***Младшие курсы!!!! Кто сейчас юзает этот конспект – скажите спасибо потому,***

***что только один человек удосужился его набрать, а прежде, чем набрать его надо***

***было написать (но так, как я его не писала!!!), то надо было отсканить))) В реале у***

***Галковского нет своего электронного конспекта так, что поток 75200Х просто***

***рулит, спасибо мне любимой))), только вот я его набирала полночи, а на следующий***

***день узнала, что Стержанов мне поставил автомат, вот облом)) Удачи, сдавать***

***будете долго и мучительно!!!***

**1)Цели и критерии проектирования:**

1. Удовлетворение функциональным требованиям.

2. Надежность

3. Удовлетворение явными – неявными требованиями по ресурсопотреблению

4. Дизайн (явн. – неявн.)

5. Цена

6. Требования к процессу разработки.

**Основные проблемы:** высокая стоимость и низкая надежность

*(Диаграмма стоимости, рисунок)*

*Сопровождение(67%), тестирование(15%), программирование(7%), проектирование(11%).*

Говорят, что ПО содержит ошибку, если его поведение не соответствует спецификации. В ПО ошибка,

если оно не выполняет того, что пользователю разумно от него ожидать. Отказ ПО – проявление в нем

ошибки. Любая программа и спецификация содержит всегда ошибки. Надежность ПО – вероятность

работы без отказа в течение определенного времени, рассчитана с учетом стоимости для пользователя

каждого отказа.

Технология проектирования – система принципов для создания надежного, экономичного, эффективного

ПО, работающего в разных сложных системах.

Различают методы, средства и процедуры для технологии проектирования ПО.

Методы обеспечивают:

1. Планирование и оценка проекта

2. Анализ системных и программных требований

3. Проектирование архитектуры и структуры программы

4. Решение задач кодирования, тестирования и сопровождения.

Средства:

1. Автоматизированная/автономная поддержка методов.

2. CASE – системы (Computer – Aided Software Engin)

Процедуры определяют порядок применения методов и утилит, формирование отчетов по определению

требований, контроль качества + координация изменений, оценка прогресса проекта. Процедуры

обеспечивают непрерывную технологическую цепочку разработки.

**2)Жизненный цикл 3)модели жизненного цикла**– это непрерывный процесс, который начинается моментом замысла и решения о

необходимости создания ПО, а заканчивается моментом изъятия из эксплуатации. Различают модели

жизненного цикла, т.е. структуры, определяющие последовательность и взаимосвязь процессов. Выбор

модели зависит от спецификации используемого ПО. Классическая модель - это спиральная модель.

Заключается в том, что: разбиение разработки на этапы. Они выполняются последовательно и с

документацией между этапами. Тут возможен откат.

1. Анализ

2. Проектирование

3. Кодирование

4. Тестирование

5. Сопровождение

1)Анализ требований и целей.

2)Описывается структура, архитектура ПО, структура модулей и алгоритмов.

3) и 4) Это выполнение программы для выявления ошибок.

5) Внесение изменений в эксплуатированное ПО.

Достоинства: документация.

Недостатки: туго отклонения от последовательности шагов, т.е. сильно зависит от точной

формулировки задачи (цели).

*(Диаграмма спиральной модели, рисунок)*

АНАЛИЗ

ПРОЕКТИРОВАНИЕ ОПРЕД,ТРЕБОВАНИЙ

РЕАЛИЗАЦИЯ И

ТЕСТИРОВАНИЕ ВЕРСИИ

ИНТЕГРАЦИЯ

Можно переходит на след. этапы, не дожидаясь завершения предыдущего.

Задача: как можно скорее предоставить работоспособный доступ.

Достоинства: более реально отражать разработку ПО, позволят переходить к след.этапу, не

дожидаясь завершения работы на текущем.

Недостатки: повышение требования к заказчику, трудности контроля и управления временем

разработки.

**Компонентно - ориентированная модель.**

В ней конкретизируется содержание этапа проектирования, т.к. разработка должна основываться на

уже имеющихся компонентах

*(Диаграмма спиральной модели, рисунок, выше, закрасить проектирование)*

Поиск нужного компонента

в базе библиотек

Не найдено

найдено

Разработка и

включение в базу

библиотек

в базу библиотек

Интеграция компоненты

Достоинства: уменьшение времени разработки до 30% и стоимости до 70%.

