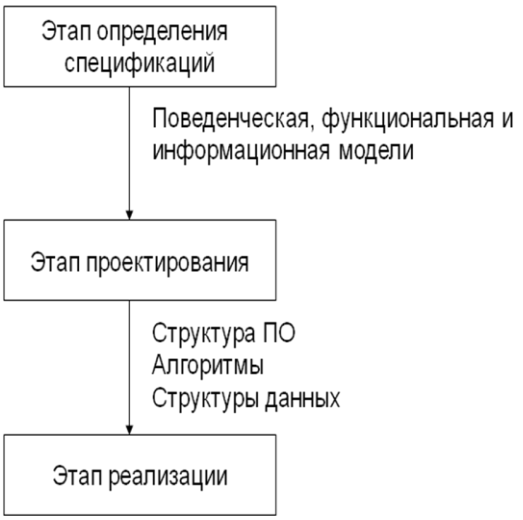
1. Процесс создания ПО. Сущность этапа проектирования.

Основные этапы (стадии) разработки ПО:

- Постановка задачи
- Анализ требований и разработка спецификаций
- Проектирование
- Реализация

По методологии RAD (Rapid Application Development):

- Анализ и планирование требований
- Проектирование
- Реализация (построение)
- Внедрение



Проектирование – процесс преобразования спецификаций ПО в инженерные представления о нем.

Две основных ступени проектирования:

I. *Предварительное проектирование.*

Обеспечивает:

- идентификацию подсистем;
- определение основных принципов управления подсистемами их взаимодействия.

Включает в себя следующие этапы:

1. Структурирование системы.
2. Моделирование управления.
3. Декомпозиция подсистем на модули.

II. *Детальное проектирование.*

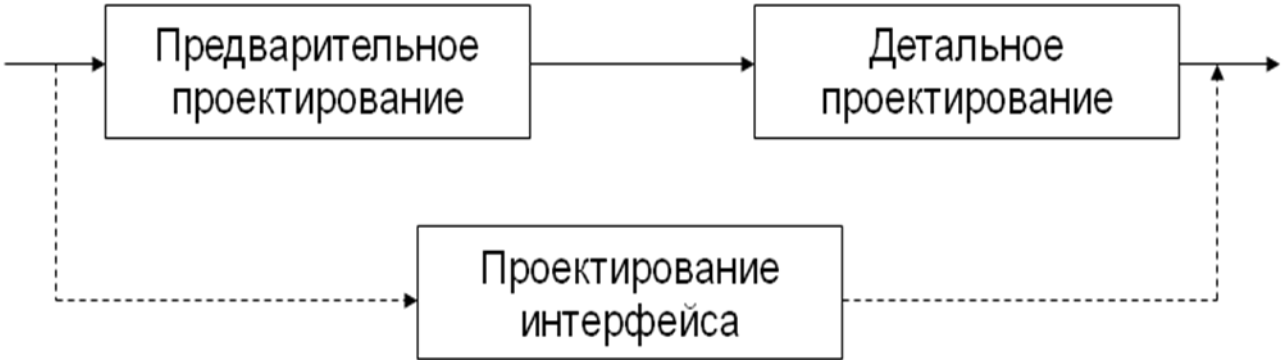
Обеспечивает:

- выработку алгоритмических и структурных подробностей, обеспечивающих логически простую реализацию на языке программирования.

Включает в себя:

1. Разработку детальных алгоритмов.
2. Разработку представления структур данных на физическом уровне.

III. *Дополнительно – интерфейсное проектирование.*



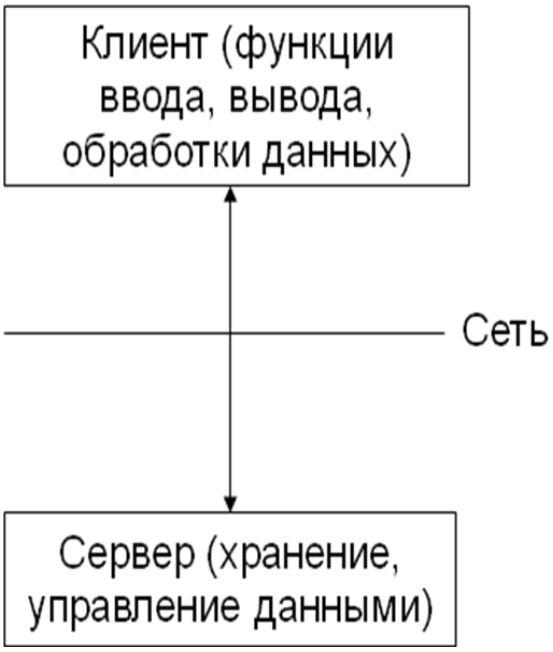
2. Архитектура программной системы. Модели управления.

Архитектура программной системы.

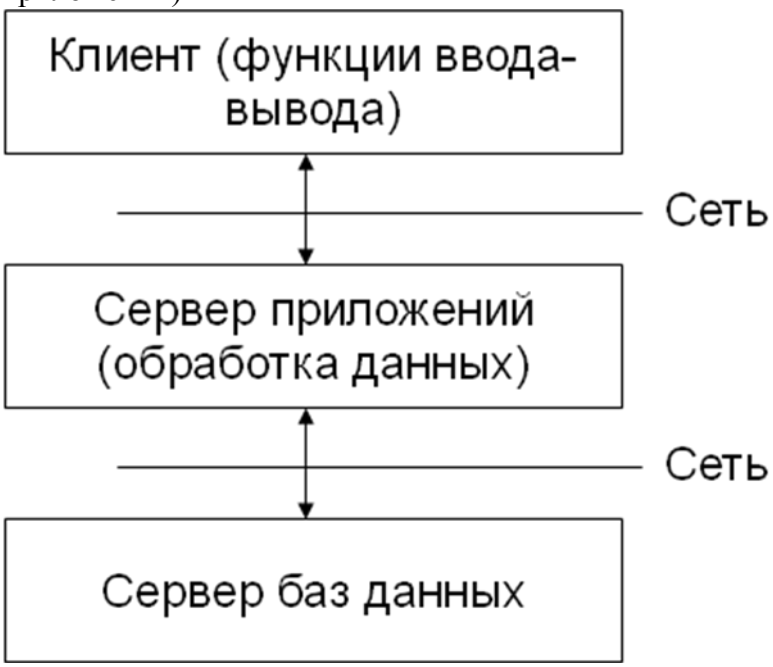
Модели системного структурирования:

1. Модель хранилища данных

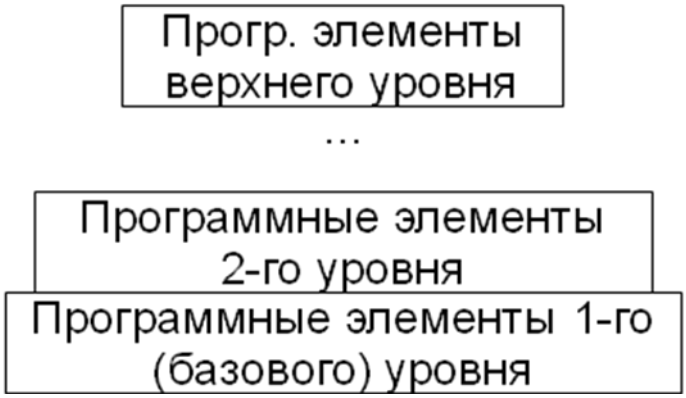
2. Модель клиент-сервер (двухуровневая)



3. Трехуровневая модель клиент-сервер (модель сервера приложений)



