не обработает вообще из-за каких-то переполнений в буфере или из-за исчерпания системных ресурсов (ОЗУ)?

Если объем оперативной памяти превышает или меньше того, при котором разработка была. Превышаем объем аппаратных ресурсов (напр, объем данных на жестком диске, ОЗУ, обрабатываемых данных), т.е. буфер, массив и т.п.

Предельное тестирование:

Это оценка возможности системы при работе на пределах возможности, проверка выполнимости времени реакции системы.

Пример:

Приходит секретарша устраиваться на работу. Даю работу: напечатать такой-то объем информации. Она справляется за 2 дня. Проверяю на предмет правильности – это нагрузочное.

А если дам тот же объем работы на 2 часа – это предельное тестирование.

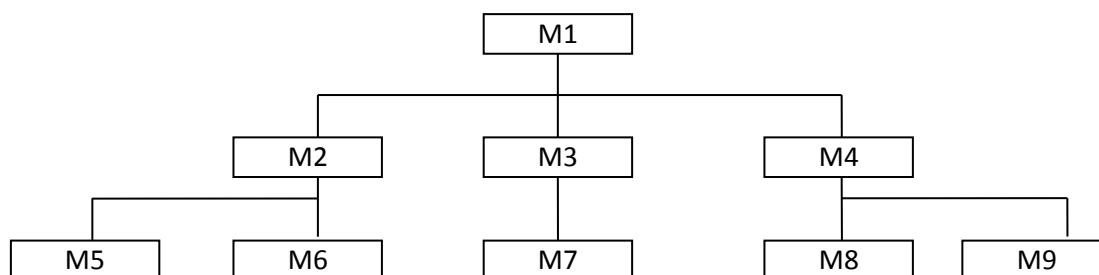
Есть еще один момент.

Поскольку тестов много, они сложны, нет эталона (каждая система уникальна, свои задачи, свои особенности, свои подходы), то я как автор все равно могу немного автоматизировать. Например, на вход дается файл, на выходе получаю файл. Могу написать тест-программу, сравнивающую эти файлы.

Или, например, программа обращает матрицу. Как проверить правильность? Можно взять исходную матрицу, умножить на получившуюся – должна получиться единичная. Т.е. такие подбные приемы, позволяющие автоматизировать тестирование, желательно использовать. Генератор случайных чисел – следует проверить, дает ли он повторения в заданных рамках? Например, часто выдает повторения, которые не могут иметь место в реальной жизни.

Интеграционное тестирование при структурном подходе к программированию.

Структура программы:



Если нижние модули отвечают за обработку и ввод информации, то нужно начинать снизу вверх. Если не отвечают – то можно попробовать обрабатывать сверху вниз. Поэтому сначала тестируем сверху вниз: сначала одну ветку: M1->M2->M5, а на остальные делаем заглушки.

Драйвер (в контексте тестирования) – модуль, обеспечивающий вызов и передачу тестируемому модулю данных и необходимых результатов.

Заглушка – модуль, имитирующий функции модулей, вызываемых при тестировании.

Пошаговое тестирование – последовательно добавляем в систему по модулю и каждый раз тестируем.

Сборка при тестировании:

- Сверху вниз
- Снизу вверх
- Пошаговая
- Монолитная – тестирование целиком системы.

Процесс тестирования – процесс многократного повторения кода программы в различных условиях для обнаружения как можно большего количества ошибок.

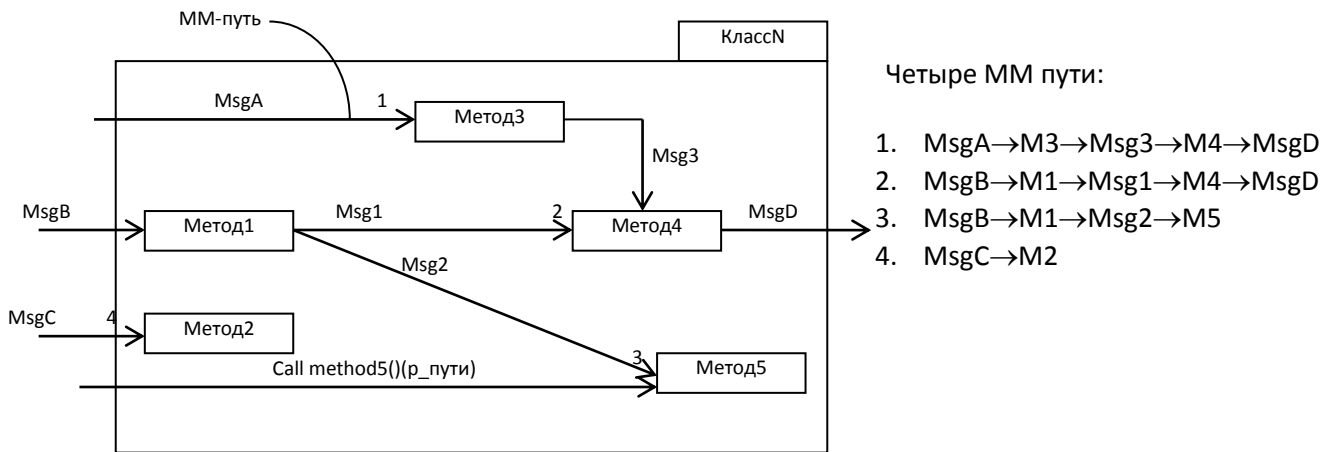
Тестирование при объектно-ориентированном подходе.