Недостатки: достоинства проявляются, когда сущ.ряд проектов для одной предметной области.

**Макетирование** (прототипирование), тоже, что и спиральная модель, но без тщательного

проектирования. Это позволяет быстро проектировать и , не сожалея, выбрасывать не правильное.

Подходит для не согласованных заказчиков и непонятливых разработчиков, т.к.обычно в худшем

случае приходиться переписывать 80 – 90%.

Создается макет (прототип) проекта для снятия неопр- ти требования заказчика. Макет может

быть на бумаге в виде спецификаций, может быть упрощенный работающий макет, тут иметься

два вида: 1) «на выброс», 2) «эволюционное макетирование» - это когда макет создается на основе

уже сущ-го макета, путем внесения ряда изменений:

Сбор и уточнение требований

Быстрое проектирование

Построение макета

Оценка макета заказчиком

Уточнение требований

Нет….

Недостатки:

1) Макет может быть принят заказчиком за продукт

2) Макет может быть принят разработчиком за продукт

3) У заказчика может сложиться неверное представление о сроках разработки программы.

4) Разработчик может идти на компромисс, используя не подходящие структуры данных и

компоненты, забывая, почему они не подходят.

**4)понятие Сложности ПО.**

Низкая надежность – во многом следствие высокой сложности ПО. Сложность определяется

количеством внутренних компонент и количеством связей между ними. Связи могут быть

разнотипными. Сложность архитектуры определяется количеством связей.

Сложность структуры программы – функция модулей и связей. Компоненты сложности:

1) Сложность самой проблемы

2) Сложность управления процессом разработки

3) Сложность обеспечения гибкости конечного программного продукта

4) Описание поведения отдельных подсистем.

Продолжать

Большие системы имеют тенденцию к эволюции в процессе их использования. Сложность

настолько высока, что один разработчик не может учесть всех тонкостей. Измеряется в

человекагодах. Например, для Windows NT – 2000 чел-т. Возникают трудности с

координацией работы. Главная проблема руководства – поддержание целостности основной

идей в ходе работы. ПО нужно разрабатывать таким образом, чтобы продукт можно было

легко модифицировать и развивать. Проектировать желательно так, чтобы взаимовлияние

частей было меньше.  
**5)Основные концепции, применяемые для уменьшения сложности программной системы.**

Для решения проблемы сложности использовуются концепции декомпозиции, абстракции и

иерархии.

**Декомпозиция** – такое разбиение системы, чтобы высокочастотная динамика ее была

заключена в данных компонентах, а межкомпонентное взаимодействие предоставляло лишь

низкочастотную динамику системы (Г.Буч). Т.е.задача эта сродни разрезания графа, чтобы

было минимум ребер. Сущ-ет декомпозиция алгоритмическая и объектно-ориентированная.

Алгоритмическая – обычное разделение алгоритмов, где **каждый** модуль системы выполняет

один из важных этапов процесса *(рисунок: основная функция – f1,f2,f2,….от них далее и т.д.)*

Объектно – ориентированная. Критерий разделения – принадлежность объекта различным

абстракциям данной предметной области.

*(тут нарисована диаграмма классов, но в конспекте ее не было)*

Особенности: основана на объектах, а не на алгоритмах.

**Абстракция** – это такие сущ-ные характеристики объекта, которые отличают его от др. видов

объектов и определяют его особенности с точки зрения рассмотрения и анализа.

Абстрагирование позволяет получить некоторую модель предметной области, которая

содержит объекты, отражающие реальные объекты. В процессе сущ.отсечение

несущественных деталей.

**Иерархия** позволяет разделить систему по уровням понимания. Бывает структурная и по типу.

Структурная: отношение «целое – часть». То есть отношение ассоциации и агрегации.

По типу: отношение «тип – подтип».