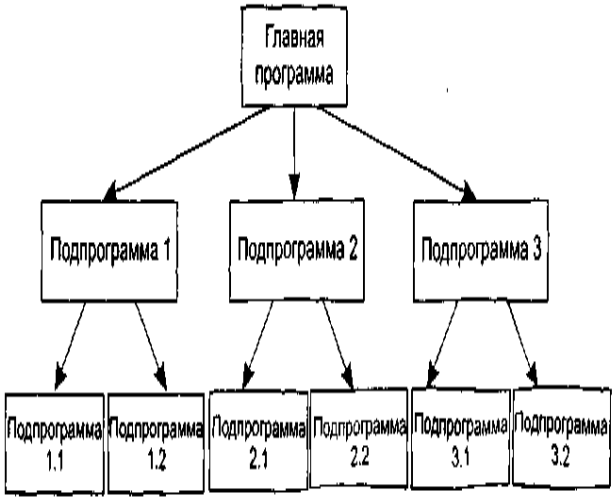
4. Многослойная модель



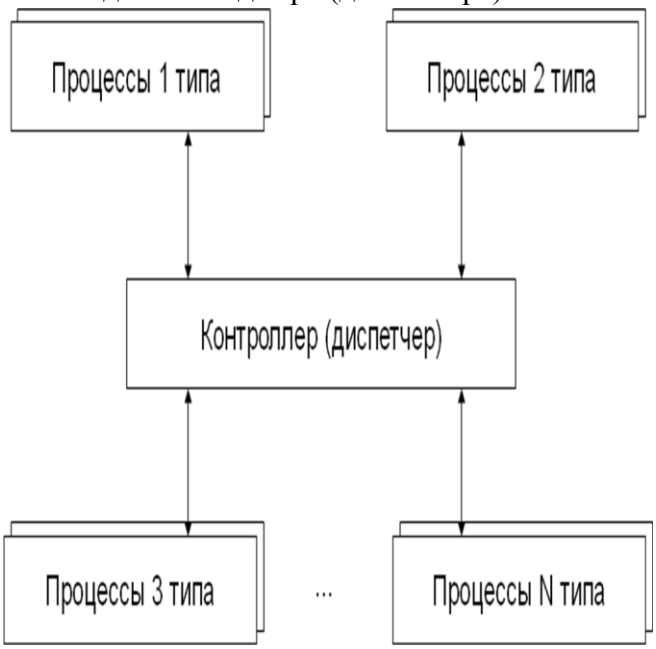
Типы моделей управления:

1. Модели централизованного управления

1.1. Модель вызов-возврат



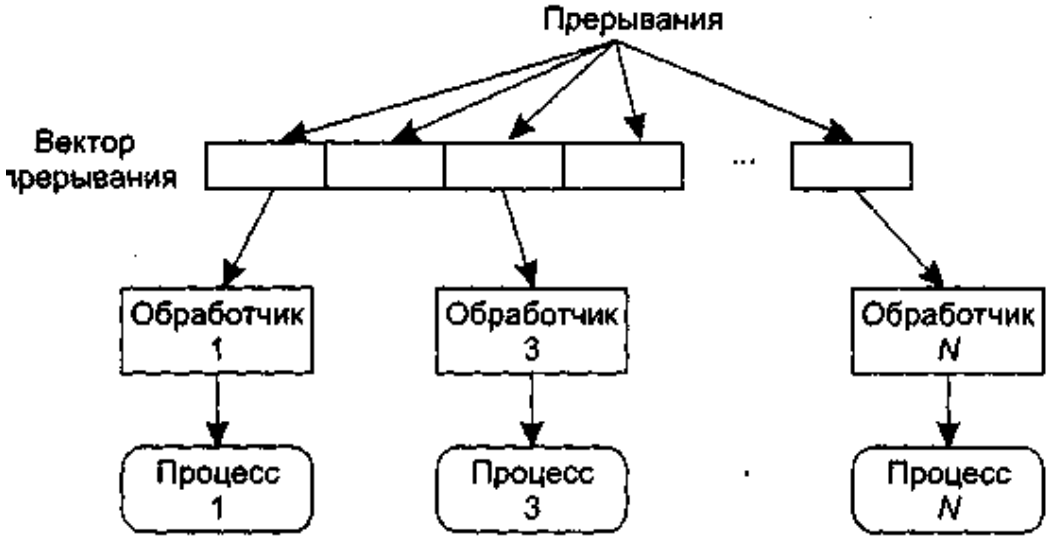
1.2. Модель менеджера (диспетчера)



2. Модели событийного управления

2.1. Широковещательная модель

2.2. Модель управления по прерыванию

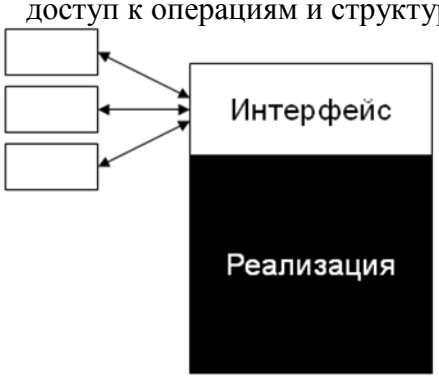


3. Выделение модулей. Свойства модулей.

Основное правило выделения модулей:

Принцип информационной закрытости – содержание модулей должно быть скрыто друг от друга:

- все модули независимы, обмениваются только информацией, необходимой для работы;
- доступ к операциям и структурам данных модуля ограничен.



Свойства модулей.

Связность

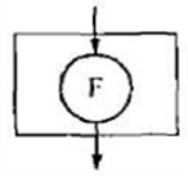
Связность модуля – это мера зависимости его частей (внутренняя характеристика модуля).

Типы связности:

1. Функциональная связность (CC=10)

Модуль содержит элементы, участвующие в выполнении одной и только одной проблемной задачи. Примеры модулей:

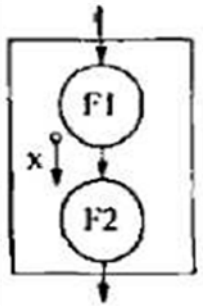
- Вычислять синус угла;
- Проверять орфографию;
- Вычислять зарплату сотрудника;



2. Информационная (последовательная) связность (CC=9)

Выходные данные одной части используются как входные данные в другой части модуля. Пример модуля:

```
Модуль Прием и проверка записи
   прочитать запись из файла
   проверить контрольные данные в записи
   удалить контрольные поля в записи
   вернуть обработанную запись
Конец модуля
```

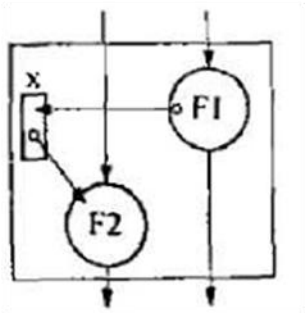


3. Коммуникативная связность (CC=7)

Части модуля связаны по данным (используют одни и те же структуры данных).

Пример модуля:

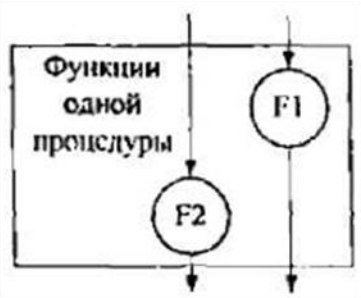
```
Модуль Отчет и средняя зарплата
   сгенерировать Отчет по зарплате
   вычислить параметр Средняя зарплата
Конец модуля
```



4. Процедурная связность (CC=5)

Модуль состоит из элементов, реализующих независимые действия, для которых задан единый порядок работы, то есть порядок передачи управления (связности по данным нет). Пример модуля:

```
Модуль Вычисление средних значений
   вычислить среднее по Таблица-А
   вычислить среднее по Таблица-В
   вернуть среднееТабл-А. среднееТабл-В
Конец модуля
```



5. Временная связность (CC=3)

Части модуля не связаны, но необходимы в один и тот же период работы системы.