Построение графовой модели программы для начала тестирования.

При объектном подходе «модульное тестирование» означает тестирование класса.

Графовая модель программы – ГМП.

- Вершины графа – методы
- Ребра графа – вызовы методов



Число ММ-путей зависит от схемы обработки сообщений данным классом, что должно быть определено в спецификации класса.

Сложность тестирования интеграционного класса.

Общая формула.

Сложность тестирования по критерию С – это функция от количества точек входа и количества коэффициентов:

$$V(cls, C) = f(K_{msg}, K_{em})$$

K_{msg} – число точек входа в класс, которые могут быть вызваны извне.

K_{em} – число определяется как сумма методов, которые могут быть вызваны из других классов.

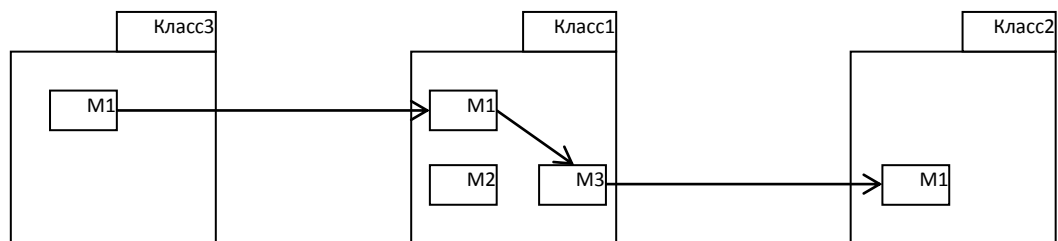
$V(классN, C1) = 5$ – если call method5() - public

$V(классN, C1) = 4$ – если call method5() - private

Итак, каждый класс тестируем отдельно. Оцениваем. Теперь строим интеграционную модель всех классов, которые могут между собой все взаимодействовать.

Дерево классов проекта.





Методика тестирования классовой модели в несколько этапов.

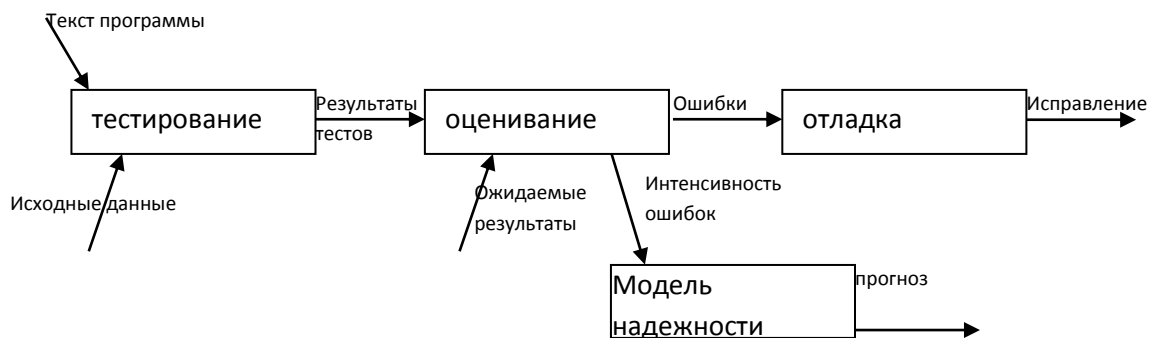
1. Сначала тестируется каждый метод в каждом классе
2. Объединяем их в графовую модель программы
3. Интеграционное тестирование как дерево классов, отслеживаются реакции программы на внешние события

19.04.2011

Структурное тестирование программного обеспечения

Всегда надо описывать цель тестирования, аппаратура, ПО и т.п. и только после этого следует приводить сами тесты.

Тестирование – это процесс выполнения программы с целью обнаружения ошибок. Тестирование считается успешным, если было найдено много ошибок.



Что дает тестирование?

1. Обнаружение ошибок.
2. Демонстрация соответствия функций программы ее назначению.
3. Демонстрацию реализации требований (записанных в ТЗ) к характеристикам программы. Например, заявлено время реактивности – 3 секунды.
4. Отображение надежности как индикатор качества разработанного ПО.

Тестирование не может продемонстрировать отсутствие ошибок. Эксплуатируют неделю программу на полной нагрузке, вывелось 2 сбоя – вот и вероятность.