Концептуальные особенности структурного и ООП:

1)схема водопада 1)спиральная модель

2)абстракция (структуры) 2)абстракция (классов)

3)декомпозиция (алгоритм.) 3)декомпозиция (об. –ориент.)

4)иерархия (структурная) 4)иерархия (структ.- типовая)

**6)Структурное проектирование.**

Требования: пользователь формирует свои потребности к будущему проекту. Проекты по

такому принцип можно разбить на:

1.Управляемые пользователем

2.Контролируемые пол-ем

3.Не зависимые от пол-ля

Схема структурного проектирования;

Требования заказчиком полностью, ответственность за опр. требования – на разработчика.

Требования к крупной системе осущ. спец. группой людей: сост. из представителей заказчиков и

разработчика. Один от заказчика имеет право для принятия решения (босс), но не явл.

пользователем, другой опытный пользователь. Процесс заключается:

- анализ имеющихся систем

- беседа с потенциальными пользователями

- проведение оценки осуществленности и достоинств проекта. Рассмотреть задачи, относящиеся к

проекту

- проверка правильности требований

Требования

Цели

Предварительно

е внешнее

проектирование

Детальное

внешнее

проектирование

Стук-ра

программы

Внеш.проек.мод

уля

Проект.лог

ики модуля

Проект БД

Арх.сист

емы

П(проект)

А(анализ)

Пользователь ответственен за проверку требований на полноту и точность, а разработчик – за

проверку осуществимости и понятности. Для каждого требования - приоритет, чтобы принимать

компромиссные решения.

Цели: конкр. ориентиры для проектирования программного продукта. Бывают цели продукта и

цели проекта. ЦПродукта с точки зрения пользователя, ЦПроекта – для разработчика.

C точки зрения пользователя:

1. Резюме

2. Определение роли пользователя, перечисление ф - ий

3. Документация (цели для документации, типы документации и круг читателей для каждого

типа)

4. Эффективность: характеристики, пропускная способность и т.п.

5. Конфигурация: указывает, от чего программа будет зависеть и работать

6. Безопасность обеспечения цели

7. Обслуживание: намеченная стоимость и время исправления ошибок

8. Установка – методы и средства настройки системы под конкретные условия

9. Надежность

Цели относятся к разработке и в явном виде не проявляются, в конечном результате:

- ориентировочная стоимость каждого этапа

- календарный план проекта

- цели для каждого процесса тестирования

- степень адаптированности ПО, которая дожна быть достигнута

- вопросы сопровождения системы, которые необходимо учитывать при разработке

- уровни надежности для каждого этапа разработки для достижения заданной надежности

системы

- внутренняя документация проекта

- критерий для оценки готовности продукта к использованию.

**11)Внешнее проектирование**.

Это процесс описания ожидаемого поведения разрабатываемого продукта с точки зрения

внешнего по отношению к нему наблюдателя. Цель – конструирование и описания внешних

взаимодействий без конкретизации внутреннего устройства. Результаты выражаются в форме

внешних спецификаций. Важно соблюдать концептуальную целостность продукта, который

предпочитает максимально соответствовать между внешними функциями системы.

Ответственные за внешние п. – системные аналитики и специалисты по GUI.

Внешнее проектирование мало связано с программированием. Больше касается проблем и

нужд пользователя и психологии общения человека с машиной.. Существует 2 этапа:

предварительный и детальное внеш.проектирование.

Предварительный: определяет все функции пользователя без определения синтаксиса и

семантики.

Детальное: описывает входные/выходные данные, преобразование системы; характеристики

надежности, эффективность и замечания по программированию.

Входные данные: точное описание синтаксиса и семантики: формат, допуст. значения, обл.

изменения.

Выходные данные: точное описание всех результатов ф –ии: реакция сообщение об ошибке и

т.д.

Должна описываться функц. связь между вх/вых данными. Должны быть описаны результаты

всех не правильных вх. данных. Изменение состояния системы – многие ф –ии могут сие

осуществлять. Это должно описано с точки зрения пользователя.

Характеристики надежности – описание воздействия всевозможных отказов ф-ии на систему и

ее файлы.

Эффективность ф-ии – затрачиваемое время и память.