6. Логическая связность (CC=1)

Части модуля объединены по принципу функционального подобия.

7. Случайная связность (CC=0)

В модуле отсутствуют явно выраженные внутренние связи.

Тип связности	Сопровождаемость	Роль модуля
Функциональная	Хорошая	«Черный ящик»
Информационная		Не совсем «черный ящик»
Коммуникативная	Средняя	«Серый ящик»
Процедурная	Плохая	«Белый» или «просвечивающийся ящик»
Временная		«Белый ящик»
Логическая		
По совпадению		

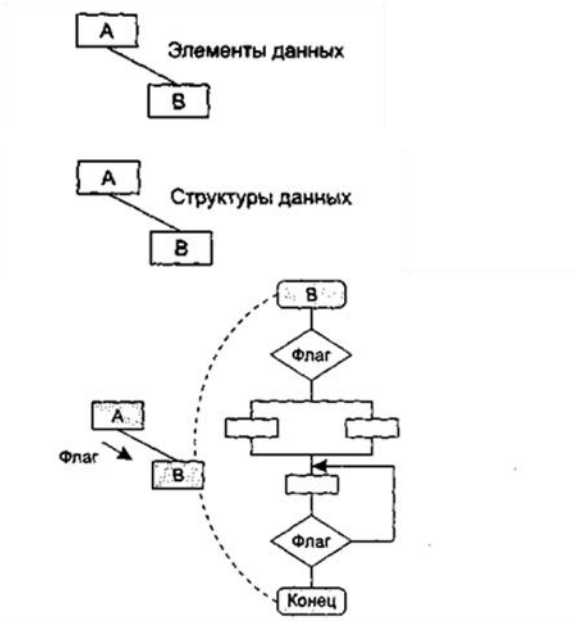
Схема определения связности модуля

Сцепление

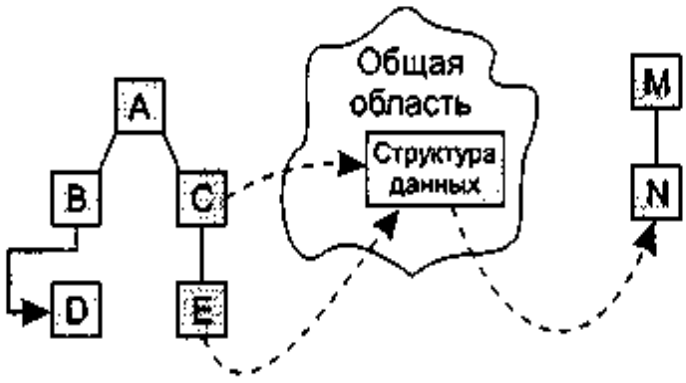
Сцепление модуля – мера взаимозависимости модулей (внешняя характеристика модуля).

Типы сцепления:

1. Сцепление по данным (СЦ=1)
Модули обмениваются данными, представленными скалярными значениями.
2. Сцепление по образцу (СЦ=3)
Модули обмениваются данными, объединенными в структуры.
3. Сцепление по управлению (СЦ=5)
Модуль посылает другому некоторый информационный объект (флаг), предназначенный для управления внутренней логикой модуля.



4. Сцепление по общей области (СЦ=7)
Модули разделяют одну и ту же глобальную структуру данных.
5. Сцепление по содержимому (СЦ=10)
Модуль содержит обращения к внутренним компонентам другого (передает управление внутрь, читает и/или изменяет внутренние данные или сами коды).



Тип сцепления	Сопровождаемость
По данным	Хорошая
По образцу	
По управлению	Средняя
По общей области	Плохая
По содержимому	

4. Разработка структурной и функциональной схем. Метод пошаговой детализации.

Разработка структурной и функциональной схем.

На этапе предварительного проектирования при структурном подходе уточняется структура ПО, т.е. определяются структурные элементы и связи между ними.

Результат – структурная и/или функциональная схемы.

Структурной называют схему, отражающую *состав и взаимодействие по управлению* частей разрабатываемого программного обеспечения.

Функциональная схема или схема данных - схема *взаимодействия* компонентов программного обеспечения *с описанием основных функций и информационных потоков*, состава данных в потоках и указанием используемых файлов и устройств.

Структурными компонентами программной системы или программного комплекса могут служить программы, пакеты программ, подсистемы, библиотеки ресурсов и т. п.

Для изображения функциональных схем используют специальные обозначения, установленные стандартом ГОСТ 19.701-90.

Метод пошаговой детализации

Основной прием – структурная декомпозиция.

Результатом декомпозиции является структурная схема программы, которая представляет собой многоуровневую иерархическую схему взаимодействия программных компонентов по управлению или иерархическую схему включения программных компонентов.

Основные принципы:

- нисходящий подход;
- принцип вертикального управления;
- принцип необходимой детальности.

Рекомендации при выделении модулей (подпрограмм):

- не отделять операции инициализации и завершения от соответствующей обработки;
- избегать излишне специализированных и излишне универсальных модулей;
- избегать дублирования действий в различных модулях;
- группировать сообщения об ошибках (информационные сообщения) в один модуль.

5. Особенности и выразительные средства проектирования ПО с использованием объектного подхода.

Укрупненные этапы:

Логическое проектирование – детальное описание объектов, полученных при объектной декомпозиции, полное описание полей и методов каждого класса.

Физическое проектирование - объединение классов и других программных ресурсов в программные компоненты, а также размещение этих компонентов на конкретных вычислительных устройствах.

1. Разработка структуры программного обеспечения
- Формируется набор классов для реализации.
- Если количество классов и других ресурсов велико, то они объединяются в группы – пакеты. В один пакет обычно собирают классы и другие ресурсы *единого назначения.* Связи между пакетами обозначают вызовы.
- Строится диаграмма пакетов.
2. Определение отношений между объектами
- За основу берется диаграмма классов этапа определения спецификаций.
- Анализируются варианты использования, уточняются взаимодействия объектов в процессе реализации вариантов.
- Строятся диаграммы последовательностей этапа проектирования или диаграммы кооперации.
3. Уточнение отношений классов
- На этапе проектирования рассматриваются еще два вида отношений: агрегация и композиция (подвиды отношения ассоциации).
- Выделяются классы особого вида – интерфейсы, содержащие только объявления методов.
- Строится диаграмма классов этапа проектирования.
4. Проектирование классов (окончательное определение структуры и поведения объектов)
- Окончательное определение набора атрибутов (полей) и операций (методов) класса.
- При необходимости реализации сложного поведения класса, разрабатываются диаграммы состояний объекта. Определяют состояния объекта, возможные переходы, условия выполнения перехода и действия, выполняемые в состоянии.
- Строится уточненная диаграмма классов.
- Проектирование методов классов: на основе диаграмм последовательностей строятся диаграммы деятельности (используется та же нотация, что и на этапе определения спецификаций).
5. Компоновка программных компонентов
- Выделяются физические части программного обеспечения (исполняемые файлы, файлы данных, таблицы баз данных и т.д.)
- Определяются связи между ними – зависимости.
- Строится диаграмма компонентов.
6. Проектирование размещения
- Определяется набор аппаратных средств, связанных с программной системой.
- Определяются соединения узлов – коммуникационные каналы.
- Строится диаграмма размещения.

6. Проектирование структур данных.

Проектирование структур данных – разработка представления абстрактных структур данных в памяти ЭВМ.

Основные учитываемые факторы:

- размерность структуры данных и степень ее «динамичности»;
- вид хранимой информации каждого элемента данных;
- связи элементов данных и вложенных структур;
- время хранения данных структуры («время жизни»);
- набор операций над элементами данных, вложенными структурами и структурами в целом.