Цели:

1. Проверить, на сколько заявленные функции заявленным требованиям.
2. В режиме нагрузочного и предельного тестирования цель тестирования: определить реактивность системы. Напр., в ТЗ заявлена реактивность 3 секунды. Цель – определить, так ли это. «Нагрузочные и предельное тестирование показало, что система укладывается в заданные временные рамки».

Программное и техническое кружение в режиме тестирования. Это ОС, библиотеки, технические платформы и т.п. – все, что используется при тестировании. Это обязательно, т.к. тестируется не только программа, но и оценивается возможность перехода из одной ОС в другую.

Способ тестирования базового пути.

Все это в рамках тестирования белого ящика. Мы говорим о структуре программы. Этот способ предложил Том МакКейб в 1976 году.

Способ тестирования базового пути дает возможность:

1. Получить комплексную оценку сложности программы
2. Использовать эту оценку для определения необходимого количества тестовых вариантов.

Потоковый граф

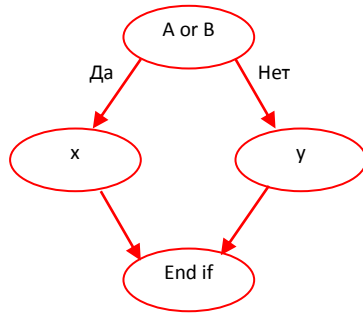
В качестве узлов графа будем использовать один или несколько линейных операторов программы. Узел или вершина – это линейные участки программы.

Дуги отображают поток управления в программе. Дуги ориентированы.

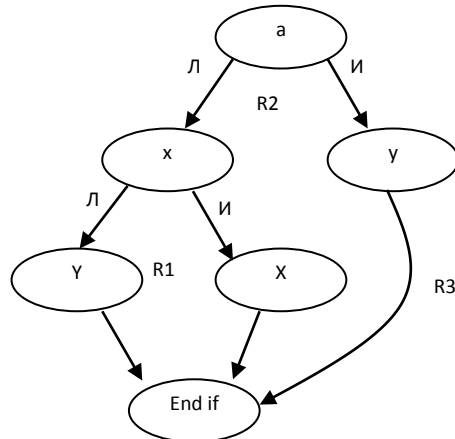
Узлы бывают операторные. В качестве отдельного оператора используются скобки. Концы цикла (end loop) и конец условия (end if) нумеруются как отдельные операторы.

If a or b then X else Y end if

НЕПРАВИЛЬНО:



Правильно:

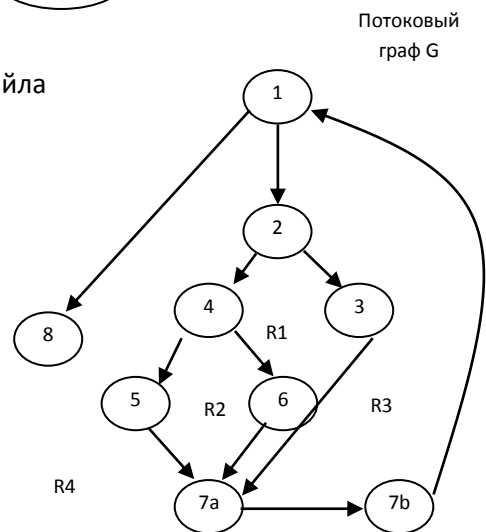


R1,R2,R3 – регионы. R3 – глобальный.

Процедура сжатия файла:

1. Выполнять пока нет OEF, читать запись из файла
2. Если запись не пуста
3. То запись удалить
4. Иначе если поле a>поле b
5. То удалить поле b
6. Иначе удалить поле a.
7.
 - a. End if, End if
 - b. Конец выполнить
8. Конец сжатия

Потоковый граф для процедуры сжатия.



Цикломатическая сложность – количественная метрика, обеспечивающая логическую метрику сложности программы. Она определяет:

1. количество независимых путей в базовом множестве
2. Верхнюю оценку количества тестов, которая гарантирует однократное выполнение всех операторов. Критерий C0 – самый мощный критерий, когда каждый оператор выполним только один раз. Для мощных программ это недостижимо.

Независимые пути в потоковом графе G:

Путь1: 1—8

Путь2: 1—2—3—7a—7b—1—8

Путь3: 1—2—4—5—7a—7b—1—8

Путь4: 1—2—4—6—7a—7b—1—8

Независимым называется любой путь на графе G, который вводит новый оператор обработки или новое условие. Т.е. не проходим по одной и той же ветке 2 раза.

Как определить цикломатическую сложность?

1. $V(G) = E - N + 2;$

E – количество дуг

N – количество узлов (вершин).

В графе G: $V(G) = E - N + 2 = 11 - 9 + 2 = 4$

2. $V(G) = p + 1$

P – количество предикатных узлов.