Замечания по прогр. – описание идей от-но устр-ва ф-ии. Проверка правильности имеет

важное значение, т.к. в.п. – поворотный этап проекта.

**12)Проектирование архитектуры системы.**

Это описание всей системы в виде взаимодействия подсистем. Сущ-т 2 основных подхода к

проектированию систем:

- на выделение абстракции

- на выделение подсистем, уравлен. методом портов

1)- правильное распределение методов в иерархии

2)- основное внимание на связи между подсистемами и обмен данными.

Концепция уровня абстракции. Предложена Дейкстрой в конце 60 –х: система разбивается на

различные иерархические упорядоченные части: уровни, которые удовлетворяют опред.

критериям. Уровень представляет группу непосвященных модулей. Идея уровней призвана

минимизировать сложность системы за счет такого их определения, которые обеспечивает

высокую степень независимости уровней др.от др.

Прикладной уровень

Уровень безопасности вычисл.системы

Нижние уровни работают с аппаратурой или с ОС.

Основные свойства, хар-ые для ур. абстракции:

УР

УР

УР

1) На каждом уровне абсолютно ничего не известно о св-ах и сущ-нии более высоких

уровней.

2) На каждом уровне ничего не известно о внутрен. строении др. уровней. Связь между ними

только через жесткие заранее определенные сопряжения.

3) Каждый уровень представляет собой группу модулей. Некоторые явл. недоступными для

др.уровней, некоторые известны используются для сопряжения.

4) Каждый уровень может обесп. некоторую абстракцию данных в системе, скрывая от

др.уровней реальную структуру этих данных.

5) Связи между уровнями ограничены явными аргументами, пред. с одного уровня на др.

Недопустимо глоб. данные нескольких уровней.

**Концепция методов портов.**

Три основ. группы механизмов для взаимодействия:

1) Передача управления без передачи данных.

2) Передача управления с передачей данных

3) Передача данных без передачи управления

При использовании механизма портов система делиться на несколько подсистем, каждая из кот.

дел. на несколько входных очередей. Каждая система рассматривается как асинхронный процесс.

Если одна хочет передать данные др., то посылает их во входной порт подсистемы. И читает из

своих входных портов свои входные данные

……………. К портам др. системы

Первичные вх. ……..

данные вторичные вх.порты

Первичные – данные для основных ф- ий системы.

Вторичные – для связи с подчиненными процессами. Если требуется инфа из др.подсистемы, то

посылает запрос и ожидает ответ на вторичном порту.

Рассм. схема - часть прямой конфигурации метода портов (посылка сообщений напрямую)

16п 1 вых

2 вых

20п

Иногда выходные порты не сущ-т физически, а явл. наимен. входн. портов др. подсистем.

подсистема

подсистема

**Проектирование структуры программы** – определяет все модули подсистемы, их связи

сопряжения. Модуль имеет три основных атрибута:

1) Выполнение ф-ии

2) Обладание логикой

3) Используется в одном или нескольких контекстах.

Функция – внешнее описание модуля, что он делает.

Логика описывает внутр. алгоритм модуля.

Контекст описывает конкр. применения модуля.

На практике разбиение на модули не приводит к уменьшению сложности по трем причинам:

1) Модули выполняют слишком много ф – ий

2) Логика может быть разнесена между разными модулями

3) Непредсказуемое поведение модулей, которые используют общ. данные.

Для решения использования сущ. композиционное проектирование. Оно состоит из двух

компонент: система оценок/критериев, по которым оценивается модуль, композиционный анализ.

**Характеристики модулей**: независимость, прочность сцепления. Уменьшение сложности

способствует высокой независимости, достигается двумя методами оптимизации:

1) Усиление внутр. связей в модуле

2) Ослабление взаимосвязей между ними

Прочность модуля – мера его внутренней связи. Сущ. 7 классов прочности:

- прочность по логике

- прочность по классу

- модуль прочный по совпадению

- процедурно – прочный модуль

- коммуникационно - прочный

- информационно и функционально прочный

1 – между элементами которого нет осмысленных связей, могут возникнуть, когда

обнаруживается одинаковая посл-ть ком. в разных участках. Коды группируются в отдельный

модуль. Особенность: эта последовательность не обладает законченной и осмысленной

функциональностью. Хотя последовательности кажутся идентичными, но могут быт использованы

для различных целей в различных местах программы. Этот модуль тесно связан с вызывающими

его модулями.