Принципиальные решения, принимаемые в процессе проектирования:

1. Использование внутренней или внешней памяти.

Недостатки внешней памяти:

- большая трудоемкость операции доступа к блоку данных;
- простой метод последовательного доступа неэффективен для решения задачи поиска элемента данных.

Внешняя память используется:

- для постоянного хранения данных;
- для обработки данных только в случае невозможности или потенциальной опасности размещения большого объема данных во внутренней памяти.

Структуры данных для внешней памяти: хешированные файлы, индексированные файлы, B-деревья.

2. Способ распределения памяти.

Последовательное распределение – организация логической последовательности элементов данных на основе свойства физической смежности ячеек памяти.

Достоинства:

- возможен прямой доступ к элементу данных;
- простота реализации;
- меньшие дополнительные затраты памяти.

Недостатки:

- невозможность или логическая сложность динамического изменения размера структуры данных;
- временная сложность операций, связанных с изменением набора элементов структуры данных (вставка или удаление);
- логическая сложность представления и реализации нелинейных структур данных.

Примеры структур данных в последовательной памяти

Связное распределение – организация логической последовательности элементов данных посредством указателей.

Достоинства:

- отсутствие ограничений на размер структуры данных и простота его изменения;
- простота представления динамически меняющихся структур данных.
- простота представления нелинейных структур данных.

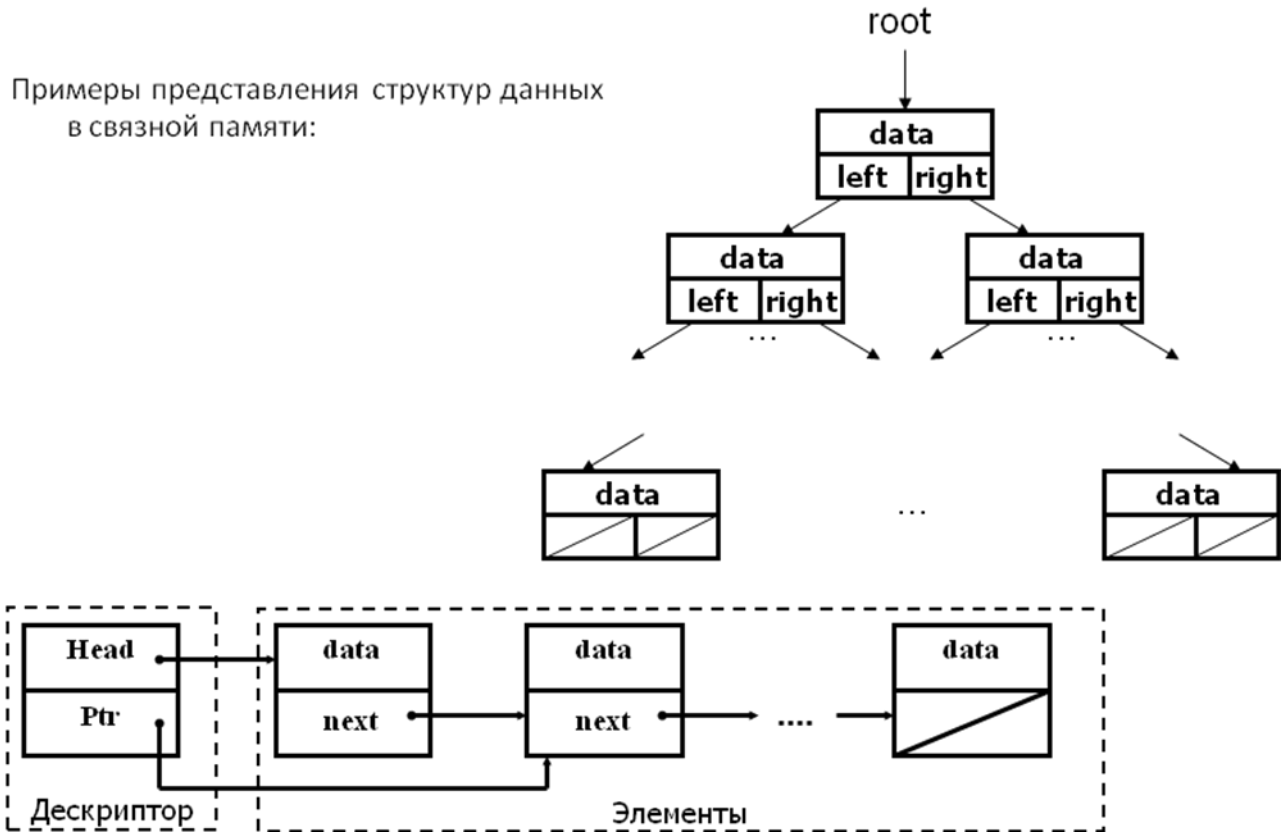
Недостатки:

- доступ к элементам строго последовательный (приводит к неэффективности по времени некоторых алгоритмов);
- дополнительные затраты памяти на указатели;
- большая сложность реализации и отладки программ.

Элемент структуры данных в общем виде состоит из двух полей:

- информационного поля или поля данных;
- поля указателей.

Примеры представления структур данных в связной памяти:

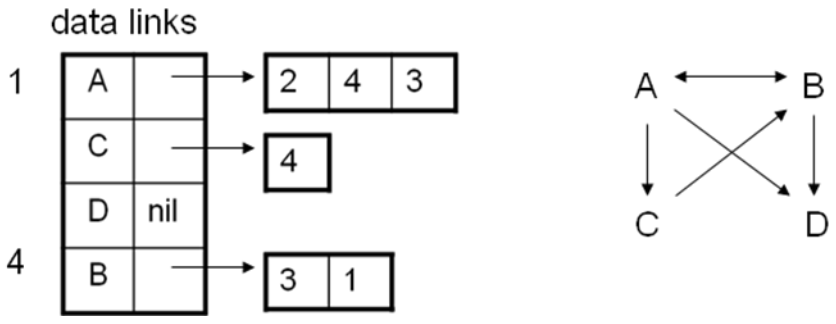


3. Способ выделения памяти (для структур данных в последовательной памяти).

1. Статическое выделение – при заранее предсказуемом размере данных и «времени жизни», сравнимом с временем выполнения программы (подпрограммы для локальных данных);

2. Динамическое выделение – при незначительном «времени жизни» или при значительном и непредсказуемом количестве операций включения/исключения элементов данных.

Пример: представление графа в последовательной памяти на основе динамических массивов:



7. Основные понятия и принципы тестирования ПО. Методика тестирования программных систем.

Основные понятия и принципы тестирования ПО.

Тестирование – процесс выполнения программы с целью обнаружения ошибок.

Каждый тест (тестовый вариант) определяет:

- набор исходных данных и условий для запуска программы;
- набор ожидаемых результатов работы программы.