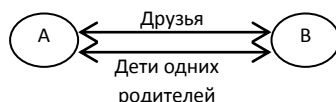
Графы и отношения

Графы – основополагающий инструмент в тестировании. В тестировании используются различные модели на основе графов:

1. Граф потоков управления
2. Граф потоков данных
3. Дерево вызовов
4. Граф конечного автомата
5. Граф потока транзакций.

Основные нотации при создании графов.

Граф представляется в ООП как набор объектов и отношений между ними. Как обозначаются отношения между ними?: $A \checkmark B$, где \checkmark – отношение.



Отношения:

1. Объект «А» вызывает объект «В»
2. Объект «А»
3. Данные объекта «А» используются для вычисления объекта «В»
4. Объект «Б» включается в объект «А»
5. Симметричные (А является братом Б, Б является братом А)
6. Несимметричные (А является другом Б, а Б не считает А своим другом)

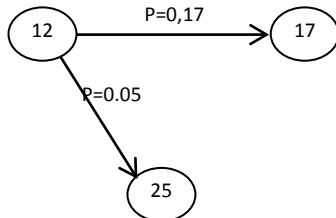


- объект в графе.

Вес узла. Узлы могут иметь свойства. Эти свойства называются весом узла. Задача тестирования – убедиться, что все узлы с заданным весом имеют действительно такой вес.

Связь – это «трелочка» (или линия – в случае двунаправленной стрелки), показывающая отношения между объектами. На этой стрелке пишутся отношения.

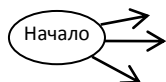
Вес связи – связи могут обладать свойствами, которые и называются «весом».



- На графе обозначена вероятность перехода из 12-й вершины в 17-ю и в 25-ю.



- входящая связь.

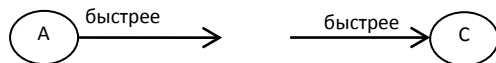


- Входной узел: ни одной входящей связи.

У выходного нет выходящих, только входящие.

Отношения (симметричность, транзитивность)

Отношения \checkmark транзитивно, если из $A \checkmark B$ & $B \checkmark C \Rightarrow A \checkmark C$. Графически:



Симметричность – очень важное свойство отношений, его следует тестировать. Какие отношения могут быть несимметричными? Отправил на печать – принтер взялся за бумагу, все, отменить назад нельзя. UNDO – симметричная операция.

Тестирование циклов

1. Сначала проходим цикл проходим заданное количество раз (для FOR).
2. Попытаемся пройти цикл N+1 раз.
3. Попытаемся пройти цикл N-1 раз.
4. Пройдем цикл 1 раз.
5. Пройдем цикл 2 раза.
6. Зададим индекс равным нулю.

Неструктурированные циклы (goto) не тестируются, а сразу перепроектируются.

Формирования очереди:

1. FIFO
2. LIFO
3. Пакет. Создается пакет и целиком запускается на выполнение.

26.04.2011

Тестирование очередей и потоков данных.

Типы очередей – FIFO, LIFO.

- Пакетная обработка – когда количество транзакций – конечное число. Напр., очередь накопилась до определенного числа – и она поступает на обработку. Пакет – очередь, состоящая из определенного конечного числа транзакций.
- Случайное обслуживание – осуществляется случайным образом (обычно на основе вероятностной величины).
- Очередь по приоритету. Каждая транзакция имеет свой приоритет, определяемый по определенному правилу. Бывает одноприоритетная очередь. Очередь по приоритету – это когда имеется одна очередь, а внутри этой очереди имеются несколько очередей.
- Множественная обработка – это когда множество очередей, и имеется свое правило, какую очередь выбрать (на курском вокзале я покупаю билет в той кассе, где народу меньше, а в кассах аэрофлота я стою только в той очереди, где продается билет на нужный мне рейс).

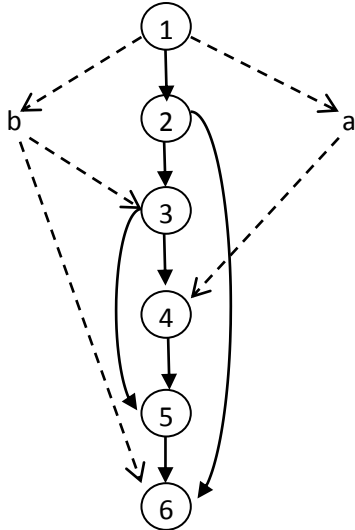
Как тестировать очереди.