2 – выполняет одну из ф- ий из набора связанных с ним. Выбираемая ф-я обычно запрашивается

вызывающим модулем. Главная проблема с модулями такого типа – использование одного итого

же сопряжения для разных ф – ий.

3 – модуль выполняет определенную последовательность связанных с ним ф -ий. Например,

загрузочный модуль. Обычно такие модули неявно связаны с др.модулями программы.

4 – выполняется набор ф –ий, непосредственно относящийся к решению задачи.

5 – процедурно – прочный с ограничением: все его ф-ии связаны по данным

6 – выполняет несколько ф-ий. Каждая из них работает с одной и той же структурой данных.

7 – выполняет одну определенную ф-ю

Во внешней спецификации: имя модуля, функция, которая реализована; список параметров,

формат, пределы, внешние эффекты.

Проектирование логики модуля. Структурное программирование и пошаговая реализация. Между

текстом программы и порядком выполнения – прямая связь, то есть без GOTO.

Сцепление – мера взаимодействия модулей по данным. Характ. способом передачи и св-ми самих

данных. Сущ. 6 видов:

1) Сцепление по содержанию

2) По общей области

3) По внешним данным

4) По управлению

5) По формату

6) По данным

1 – прямое сцепление(ассемблер)

2 –сцепление по общей области(если ссылаться на первую глобальную структуру

данных).Ссылающиеся на абсолютные адреса, тоже относятся.

Глобальные данные сильно связывают автоматные модели и сипетрии**. Шлеер – Меллер**:

деление объектов на пассивные (для хранения данных) и активные(облад. сложным поведением).

Активные описываются автоматом Мура. Их метод состоит из трех этапов:

1. Создание информационных моделей

2. Создание моделей состояния

3. Создание моделей процессов

1 – абстрагирование концептуальных сущностей в терминах объектов и атрибутов. Отношения

между сущностями формализуются в связи, как основа линии поведения, законах и правилах

реального мира. Результат – построение диаграммы классов для подсистем.

2 – на диаграмме классов выделяют активные объекты и для них строиться автомат Мура. Для

каждой модели переходов состояний все описывается(действия).

**Домен** – четко определенные предметные области. Домен – реальный гипотетически или

абстрактный мир, населенный отчетливым набором объектов, которые ведут себя в соответствии с

правилами поведения в этом домене. Домен расссм., как единица для цели анализа. Схема

доменов:

Это деление по принципу уровней абстракцией Дейкстры. Уровни: 1 – прикладной, 2 – сервисный

подсистемы, 3 – архитектурный, 4 – реализация.

1 – поддержка общих механизмов и сервисных ф – ий для прикладного.

3 –обеспечивает общ. механизмы управления данными и системой как единым целым. В нем

содержаться интерфейсы.

Информационная модель 20 – 60 классов. Строятся диаграммы классов. Далее строиться автомат

для классов. Например: для желаемой дороги инф. модель – набор классов поезда, вагонов, путей

и т.д. Автомат – поведение поезда. Для согласования поведения различных объектов строятся

отдельные модели взаимодействия объектов, где отображаются какие события и какие объекты

посылаются.

Модель процессов –каждое действие под каждое состояние объекта.

1) Имеется единственная кнопка доступная пользователю, нажатие при закрытой двери

включение подогрева на одну минуту.

2) Повторное нажатие , добавление одной минуты

3) Нажатие при открытой двери эффекта не имеет.

4) Лампочка внутри должна быть включена, при открытии двери и при готовке.

5) При открытии двери готовка остановлена.

6) При закрытии лампочка гаснет

7) По истечении времени выключается все и подается звук

Микроволновка

Lamp

WElement

Регистрация

тенденций

прикладной

Процессы

I/O

Использова

ние

интерфейса

архитектурный

ОС

сеть

ЯП

сигналы

Создание автомата *(схема, только ее не будет!!!).* Такая форма автомата предназначена для

проектировщика. Сущ-т еще одна форма автомата в виде таблиц переходов

события

состояния V1 V2 …..