Полную проверку программы гарантирует *исчерпывающее тестирование.*

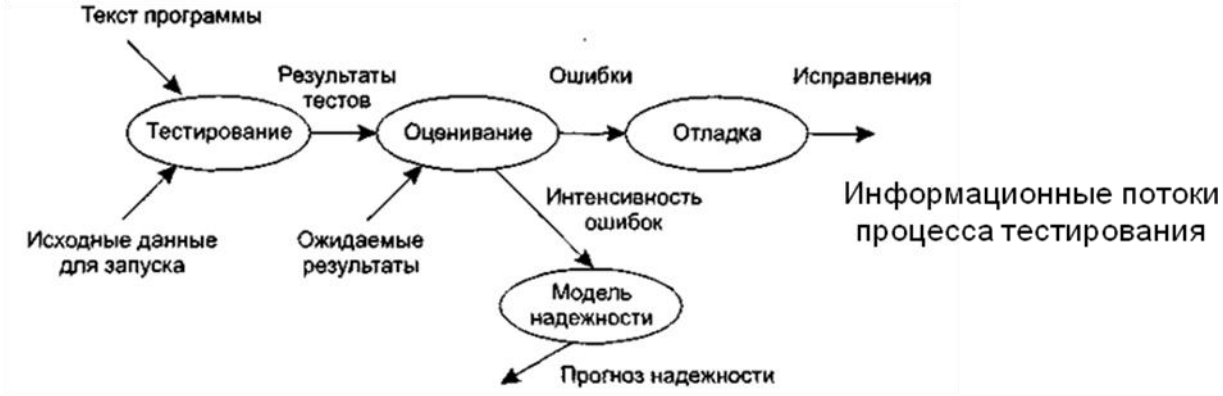
Целью проектирования тестовых вариантов является систематическое обнаружение различных классов ошибок при минимальных затратах времени и стоимости.

Тестирование обеспечивает:

- обнаружение ошибок;
- демонстрацию соответствия функций программы ее назначению;
- демонстрацию реализации требований к характеристикам программы;
- отображение надежности как индикатора качества программы.

Принципы тестирования программного обеспечения:

- функциональное тестирование (тестирование «черного ящика»);
- структурное тестирование (тестирование «белого ящика»).



Методика тестирования программных систем

Общий процесс тестирования объединяет различные способы тестирования в спланированную последовательность шагов, которые приводят к успешному построению программной системы.

8. Особенности структурного тестирования ПО. Способ тестирования базового пути.

Особенности структурного тестирования ПО.

Известно: внутренняя структура программы.

Исследуется: внутренние элементы программы и связи между ними.

Тестирование по принципу «белого ящика» характеризуется степенью, в какой тесты выполняют или покрывают логику (исходный текст) программы.

Обычно анализируются управляющие связи элементов, реже – информационные связи.

Общий принцип:

- программа представляется в виде графа (вершины – операторы, дуги – возможные варианты передачи управления);
- в соответствии с определенным способом формируется набор тестов, обеспечивающий покрытие маршрутов (последовательностей операторов).

Основные способы:

- способ тестирования базового пути (покрытие операторов);
- способы тестирования условий (покрытие условий, покрытие решений);
- способ тестирования потоков данных.

Недостатки:

1. Количество независимых маршрутов может быть очень велико.
2. Исчерпывающее тестирование маршрутов не гарантирует соответствия программы исходным требованиям к ней.
3. В программе могут быть пропущены некоторые маршруты.
4. Нельзя обнаружить ошибки, появление которых зависит от обрабатываемых данных.

Достоинства:

1. Учет особенностей программных ошибок.
2. Возможность логически простого тестирования отдельных модулей на ранних этапах реализации.

Способ тестирования базового пути.

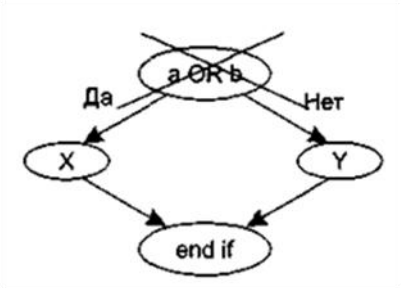
Шаги способа тестирования базового пути:

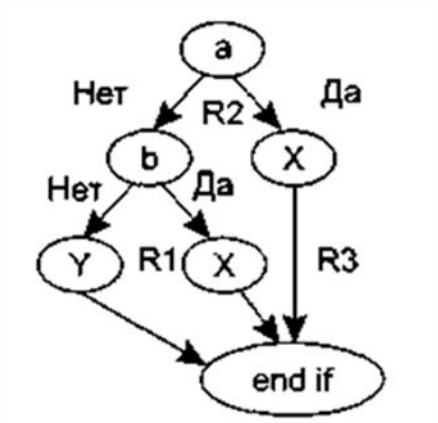
1. На основе текста программы формируется потоковый граф.
2. Определяется цикломатическая сложность потокового графа.
3. Определяется и выписывается базовое множество независимых линейных путей.
4. Подготавливаются тестовые варианты, инициирующие выполнение каждого пути.

Свойства потокового графа:

1. Граф строится отображением управляющей структуры программы. В ходе отображения закрывающие конструкции условных операторов и операторов циклов могут рассматриваться как отдельные (фиктивные) операторы.
2. Узлы (вершины) потокового графа соответствуют линейным участкам программы, включают один или несколько операторов программы.
3. Дуги потокового графа отображают поток управления в программе (передачи управления между операторами).
4. Различают операторные и предикатные узлы. Из операторного узла выходит одна дуга, а из предикатного — две дуги.
5. Предикатные узлы соответствуют простым условиям в программе. Составное условие программы отображается в несколько предикатных узлов.
6. Замкнутые области, образованные дугами и узлами, называют регионами. Окружающая граф среда рассматривается как дополнительный регион.

```
if a OR b
        then x
        else y
end if;
```





Цикломатическая сложность – метрика ПО, которая обеспечивает количественную оценку логической сложности программы.

В способе тестирования базового пути цикломатическая сложность определяет:

- количество независимых путей, составляющих базовое множество программы;
- верхнюю оценку количества тестов, которое гарантирует однократное выполнение всех операторов.

Независимым называется любой путь, который вводит новый оператор обработки или новое условие. В терминах потокового графа независимый путь должен содержать дугу, не входящую в ранее определенные пути.

Все независимые пути графа образуют *базовое множество*.

Свойства базового множества:

1. тесты, обеспечивающие его проверку, гарантируют:
   - однократное выполнение каждого оператора;
   - выполнение каждого условия по True-ветви и по False-ветви;
2. мощность базового множества равна цикломатической сложности потокового графа.

Способы определения цикломатической сложности:

1. цикломатическая сложность равна количеству регионов потокового графа;
2. определяется по формуле

$V(G)=E-N+2$,

где E — количество дуг, N — количество узлов потокового графа;

3. определяется по формуле

$V(G)=p+1$,

где p — количество предикатных узлов в потоковом графе G.

9. Способы тестирования условий. Способ тестирования потоков данных.

Способы тестирования условий.

Основная особенность – акцент на обнаружение ошибок в условиях (ошибка логического оператора; ошибка логической переменной; ошибка скобки; ошибка оператора отношения; ошибка арифметического выражения).

Шаги способа:

1. Строится ограничение условий, представляющее множество ограничений на результат каждого простого условия в исходном составном.
2. Строится ограничивающее множество, элементы которого являются сочетаниями простых условий.
3. Для каждого элемента ограничивающего множества разрабатывается тестовый вариант.

Тестирование ветвей и операторов отношений

1. **Ограничения условий (ограничения на результат)**

Вид условия	**Определение**	**Формальное определение**
Составное условие	– множество ограничений элементарных условий	$ОУ_c = (d_1, d_2, d_3 ..... d_n)$, где d_i – ограничение на результат i-го простого условия
Условие – логическая переменная	– множество допустимых значений логической переменной	d_i = (true, false)
Условие – выражение отношения	– множество допустимых значений выражения отношения	d_j = (>,<,=)

2. **Ограничивающее множество**

Элементами ограничивающего множества являются все возможные комбинации значений d_1, d_2, d_3, ..., d_n, обеспечивающие проверку всех простых условий.

Способ тестирования потоков данных

Объект анализа – информационная структура программы.

Цель – проверить маршруты, активно участвующие в формировании и использовании данных.