Несколько общих рекомендаций, которые дает Бейзер в тестировании черного ящика:

- Тестирование пределов длины очереди. Т.е. пробуем превысить максимально заданное количество.
- Тестирование пустой очереди. Активация обработки, когда в очереди нет ничего.
- Проверка циклов. Очередь образуется, возможно, в каком-то цикле. Тогда нужно протестировать этот цикл по правилам тестирования циклов (см. предыдущую лекцию).
- Динамическое изменение длины очереди.
- Тестирование сортировки и выбора. В FIFO и LIFO идет либо прямая либо обратная сортировка соответственно. Поэтому здесь следует тестировать правильность сортировки, чтобы очередь работала как надо. Если идет сортировка на основе ключа, тогда надо еще посмотреть, как пройдет сортировка в случае повторяющихся ключей.
- Тестирование очереди с множественными приоритетами. Тестируют каждую очередь, а затем более высокий уровень приоритетов – между очередями (см. пример с аэрофлотом выше).

Тестирование потоков данных.

Построим граф потока данных (граф потока данных). На основе этого графа тестируем.



$DEF(i) = \{x \mid i\text{-я вершина определяет переменную } x\}$

множество определений данных

$USE(i) = \{x \mid i\text{-ая вершина использует } x\}$

Множество использований данных

DU-цепочка $[x, l, j]$

DU-цепочки: $[a, 1, 4]$, $[b, 1, 3]$, $[b, 1, 6]$, $[c, 4, 6]$.

Шаги способа DU-тестирования.

1. Построение управляющего графа программы.
2. Построение информационного графа программы.
3. Формируем DU-цепочки и записываем все переменные.
4. Построение маршрутов, которые следует протестировать.
5. Подготовка тестов.

Достоинства метода:

1. простота формирования DU-цепочек.
2. Простота автоматизации. Граф строится автоматически.

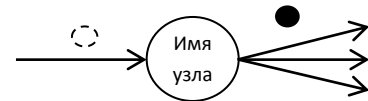
Недостаток метода:

1. Трудно определить оптимальное число независимых путей. Дело в том, что число независимых путей тяжело определить.

Тестирование потока транзакций.

Транзакция – единичная (атомарная) операция по обработке данных.

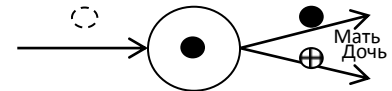
Узел ветвления – когда входящая транзакция выбирает один из возможных путей, по которому она пойдет дальше. Предикат обозначается пунктирным кругом.



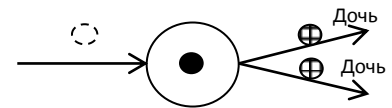
Узел соединения – транзакция поступает по любой из входящих веток и выходит по одной. Пунктирной окружностью обозначается входящая транзакция.



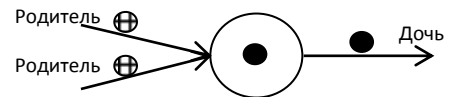
Узел порождения – когда транзакция порождает две другие – материнскую и дочернюю.



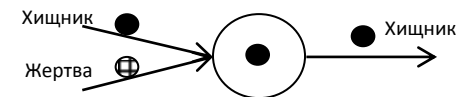
Узел расщепления. Входит по одной – выходит две дочерние. Фактически, копирование. НАпр, надо выдать два одинаковых бланков.



Узел слияния – когда две родительские транзакции порождают одну дочернюю.



Узел поглощения. Транзакция поглощает другую.



Марковский узел – узел, действие которого зависит от типа и состояния входящих транзакций. Т.е. фактически, в одном случае она может быть «слиянием», другом «поглощением». Зависит от типа входящих транзакций, но не от пути, по которому транзакция добралась.

Тестируем декларацию.

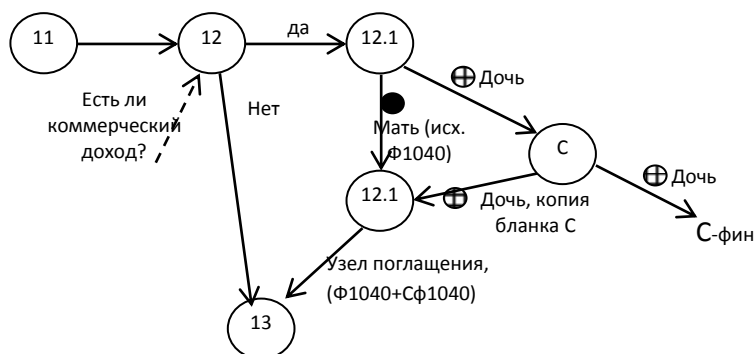
(С – это «бланк Цэ»)

- 11: 12: Предыдущий шаг в модели. (т.е. из 11 узла (пункта) выходит, в 12 входит, далее комментарий)
- 12: 13: Нет {Узел ветвления (имеется ли коммерческий доход?) – управляющий предикат
- 12.1: Да
- 12.1: 12.2: Узел порождения ф1040 продолжается
- С: дочерняя транзакция для обработки бланка С
- С: 12.2: Заполнение бланка С (узел расщепления)
- С-фин: Заполнение бланка С для финансовых структур.
- 12.2: 13: Узел поглощения, данные бланка С поглощаются формой 1040.
- 13: Узел соединения.