1

2

В ячейках: новое состояние/игнор., события/событие не может произойти.

От проектирования к реализации: (*схема, которой тоже не будет!)*

Описание к этой схеме:

Publik:

Void Event(int number);

Protected:

Void Action(int number);

**Патерны.**

«State» состояние, представляет собой объект. Используется, когда поведение объекта зависит от

внутреннего состояния, когда встречаются многоветвевые операторы в коде(выбор зависит от

состояния). Переключает состояния либо Context, либо ConcretState. Объект состояния можно

совместно использовать другими состояниями, но чаще лучше использовать таблицы, там не

нужно менять кодировку.

**Тестирование**

Существуют разные подходы к определению работы. Как правило 1 роль = 1 чел. Но в небольших

компаниях сотрудник выполняет несколько функций. Project manager – руководитель проекта,

отвечает за качество продукции, планирование работ, составление бюджета, взаимодействует с

клиентом. Team bader может быть.

Проектировщики. Роли:

1. Разработчик архитектуры. Обмен данными между подсистемами. Опред. Обеспечение

гибкости системы и повторное использование компонентов. Может составлять план

«Тестирование стеклянного ящика» на самом верхнем уровне и анализировать обзоры всех

спецификаций.

2. Специалист по анализу предметной области (software analyst, subject matter expert) .

Должен разбираться в составе предметной области, должен понимать, чего хотят

пользователи и как выразить это в терминах, понятных другим разработчикам.

3. Специалист по человеческому фактору (human factor analyst, ergonomist). Имеет

психологическое образование и знает, как спроектировать программу, чтобы она была

понятной и удобной. Некоторые такие могут непосредственно разрабатывать

пользовательский интерфейс

4. User interface programmer – специализируется на создании пользовательского интерфейса:

как написать и в каком виде отобразить

5. Team programmer\team bader – ведущий программист. У него имеется некоторая команда

программистов. Сам он занимается спецификацией, которая относится ко внутренней

структуре проекта. Может отвечать за разработку частей архитектуры.

6. Менеджер по маркетингу (project manager, product marketing manager) отвечает за

соответствие продукта долгосрочной стратегии и имиджу кампании, а также за рекламу,

распространение новых версий и др. Эти специалисты определяют требования рынка,

функции и возможности продукта, от которых зависит его конкурентоспособность.

7. Представитель группы технической поддержки (technical support). Контактирует с

пользователями. Исправляет ошибки и добавляет новые функции на этапе эксплуатации.

Выполняет анализ жалоб пользователей и отвечает на их вопросы. Участвуют в

обсуждениях на этапе анализа, при создании документации, чтобы сделать ее понятной

(для минимизации кол-ва обращений пользователей)

8. Технический описатель (technical writer) – делает документацию и интерактивную справку

9. Тестировщик (tester). Тестирует продукт по методу черного ящика

Цели тестирования – протестировать программу. После тестирования программа будет работать

правильно. Задача тестировщика – гарантировать правильность работы программы путем

проведения полного тестирования

Причины, по которым невозможно полное тестирование:

1. Кол-во комбинаций входных данных слишком велико

2. Кол-во возможных ветвей в коде тоже слишком велико

3. Пользовательский интерфейс слишком сложен для полного тестирования

Для правильной реакции системы на все комбинации входных данных необходимо:

1. Поверять сочетание всех допустимых значений, сочетания недопустимых входных

значений, все способы редактирования входных данных.

2. Реакция на одни и те же входные данные зависит от текущего состояния системы

Тезис о логическом обосновании правильности программы – проанализировав логику программы

можно определить ее состояние в любой точек кода в любой момент времени. Однако сама

спецификация может содержать ошибку. Программа не работает абсолютно правильно и всегда

содержит ошибки. Вопрос в их количестве и степени последствий из проявления.

Задача тестировщика – находить и исправлять ошибки. Минимизация ошибок и минимизация

последствий за ограниченное для тестировщика время.