Шаги способа:

1. Построение управляющего графа программы.
2. Построение информационного графа.
3. Формирование полного набора DU-цепочек.
4. Формирование полного набора отрезков путей в управляющем графе отображением набора DU-цепочек информационного графа на управляющий граф.
5. Построение маршрутов – полных путей на управляющем графе, покрывающих набор отрезков путей управляющего графа.

1. Информационный граф

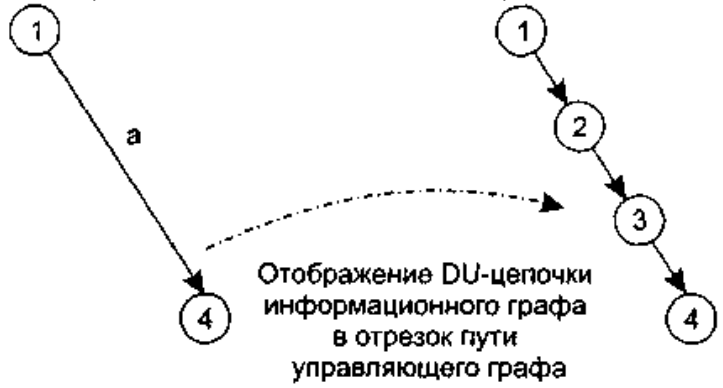
Информационный граф – потоковый граф, дополненный информационными дугами.

Информационная дуга исходит из узла, где определяются данные (конкретная переменная, структура данных), и входит в узел, где данные используются.

2. DU-цепочки

DU-цепочка определяется как [x, i,j], где i,j – имена вершин, причем переменная x определена в i-й вершине и используется в j -й вершине.

3. Отображение информационных дуг на управляющий граф



10.Особенности функционального тестирования ПО. Способ разбиения по эквивалентности.

Особенности функционального тестирования ПО.

Известно: входные данные и соответствующие выходные данные (ТЗ и спецификации).

Исследуется: поведение программы при решении возложенных на нее функциональных задач.

Составление тестов предполагает получение:

- наборов входных данных, которые приводят к аномалиям поведения программы (IT);
- наборов выходных данных, которые демонстрируют дефекты программы (OT).

Целевые категории ошибок:

- некорректные или отсутствующие функции;
- ошибки интерфейса;
- ошибки во внешних структурах данных или в доступе к базе данных;
- ошибки характеристик (необходимая емкость памяти и т. д.);
- ошибки инициализации и завершения.

Достоинства:

- сокращение необходимого количества тестовых вариантов;
- выявление классов ошибок, а не отдельных ошибок.

Недостатки:

- тестирование возможно только на завершающих стадиях реализации;
- трудность определения конкретного оператора (подпрограммы), приведшего к ошибке.

Способ разбиения по эквивалентности

Область данных программы делится на *классы эквивалентности*.

Класс эквивалентности – набор данных с общими свойствами (с точки зрения логики выполнения программы).

Класс эквивалентности включает множество значений данных, допустимых или недопустимых по условиям ввода.

Условие ввода может задавать:

1. Диапазон n...m:

- V_Class = {n…m};
- Inv_Class1 = {x | для любого x: x < n};
- Inv_Class2 = {y | для любого y: y > m}.

2. Конкретное значение a:

- V_Class = {a};
- Inv_Class1 = {x | для любого x: x < a};
- Inv_Class2 = { y | для любого y: y > a}.

3. Множество значений {a, b, c} :

- V_Class = {a, b, c};
- Inv_Class = {x | для любого x: (x≠a)&(x≠b)&(x≠c)}.

4. Логическое значение, например true:

- V_Class = {true};
- Inv_Class = {false}.

11.Способ анализа граничных значений. Способ диаграмм причин-следствий.

Способ анализа граничных значений.

Данный способ дополняет способ разбиения по эквивалентности.

Основные отличия анализа граничных значений от разбиения по эквивалентности:

- тестовые варианты создаются для проверки только ребер классов эквивалентности;
- при создании тестовых вариантов учитывают не только условия ввода, но и область вывода.

Основные правила анализа граничных значений:

1. Если условие ввода задает диапазон m...n, то тестовые варианты должны быть построены:
   - для значений m и n;
   - для значений чуть левее n и чуть правее m на числовой оси.
2. Если условие ввода задает дискретное множество значений, то создаются тестовые варианты:
   - для проверки минимального и максимального из значений;
   - для значений чуть меньше минимума и чуть больше максимума
3. Правила 1 и 2 также применяются и к условиям области вывода.
4. Если внутренние структуры данных программы имеют предписанные границы, то разрабатываются тестовые варианты, проверяющие эти структуры на их границах.
5. Если входные или выходные данные программы являются упорядоченными множествами (например, последовательным файлом, линейным списком, таблицей), то тестируется обработка первого и последнего элементов этих множеств.

Способ диаграмм причин-следствий.

Способ обеспечивает формальное выведение высокорезультативных тестовых вариантов, основанное на анализе причинно-следственных связей.

Причина – отдельное входное условие или класс эквивалентности.

Следствие – выходное условие или действие системы.

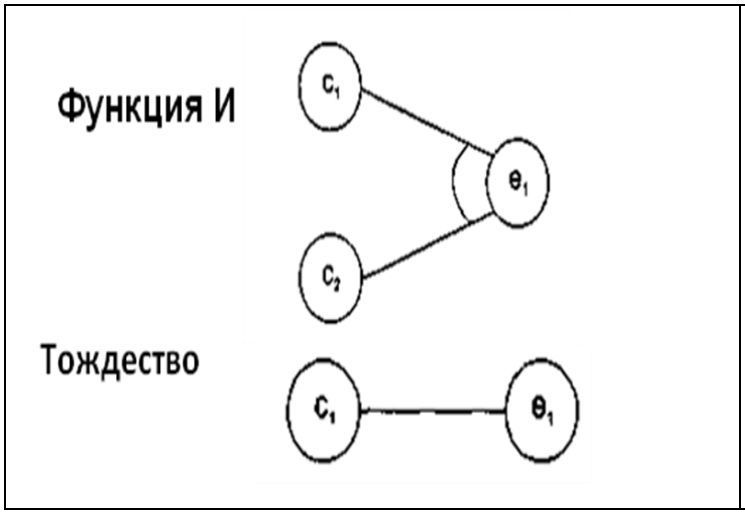
Дополнительный эффект – обнаружение неполноты или неоднозначности спецификаций.

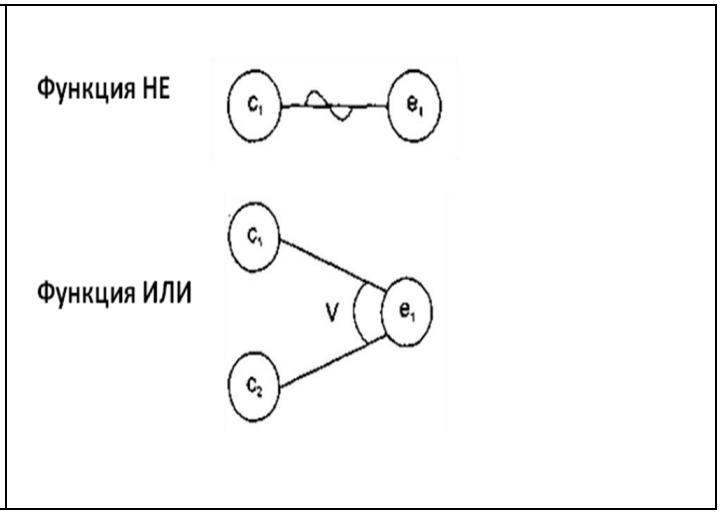
Шаги способа:

1. Разбиение спецификаций на «рабочие» участи и выделение групп причин и следствий.
2. В каждой группе выделяются причины и следствия, им присваиваются идентификаторы.
3. Разрабатывается граф причинно-следственных связей.
4. Граф преобразуется в таблицу решений.
5. Столбцы таблицы решений преобразуются в тестовые варианты.