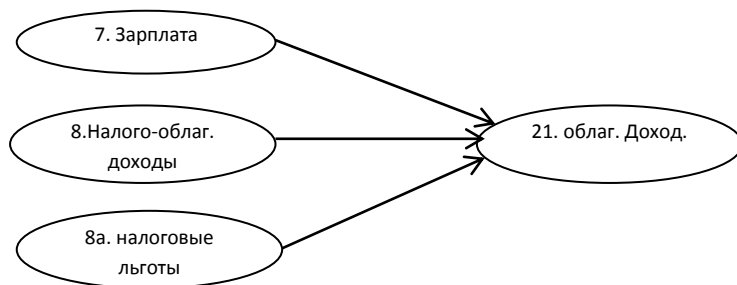
В общем, если нет коммерческого дохода, то переходим в 13-й пункт. А вот если есть, то создаются отдельные бланки – Ф1040, который в дальнейшем уходит в финансовые органы, а второй С – он добавляется в основной бланк (т.е. поглощается основным бланком).

Теперь строим граф.

Граф потока транзакций для заполнения деклараций формы 1040.



Еще пример:



Международные стандарты на разработку ПО

Стандарты, регламентирующие документирование программных средств и баз данных.

wwwcdl.bmstu.ru – сайт кафедры РК-6. В разделе «дистанционное облучение» и «учебные-методические материалы». Авторы: Волосатова Тамара Михайловна (заместитель зав.каф. РК-6), Родионов С.В., Романова Т.Н. (Адрес сайта написан без опечаток).

«Требования к документации: выдать пользователю все UML-диаграммы, выдать все, что угодно, кроме того, что надо» - неправильное. Эта лекция посвящена тому, чего давать, а чего нет.

По своему назначению и ориентации на определенные задачи и группы пользователей, техническую документацию делят на три группы:

- Технологическая документация. Передается в том случае, если мы коллективно разрабатывали какую-то систему, а затем мы передаем на доработку другой команде разработчиков. В таком случае передается вообще все. Тех.док. должна содержать информация:
 - должна отражать все разрабатываемые объекты и процессы жизненного цикла прикладных программ и данных.
 - В комплекте должны содержаться профиль стандартов (набор стандартов, на основе которых велась разработка), регламентирующий процесс разработки и качества,
 - функциональные требования к системе и подсистемам.
 - реальные значения достигнутых показателей качества и всю подробную информацию о проекте, который был реализован.

Все это нужно для того, чтобы группа разработчиков смогла продолжить работу над проектом. В Тех.док. фиксируются исходные данные, входные и выходные данные, процедура контроля проекта (т.е. набор тестов, которые следует запускать перед запуском системы).

- Эксплуатационная документация (ниже)
- Исследовательская документация (ниже)

Профиль стандартов документирования объектов

Это группа стандартов, которая должна определять:

1. Логическую структуру (по стандарту еспд) программных и информационных компонентов и баз данных компонента.
2. Профиль стандартов должен содержать спецификации (т.е. подробные описания) на внутренние межмодульные интерфейсы между компонентами системы и интерфейсы с внешней средой.

3. Язык и правило программирования. Обосновываем выбор языка, описание используемых библиотек.
4. Комментарии текстов программ, описание данных.
5. Структуру и содержание исходных и отчетных документов по этапам разработки (проектирования) и тестирование, которое было проведено в рамках создания. SNN.
6. Критерии оценки качества – это отвечает заявленным в ТЗ характеристикам, которые измеряются секундомером, эти характеристики должны вписываться в стандарты качества.

Стандарт SMM (уровни):

1. Команда работает впервые. Самый нижний уровень
2. ..
3. ..
4. ..
5. Сертификаты IBM, Microsoft и т.п.

Эксплуатационная документация

Напр., требуется в аутсорсинговую компанию переслать.

Эксплуатационная докуменатция должна обеспечить отчужденность программных систем от первичных разработчиков и возможность эффективного применения достаточно квалифицированными специалистами на этапе ее сопровождения.

Не следует передавать слишком много информации. Ее пользователю и передаем.

Эксплуатационная документация должна содержать следующие элементы:

1. Краткую (однозначная!) аннотацию и описание решаемых задач и алгоритмов.
2. Описание структуры и функции программы, иерархии модулей программы и межмодульных интерфейсов.
3. Описание пользовательских (внешних) интерфейсов.
4. Описание входных и выходных данных. Если входных таблиц много, то идет все в приложении, а в ТЗ ссылка на них.
5. Указание конкретной среды функционирования программной системы, способов проверки работоспособности программы и контрольные задачи, тесты.

Исследовательская документация

Она имеет экспериментальный характер, зависящий от возможных целей исследования.

Мы решаем систему диф.ур. ДисТокса. Система слишком сложна для аналитического решения.