**Виды тестирования:**

1. Структурное

2. Функциональное

3. Восходящее

4. Нисходящее

5. Целостное

6. Статическое

7. Динамическое

8. Псевдоотладка

9. Мутационное тестирование

**Регрессионное тестирование:**

1. «Черный ящик» - неизвестна внутренняя структура программы. Подбирает входные

данные.

2. «Стеклянный ящик» - как правило, программисты. Тесты основаны на знании кода.

Задача – подобрать тесты для тестирования внутренних компонент системы.

Преимущества: детальное тестирование, полный обхват кода. Управление потоком:

программист знает, какая функция будет следовать и каким будет ее текущее

состояние, как и какая часть программы изменяет каждый элемент данных. Видны все

граничные точки, которые скрыты от взгляда извне. Недостатки: это делает

программист, и из-за дополнительного кода накладные расходы. Тестирование

становится частью процесса программирования

**Структурное тестирование** – разновидность тестирования стеклянного ящика. Главное – выбрать

правильные пути. Используется так называемый «критерий охвата» (полноты):

1. Критерий охвата строк: наиболее сложный, чтобы выполнялась каждая строчка кода

2. Критерий охвата ветвлений: проверяется каждый if и else.

3. Критерий охвата условий: наиболее строгий. Нужно проверить все составляющие

переменные условного оператора для каждого логического условия

**Функциональное тестирование** – относится к черному ящику. Каждая функция программы (с

точки зрения пользователя) тестируется на входных и выходных данных.

**Восходящее тестирование** – тестируется нижний уровень иерархии модулей, затем более

высокие и т.д.

**Нисходящее** – наоборот.

Достоинства и недостатки: восходящий – прекрасный способ локализации ошибок. Обнаруженная

ошибка локализована в одном-двух модулях. Оно, как правило, проводится более тщательно.

Главный недостаток – написание специального кода-оболочки, который вызывает тестируемый

модуль. Нужно писать с заглушками. Обе эти технологии называются инкрементными. А

совместное тестирование называется интеграционным, как часть инкрементирующего.

**Целостное тестирование** – противоположность инкрементному. Предполагается, что до полной

интеграции системы модули не проходят особо тщательного тестирования. Преимущество: не

нужен дополнительный код. Недостатки: трудно выявить источник ошибки, трудно организовать

исправление ошибки.

**Динамическое тестирование** – это стеклянные и черные ящики.

**Статическое тестирование** – код не выполняется, а логически анализируется. Можно (и

рекомендуется) использовать компилятор для исправление синтаксический ошибок.

**Псевдоотладка** (debugging) . В программу внедряется ряд ошибок, и тестировщик, зная что они

есть, будет тестировать гораздо тщательнее. По проценту найденных\внесенных ошибок можно

примерно определить качество тестирования.

**Мутационное** – в программу вносятся маленькие изменения (мутации). В процессе тестирования

такое изменение должно проявиться. Если не проявляется, то тест неудачный.

**Регрессионное тестирование** – основная работа тестировщика. Имеет 2 значения, объединенных

идеей повторного использования ранее разработанных тестов.

Задача регрессионного тестирования состоит в том, чтобы убедиться, что ошибка исправлена.

Проводится тест, обнаруживается ошибка. Проводится тот же тест, чтобы убедиться.

Еще одна задача – тестирование целостности программы, а не только исправление одной ошибки.

Убедиться, что исправление одного не испортит другое.

**Компонентный подход**

**COM (Component Object Model)**

( *схема про то, что компоненты присоединяются к ядру с помощью интерфейсов)*

В COM компоненты состоят из бинарного кода (в виде Dll\EXE). Подключаются механизмы не

совсем обычного Dll. Объявляют о своем присутсвии стандартным образом. Клиентские

программы могут находить компоненты COM и локально, и в сети. COM – спецификация для

регистрации и создания компонентных приложений. Есть API.

Можно рассматривать COM как новый стиль программирования. Первоначально это было OLE

(Object Lining & Embedding). Использовалась технология DDE (Dynamic Data Exchange), которая

основана на передаче WinMsg.