Граф причин следствий

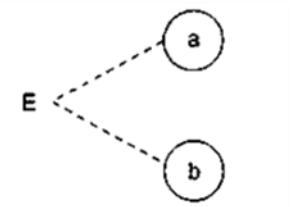
- Обозначения узлов: c_i – причина, e_i – следствие.
- Узел может находится в двух состояниях: 0 и 1.
- Базовые символы:





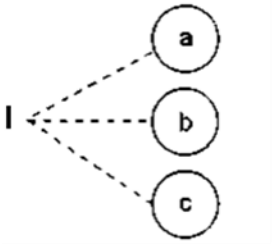
Ограничение E (Exclusive - исключение):

только одна из величин может принимать значение 1



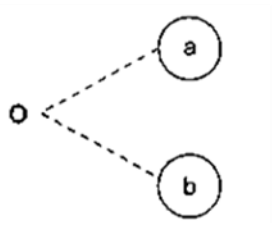
Ограничение I (Inclusive - включение):

по крайней мере одна из величин должна быть равной 1



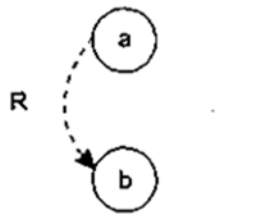
Ограничение O (Only one - только одно):

одна и только одна из величин должна быть равна 1



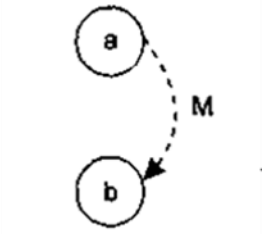
Ограничение R (Requires - требование):

если a=1, то b должно принимать значение 1



Ограничение M (Masks - скрытие):

если следствие *a*=1, то следствие *b* должно принять значение 0



Генерация таблицы решений.

1. Выбирается некоторое следствие, которое должно быть в состоянии «1».
2. Находятся все комбинации причин (с учетом ограничений), которые устанавливают это следствие в состояние «1». Для этого из следствия прокладывается обратная трасса через граф.
3. Для каждой комбинации причин, приводящих следствие в состояние «1», строится один столбец.
4. Для каждой комбинации причин доопределяются состояния всех других следствий. Они помещаются в тот же столбец таблицы решений.
5. Действия 1-4 повторяются для всех следствий графа.

12.Тестирование элементов, интеграции, правильности, системное тестирование.

Тестирование элементов

Объект тестирования – наименьшая единица проектирования ПС – модуль.

Цель – индивидуальная проверка каждого модуля.

Основной способ тестирования – структурное.

Последовательно подвергаются тестированию:

- интерфейс модуля (ошибки ввода-вывода);
- внутренние структуры данных (целостность данных);
- независимые пути (ошибочные вычисления, некорректные сравнения, неправильный поток управления);
- пути обработки ошибок (некорректность работы операторов обработки ошибок);
- граничные условия (анализ граничных значений).

Тестирование интеграции

Объект тестирования – сборка программной системы.

Цель – проверка корректности сборки модулей в программную систему.

Основной способ тестирования – функциональное.

Конкретный объект тестирования – межмодульные интерфейсы.

Основные ошибки:

- потеря данных при прохождении интерфейса;
- отсутствие необходимого вызова;
- влияние одного модуля на другой (недопустимое сцепление);
- композиция подфункций не дает нужную функцию;
- некорректная работа с глобальными данными.

Способы тестирования интеграции:

1. Нисходящее тестирование.

Модули объединяются движением сверху вниз по управляющей иерархии, начиная от главного управляющего модуля. Выбор модулей осуществляется по принципу обхода в глубину или в ширину (по уровням).

Общая методика тестирования интеграции нисходящим способом:

1. Главный управляющий модуль используется как тестовый драйвер. Все непосредственно подчиненные ему модули временно замещаются заглушками.
2. Одна из заглушек драйвера заменяется реальным модулем.
3. На этом модуле устанавливаются заглушки и проводится набор тестов, проверяющих полученную структуру.
4. Если в модуле-драйвере уже нет заглушек, производится смена модуля-драйвера.
5. Если есть неподключенные модули, выполняется возврат на шаг 2.

Достоинство – ошибки в главной, управляющей части выявляются в первую очередь.

Недостаток – трудности в ситуациях, когда для полного тестирования на верхних уровнях нужны результаты обработки с нижних уровней иерархии.

Способы устранения недостатков:

1. Откладывать некоторые тесты до замещения заглушек модулями.
2. Разработка более сложных заглушек, частично выполняющие функции модулей.
3. Использовать восходящее тестирование интеграции

2. Восходящее тестирование.

Модули подключаются движением снизу вверх. Подчиненные модули всегда доступны, и нет необходимости в заглушках.

Общая методика тестирования интеграции восходящим способом:

1. Модули нижнего уровня объединяются в кластеры, выполняющие определенную программную подфункцию.
2. Для координации вводов-выводов тестового варианта реализуется драйвер, управляющий тестированием кластеров.
3. Тестируется кластер.
4. Драйверы удаляются, а кластеры объединяются в структуру движением вверх, осуществляется тестирование частичной сборки до объединения всех модулей.

Недостаток – система не существует как целое до тех пор, пока не будет добавлен последний модуль.

Достоинство – упрощается разработка тестовых вариантов, отсутствуют заглушки.

3. Комбинированный подход:

- для верхних уровней иерархии применяют нисходящую стратегию;

- для нижних уровней – восходящую стратегию тестирования;
- выделяются критические модули, они должны тестироваться как можно раньше, тестироваться не один раз.

Признаки критического модуля:

- реализует несколько требований к программной системе;
- имеет высокий уровень управления;
- имеет высокую сложность или склонность к ошибкам (как индикатор может использоваться цикломатическая сложность – больше 10).

Тестирование правильности

Объект тестирования – единая программная система.

Цель тестирования – проверка и подтверждение реализации в ПС функциональных требований заказчика.

Способ тестирования – функциональное.

Тестирование правильности включает также тестирование конфигурации ПС, т.е. совокупности всех элементов информации, полученных в процессе разработки ПС. Базовые элементы:

- план программного проекта;
- спецификация требований к ПС;
- листинги исходных текстов программ;
- план и методика тестирования; тестовые варианты и полученные результаты;
- руководства по работе и инсталляции;
- описание базы данных;
- руководство пользователя;
- и др.

В ходе тестирования правильности реализуются альфа- и бета-тестирование.

Системное тестирование

Объект тестирования – система в целом.

Цель тестирования – проверка правильности объединения и взаимодействия всех элементов компьютерной системы, реализации всех системных функций.

Способ тестирования – Функциональное.

Системное тестирование включает в себя:

1. Тестирование восстановления (время восстановления, правильность повторной инициализации, восстановление данных и др.)
2. Тестирование безопасности (проверка функционирования защитных механизмов системы в случае проникновения).
3. Стрессовое тестирование (тестовые варианты направлены на оценку стабильности работы ПС при «ненормальных» нагрузках: по частоте запросу, объему данных). Частный случай – тестирование чувствительности.
4. Тестирование производительности (проверяется скорость работы ПО в компьютерной системе).

13.Регрессионное тестирование.

Регрессионное тестирование – тестирование функциональности программного обеспечения, после внесения изменений на фазе системного тестирования или сопровождения продукта. Регрессионное тестирование можно выполняться вручную или средствами автоматизации тестирования.

Преимущества регрессионного тестирования

- Сокращение количества дефектов в системе к моменту релиза;
- Исключение деградации качества системы при росте функциональности;
- Уменьшение вероятности критических ошибок при эксплуатации.