Промоделировали физический процесс (ядерную реакцию, запуск самолета или ракеты). Протестировали по независимым тестам независимые тестировщики. Затем идет исследование процесса на основе программы.

В исследовательской документации следует указать:

- Физический процесс.
- Математическая постановка задачи.
- Тестирование

В общем, это все, что касается тестирования и моделей: информационной, и еще двух каких-то.

Пользовательская документация

В дипломе должна быть: инструкция для администратора и инструкция для пользователя. Там пишут:

- Как устанавливать
- Как работать
- Особые моменты (рестарт, устранение неполадок)
- Ограничения, которые следует иметь в виду.

Номера стандартов, которые регламентируют Это:

1. ISO 6592:1985.ОИ.Руководство по документации для вычислительных систем.
2. ISO 9294:1990-ТО.ИТ.Руководство по управлению документированием ПО.
3. ISO 9127:1988.СОИ.Пользовательская документация на ПС.

Книга Липаева... это один из ведущих людей, который принимает участие в разработке ГОСТов. Дело в том, что международные ГОСТы не согласуются с российскими. Выписки из его книги:

1. ISO 12207:1995 (ГостР-1999). ИТ.Процессы жизненного цикла программных средств. Этот стандарт регламентирует жизненные циклы, которые мы знаем.
2. ГОСТ 51904 – 2002. Программное обеспечение встроенных систем. Общие требования к разработке и документированию.
3. ISO 9126:1991 (ГостР – 1993).ИТ.Оценка программного продукта (не системы, не вычислительной системы, не информационной и т.п., а именно *продукта*). Характеристики качества и руководство по их применению.
У этого стандарта есть несколько частей.
ISO 9126-1–4:1991.ИТ.Качество ПС.
Часть1. Модель качества
Часть2. Внешние метрики (которые оценивают качество)
Часть3. Внутренние метрики
Часть4. Метрики качества в использовании применительно к системам.
4. ISO 14102:1995. Оценка и выбор CASE-средств.
5. ISO 14471:1995. Руководство по адаптации CASE-средств.
6. ISO 15910:1999. (ГостР – 2002) ИТ. Пользовательская документация.

Эти вещи она обещала спросить на экзамене. Еще надо знать, чем надежность от отагоустойчивости отличается (это см.след.лекцию).

Стандарты, регламентирующие сопровождение и управление конфигурацией сложных программных средств.

При сопровождении мы должны фиксировать:

- Анализ и фиксация дефектов.

Реализация модификаций системы состоит из следующих задач:

- Персонал сопровождения должен проводить анализ и определять, какие модули программного средства должны измениться, какая документация должна измениться, оценить время, которое потребуется на эту модернизацию, и все это задокументировать.
- Персонал должен записать критерии оценки при тестировании и оценить
- Перенос на иную платформу. Должен быть разработан миграционный план. Документирован и выполнен. В нем нужно указать участие разработчиков
- Процесс управления конфигурацией:
 - Составляется план управления конфигурацией (в какие сроки, какую версию, что эта версия делать будет)
 - Идентификация и запись запросов на изменение (Т.е. что требовалось в новой версии изменить, анализ оценки изменений), а также оценка изменений.
 - Ликвидация дефектов в программе.

Лекция 10.05.2011

Первую часть лекции дописать.

Основные качественные характеристики программных средств и их атрибутов.

| Характеристики | Мера | Школа | Приоритет (1-10) |
|---|--------------------------------------|---|------------------|
| Практичность 1. Понятность: - четкость концепции программных средств - демонстрационные возможности - наглядность и полнота документации | Порядковая | Отл. Хор. Удовл. Неуд. | 1-10 |
| 2. Полнота использования - простота управления функциями программной системы - комфортность эксплуатации - среднее время ввода заданий - среднее время отклика на задание | Порядковая Сек. Сек | Отл. Хор. Удовл. Неуд. 1-1000 1-1000 | 1-10 |
| 3. Изучаемость - трудоемкость освоения ПС - продолжительность изучения - объем эксплуатационной документации - объем электронных учебников | Человеко-часы Час Стр Кбайт | 1-100 1-100 10-1000 100-10 000 | |
| 4. Привлекательность - субъективные или экспертные оценки. | | | |
| Сопровождаемость 1. Анализируемость - стройность архитектуры программного средства - унифицированность интерфейса - полнота и корректность документации | Порядковая | Отл., хор., уд., неуд. | |
| 2. Изменяемость - трудоемкость подготовки изменений - длительность подготовки изменений | Человеко-часы, Часы | 1-1000 1-1000 | |
| 3. Стабильность - устойчивость при негативных проявлениях при изменениях | Порядковая | Отл,ХУНеуд | |
| 4. Тестируемость - трудоемкость тестирования изменений на этапе сопровождения - длительность тестирования изменения | Человеко-часы Часы | 1-1000 1-100 | |
| Мобильность 1. Адаптируемость - трудоемкость адаптирования - длительность адаптации | Чел-час Час | 1-100 1-100 | |
| 2. Простота установки: - трудоемкость установки - длительность установки | Чел-час Час | 1-100 1-100 | |
| 3. Замещаемость - трудоемкость установки | Чел-час | 1-100 | |

Пример требований к количественным характеристикам качества программного средства.