Преимущества:

1. Хорошая адаптация приложений ( все на основе готовых компонентов)

2. Повторное использование кода

3. Возможность приложения игнорировать местоположение своих компонент

Основа COM-технологии – интерфейсы. Для спецификации интерфейса используется GUID

(Global Unique ID) – это 128битный код. ProgID может быть и не уникальным

Должно поддерживаться каждым объектом COM:

1. IUnknown

2. IDispatch

3. IClassFactory

4. ITypeLib

5. ITypeInfo

6. QueryInterface() – получение доступа к интерфейсу

7. AddRef() – увеличение счетчика ссылок на объект

8. Realese() – уменьшение счетчика ссылок на объект и уничтожение COM-объекта \_\_\_\_\_\_\_\_\_oпри

достижении нуля

Этот интерфейс может дать указатель на любой другой интерфейс, поддерживаемый объектом.

class IAccount : public IUnknown //GUID подставит Define-on

{

public:

virtual HRESULT GetBalance(int \*pBal) = 0;

virtual HRESULT Dispose(int count) = 0;

};

void main()

{

CoInitialize(NULL);

IUnknown \*PIUnk = NULL;

HRESULT hr =

CoCreate Instance(CLSID\_MyComponent, //GUID компонент

NULL,

CLSCTX\_INPROC\_SERVER, //где искать

IID\_IUNKNOWN, //на какой интерфейс дать доступ

(void \*\*)&pIUnk); //GUID интерфейса

IAccount \*pIAcc = NULL;

hr = pIUnk->QueryInterface(IID\_IACCOUNT, //GUID интерфейса

(void \*\*)&pIAcc);

}

Есть GUID компонентов, а есть GUID интерфейсов. В реестре есть и GUID , и ProgID, и

расположение библиотеки

HKEY\_CLASSES\_ROOT

//тут тоже какая-то схема

**Отличие COM-Dll от обычной Dll:**

Функции:

DLLGetClassObject – для создание фабрики классов

CLLRegisterServer - регистрация

DLLUnregisterServer – удалелие компонента из реестра Windows

DllCanUnloadNow – сообщает, что Dll может быть выгружена. Это вызывается системой.

regsvr32.exe

regsvr32 mydll.dll

regsvr32 –a

Доступ к компонентам можно осуществлять двумя способами:

1. Прямой доступ к интерфейсу (был рассмотрен)

2. При помощи интерфейса IDispatch. Он – производный отIUnknown и обеспечивает

уневерсальный доступ к методам и свойствам COM-объектов. IDispatch на порядок

медленнее, когда компонента загружается с данного компьютера

GetIDsOfNames – получение ID функции по имени

GetTypeInfo - получение информации о методах, параметрах

GetTypeInfoCount

Invoke – вызов методов интерфейса по ID

OleInitialize(NULL);

wchar\_t progid[] = "";

CLSID clsid;

IUnknown \*PIUnk = NULL;

HRESULT hr = CLSIDfromProgID(progid, &clsid);

IDispatch \*pDisp = NULL;

hr = CoCreate Instance(clsid, NULL, CLSCTX\_INPROC\_SERVER,

IID\_IDISPATCH, (void \*\*)&pDisp);

DISPID dispid;

OLECHAR \*frame = L"GetBalance";

hr = pDisp->GetIDsOfNames(Frame, 1, GetUserDefaultLCID(), &dispid);

//подготовка параметров

VARIANTARG varg;

VariantInit(&varg);

varg.plval = 0;

varg.vt = VIBYREF | VI\_I4; // указатель на Int4

DISPPPARAMS param;

param.CArg = 1;

param.rgvarg = &varg; // указатель на строку

param.cNamedArgs; // нет именованных аргументов

param.rgdispidNamedArgs = NULL;

hr = pDisp->Invoke(dispid, IID\_NULL,

GetUserDefaultLCID(), DISPATCH\_METHOD,

&param, NULL, NULL, NULL);

typdef struct tag Variant; {

...

VARTYPE vt;

...

union {

LONG lVal;

SHORT iVal;

LONG \*plVal;

...

};

} VARIANT

DIM obj AS NEW Account; // имя интерфейса

DIM balance AS LONG;

obj. GetBalance balance;\_\_