Задачи регрессионного тестирования

Основная задача регрессионного тестирования — проверка того, что исправление ошибки не коснулось существующей функциональности. Из-за частого выполнения одних и тех же наборов сценариев, рекомендуется использовать автоматизированные регрессионные тесты, что позволит сократить сроки тестирования.

- проверка и утверждение исправления ошибки;
- тестирование последствия исправлений, так как внесенные исправления могут привнести ошибку в код который исправно работал;
- гарантировать функциональную преемственность и совместимость новой версии с предыдущими;
- уменьшение стоимости и сокращение времени выполнения тестов.

Для регрессионного тестирования используются тест кейсы, написанные на ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Рекомендуется проводить автоматизацию регрессионных тестов, для ускорения последующего процесса тестирования и обнаружения дефектов на ранних стадиях разработки программного обеспечения.

Проводить регрессивное тестирование, следует после любого изменения функционала, для того, чтобы убедиться в отсутствии новых и/или устранении предыдущих ошибок. Включение блочного регрессивного тестирования в процесс разработки позволяет защититься от ошибок.

Регрессия необходима практически всегда, когда нет выстроенного процесса разработки:

- ошибки в архитектуре приложения;
- слабая аналитика по влиянию изменения требований на другие модули программы;
- высокая связность модулей и кода;
- нет интеграционных автоматических тестов;
- тестировщики не общаются с разработчиками и не обсуждают риски;
- тестировщики плохо представляют архитектуру продукта и внутренние взаимосвязи.

Считается хорошей практикой при исправлении ошибки создать тест на неё и регулярно прогонять его при последующих изменениях программы. Хотя регрессионное тестирование может быть выполнено и вручную, но чаще всего это делается с помощью специализированных программ, позволяющих выполнять все регрессионные тесты автоматически. В некоторых проектах даже используются инструменты для автоматического прогона регрессионных тестов через заданный интервал времени.

14.Общие понятия и методы отладки ПО.

Отладка – это локализация и устранение ошибок. Она следует за успешным тестированием программного обеспечения.

Основной сложностью в процессе отладки является именно обнаружение оператора, содержащего ошибку.

Способы отслеживания ошибки – аналитические и экспериментальные.

Классификация ошибок по принадлежности к этапу обработки программы

Способы проявления ошибок выполнения:

- появление системного сообщения об ошибке низкого уровня;
- появление системного сообщения об ошибке;
- «зависание» компьютера;
- несовпадение полученных результатов с ожидаемыми.

Методы отладки программного обеспечения

1. Метод ручного тестирования.

2. Метод индукции.

Основан на тщательном анализе симптомов ошибки. Информацию организуют и тщательно изучают. Выдвигают гипотезы об ошибках, каждую из которых проверяют.

3. Метод дедукции.

Формируют множество возможных причин. Анализируя причины, исключают невозможные. Наиболее вероятную гипотезу пытаются доказать.

4. Метод обратного прослеживания.

Эффективен для небольших программ. Для точки вывода некорректного результата строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Делают предложения о значениях переменных в предыдущей точке и т.д. до обнаружения причины ошибки.

15.Общая методика отладки программного обеспечения. Методы и средства получения дополнительной информации при отладке.

Общая методика отладки программного обеспечения

1 этап – изучение проявления ошибки (используются индуктивные и дедуктивные методы). Выдвигаются и проверяются версии (могут примениться методы и средства получения дополнительной информации).

2 этап – локализация ошибки (путем отсечения частей программы или с использованием отладочных средств).

3 этап – (если ошибка не в том месте, где она проявилась) определение причины ошибки - изучение результатов второго этапа и формирование версий возможных причин ошибки.

4 этап – исправление ошибки.

5 этап - повторное тестирование.

Основные рекомендации структурного подхода:

- программу наращивать «сверху-вниз», от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;
- выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;
- предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

Дополнительно: фрагменты программного обеспечения, где уже были обнаружены ошибки, следует проверять тщательнее.

Методы и средства получения дополнительной информации

1. Отладочный вывод

Основная идея: включение дополнительных операторов вывода в узловых точках.

В настоящее время используется редко. Недостаток: обычно большой объем вывода.

2. Интегрированные средства отладки

Основные функции:

- пошаговое выполнение программы;
- поддержка точек останова;
- выполнение программы «до курсора»;
- отображение содержимого переменных при пошаговом выполнении;
- отслеживание потока сообщений;
- и т.п.

3. Использование независимых отладчиков

16.Единая система программной документации: общие положения и виды документов.

Единая система программной документации (ЕСПД) – комплекс государственных стандартов, устанавливающих взаимоувязанные правила разработки, оформления и обращения программ и программной документации.

В стандартах ЕСПД устанавливаются требования, регламентирующие разработку, сопровождение, изготовление и эксплуатацию программ.

ЕСПД обеспечивает:

- унификацию программных изделий для взаимного обмена программами и применение ранее разработанных программ в новых разработках;
- снижение трудоемкости и повышение эффективности разработки, сопровождения, изготовления и эксплуатации программных изделий;
- автоматизацию изготовления и хранения программной документации.

Область распространения – программы и программная документация для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Состав ЕСПД:

- основополагающие и организационно-методические стандарты;
- стандарты, определяющие формы и содержание программных документов, применяемых при обработке данных;
- стандарты, обеспечивающие автоматизацию разработки программных документов.

Правила обозначения стандартов:

ГОСТ 19.GNN-YY

ГОСТ – категория стандарта – государственный;

19 – класс стандартов ЕСПД;

G – цифра – код классификационной группы;

NN – две цифры – порядковый номер стандарта в группе;

YY – год регистрации стандарта.

Классификационные группы:

0 – Общие положения

1 – Основополагающие стандарты

2 – Правила выполнения документации разработки

3 – Правила выполнения документации изготовления

4 – Правила выполнения документации сопровождения

5 – Правила выполнения эксплуатационной документации

6 – Правила обращения программной документации

7,8 – Резервные группы

9 – Прочие стандарты

Виды программных документов и их содержание устанавливаются ГОСТ 19.101-77.

Основные виды документов:

Спецификация (обязательна для программных изделий, имеющих самостоятельное назначение).

Текст программы (обязателен).

Описание программы.

Описание применения.

Руководство программиста.

Руководство оператора.

Пояснительная записка должна содержать всю информацию, необходимую для сопровождения (модификации) программного обеспечения.

Содержание пояснительной записки по ГОСТ 19.404-79:

1. Введение.
2. Назначение и область применения.
3. Технические характеристики.

- постановка задачи, описание применяемых математических методов, допущений и ограничений;
- описание алгоритмов и функционирования программы с обоснованием принятых решений;
- описание и обоснование выбора способа организации входных и выходных данных;
- описание и обоснование выбора состава технических и программных средств на основании проведенных расчетов или анализов.

4. Ожидаемые технико-экономические показатели.
5. Источники, используемые при разработке.

Руководство пользователя – более современный эксплуатационный документ, включающий отчасти требования руководств системного программиста, программиста и оператора.

Приблизительный состав разделов руководства пользователя:

1. Общие сведения о программном продукте.
2. Описание установки.
3. Описание запуска.
4. Инструкции по работе (или описание пользовательского интерфейса).
5. Сообщения пользователю.

Руководство системного программиста должно содержать всю информацию, необходимую для установки программного обеспечения, его настройки и проверки работоспособности.

Содержание руководства системного программиста по ГОСТ 19.503-79:

1. Общие сведения о программном продукте.
2. Структура программы.
3. Настройка программы.
4. Проверка программы.
5. Дополнительные возможности.
6. Сообщения системному программисту.