Надежность.

Сколько нужно наработать на отказ системе, чтобы она считалась надежной.

- завершенность – наработка на отказ при работе без рестарта.
- устойчивость – требуемое значение 50 часов.
- восстанавливаемость – длительность восстановления в минутах. Требуется 5 минут.
- доступность – вероятность доступности 0.998
- эффективность – не более 5 секунд.
- время отклика – время получения результата на получение типового задания – 5 секунд.
- пропускная способность – не менее 20 типовых заданий в минуту.
- используемость ресурсов – вероятность использования ресурсов ЭВМ – 0.8. Т.е. если пользователь работает с программой, которая использует ресурсов меньше 0.8, то программа не эффективна.

Практичность

- среднее время ввода заданий, среднее время отклика заданий. Не более 10 секунд.
- среднее время ввода заданий не более 10 секунд.

Изучаемость

- трудоемкость – не более 200 человеко-часов
- время изучения – не более 50 часов
- объем эксплуатационной документации 50

Сопровождаемость

- трудоемкость подготовки изменений - 10
- длительность подготовки изменений – не более 5

Тестируемость

- трудоемкость тестирования изменений – 20
- длительность тестирования изменений – не более 5.

Мобильность

- трудоемкость адаптации к новой платформе – не более 50 человеко-часов
- длительность адаптации – не более 10.

Инсталляция

- трудоемкость установки – не более 10 ч-часов.
- длительность установки – не более 5 часов

Замещаемость

- трудоемкость – 50
- длительность – 10.

Характеристики качества баз данных.

Стандарт ISO 9026 подробно описывает характеристики качества каждой из баз данных.

Базу данных можно рассматривать как два независимых компонента:

1. СУБД (для управления данными)
2. Сами данные, которые структурирую по определенным данным.

Поскольку задач много, данные разнообразны, поэтому всеми данными нельзя управлять только одной СУБД, поэтому появилось их столько. У них разные модели управления данными. Для каждой модели существует свой класс системы управления базы данных. Чаще всего используем реляционную модель.

Как тестировать базу данных? Выбрали Oracle. Протестированная... навороченная... все может. Построили свою БД. Кэшировали какие-то запросы и т.д. Приступаем к тестированию. Что следует тестировать?

1. хранимые процедуры отдельно, белым и черным ящиком.
2. взаимодействие – последовательность вызова процедур.
3. Если распределенная система, то тестируем время реактивности системы при самом удаленном и при самом близком вызове. Время должно в любом случае укладываться в рамки заявленных. Этот тест затрагивает и вопрос кэширования.
4. Тестирование в режиме сбоя. Описать все возможные сбойные ситуации, протестировать. Итак, полнота, непротиворечивость, кэшированные запросы, в режиме сбоя, актуальность измененной информации.

Жизненный цикл профилей стандартов

Жизненный цикл профилей стандартов систем и программных средств.

Жизненный цикл программного изделия – этапы, которые мы должны пройти, чтобы создать ПС.

Профиль стандартов конкретной системы не является статичным, он развивается и конкретизируется в процессе разработки. Конкретизируется, возможно, при взаимодействии с заказчиком.

Существует две группы профилей разрабатываемых программных систем:

1. функциональные профили
2. технологические профили, регламентирующие создание и сопровождение ПО,

Функциональные профили, в них входит:

1. профили жизненного цикла системы.
2. Профиль аппаратной и операционной среды – набор стандартов, которые будут регламентировать аппаратную и операционную среду. Речь идет о всех системах (включая саму ОС).
3. Профиль внешней и пользовательской среды – стандарты регламентирующие взаимоотношения с внешней средой. Речь идет об интероперабельности. Т.е. с чем система будет взаимодействовать и чьи обязуется не нарушить и т.п.
4. Профиль обеспечения безопасности и защиты (от сбоев) системы и информации (которая циркулирует в моей системе).
5. Профиль инструментальных средств, поддерживающих жизненный цикл программной системы.

Технологические профили – регламентируют создание, развитие и сопровождение программных средств и баз данных.

1. профиль жизненного цикла
2. стандарты (90126), обеспечивающие качества программного обеспечения и баз данных
3. стандарты, регламентирующие верификацию, тестирование и сертификацию программных средств и баз данных
4. стандарт, регламентирующий сопровождение и управление конфигурацией ПС и информацией в БД (если таковая есть).
5. Стандарт, регламентирующий документирование программных средств и баз данных.