[**1.** **Основные понятия надежности аппаратного обеспечения компьютерных систем. (надежность, безотказность, отказ, наработка до отказа, сбой, работоспособное состояние, устойчивость, достоверность).**](#_Toc35467796)

[**2.** **Показатели надежности невосстанавливаемых объектов. (P(t), Q(t), f(t)).**](#_Toc35467797)

[**3.** **Показатели надежности невосстанавливаемых объектов (λ(t), m(t), D(t), σ(t)).**](#_Toc35467798)

[**4.** **Показатели надежности восстанавливаемых объектов (Kr(t), Kr ст.). Экспоненциальный закон распределения времени до отказа.**](#_Toc35467799)

[**5.** **Общая схема расчета надежности компьютерных систем. Расчет надежности невосстанавливаемых резервируемых и не резервируемых компьютерных систем. Пример.**](#_Toc35467800)

[**6.** **Связь эффективности и надежности компьютерных систем.**](#_Toc35467801)

[7. **Понятие требований к системам и ПО. Характеристики для оценки отдельных требований по ISO/IEC/IEEE 29148:2011.**](#_Toc35467802)

[**8.** **Характеристики для оценки наборов требований по ISO/IEC/IEEE 29148:2011. Атрибуты требований.**](#_Toc35467803)

[**9.** **Понятие ошибки и отказа ПО. Понятие надежности ПО. Особенности надежности программ по сравнению с надежностью аппаратуры.**](#_Toc35467804)

[**10.** **Процесс разработки ПС в соответствии с СТБ ИСО/МЭК 12207-2003. Причины появления ошибок в ПО.**](#_Toc35467805)

[**11.** **Методы обеспечения надежности ПО.**](#_Toc35467806)

[**12.** **Модель Джелинского-Моранды. Определение характеристик модели с помощью метода максимального правдоподобия.**](#_Toc35467807)

[**13.** **Модель Шика-Волвертона. Определение характеристик модели с помощью метода максимального правдоподобия.**](#_Toc35467808)

[**14.** **Геометрическая модель надежности ПО. Определение характеристик модели с помощью метода максимального правдоподобия.**](#_Toc35467809)

[**15.** **Модель Миллса.**](#_Toc35467810)

[**16.** **Модель надежности ПС в соответствии с ГОСТ 28195-99.**](#_Toc35467811)

[**17.** **Оценка надежности ПС в соответствии с ГОСТ 28195-99.**](#_Toc35467812)

[**18.** **Модель надежности ПС в соответствии с СТБ ИСО/МЭК 9126-2003.**](#_Toc35467813)

[**19.** **Модель надежности ПС в соответствии с ISO/IEC 25010-2011.**](#_Toc35467814)

[**20.** **Процесс оценки надежности ПС в соответствии с СТБ ИСО/МЭК 9126-2003.**](#_Toc35467815)

[**21.** **Внутренние метрики надежности ПС.**](#_Toc35467816)

[**22.** **Внешние метрики надежности ПС.**](#_Toc35467817)

[**23.** **Введение в тестирование ПО. Понятие тестирования ПО. Что может и не может тестирование ПО. Объекты тестирования ПО.**](#_Toc35467818)

[**24.** **Виды тестирования ПО.**](#_Toc35467819)

[**25.** **Принципы разработки тестов.**](#_Toc35467820)

[**26.** **Информационные потоки процесса тестирования ПО. Аксиомы тестирования ПО.**](#_Toc35467821)

[**27.** **Структурное тестирование ПО. Понятие потокового графа, пути, базового пути, ветви, цикломатической сложности.**](#_Toc35467822)

[**28.** **Метод тестирования базовых путей. Тестирование циклов.**](#_Toc35467823)

[**29.** **Функциональное тестирование ПО. Общие сведения. Цели функционального тестирования. Уровни функционального тестирования.**](#_Toc35467824)

[**30.** **Понятие класса эквивалентности. Примеры.**](#_Toc35467825)

[**31.** **Метод эквивалентного разбиения. Метод анализа граничных значений. Примеры.**](#_Toc35467826)

[**32.** **Документирование тестов. Структура Test Case. Примеры.**](#_Toc35467827)

[**33.** **Методика тестирования ПО. Тестирование модулей.**](#_Toc35467828)

[**34.** **Тестирование сборки (интеграции).**](#_Toc35467829)

[**35.** **Тестирование правильности. Системное тестирование. Типы системного тестирования.**](#_Toc35467830)

[**36.** **Регрессионное тестирование.**](#_Toc35467831)

[**37.** **Документирование ошибок. Структура отчета об ошибке.**](#_Toc35467832)

[**38.** **Жизненный цикл дефекта.**](#_Toc35467833)

[**39.** **Проектирование тестов, основанных на случайных наборах исходных данных. Получение закона распределения по заданным моментам исходного распределения.**](#_Toc35467834)

[**40.** **Проектирование тестов, основанных на случайных наборах исходных данных. Генерация случайных значений исходных данных.**](#_Toc35467835)

[**41.** **Понятие верификации. Понятие высказывания, высказывательной формы и предиката. Операция импликации. Примеры.**](#_Toc35467836)

[**42.** **Метод индуктивных утверждений. Общие сведения.**](#_Toc35467837)

[**43.** **Процесс верификации в жизненном цикле ПО.**](#_Toc35467838)

[**44.** **N-версионное программирование.**](#_Toc35467839)

[**45.** **Отказоустойчивость компьютерных систем. Граф процесса восстановления типичной отказоустойчивой сист.**](#_Toc35467840)

[**46.** **Оптимальное распределение ресурсов в отказоустойчивых компьютерных системах.**](#_Toc35467841)

### **Основные понятия надежности аппаратного обеспечения компьютерных систем. (надежность, безотказность, отказ, наработка до отказа, сбой, работоспособное состояние, устойчивость, достоверность).**

**Надежность** – св-во технического объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования.

**Безотказность** – св-во объекта непрерывно сохранять работоспособное состояние в течение некоторого времени или наработки.

**Отказ** – событие, заключающееся в нарушении работоспособного состояния объект. Отказ почти всегда вызывается физическим разрушением объекта (полностью или частично).

**Наработка до отказа** – объем работы объекта от начала эксплуатации до возникновения отказа.

**Сбой** – самоустраняющийся отказ или однократный отказ, устраняемый незначительным вмешательством оператора.

**Работоспособное состояние** – это состояние объекта, при котором значение всех параметров, характеризующих способность выполнять заданные функции, соответствуют требованиям нормативно – технической и (или) конструкторской (проектной) документации.

**Устойчивость (живучесть)** – св-во объекта сохранять работоспособность полностью или частично в условиях неблагоприятных воздействий, не предусмотренных нормальными условиями эксплуатации.

**Достоверность информации выдаваемой объектом** – св-во объекта производить безошибочно преобразование, хранение и передачу информации.

### **Показатели надежности невосстанавливаемых объектов. (P(t), Q(t), f(t)).**

**Вероятность безотказной работы** ***P(t)*** - вероятность того, что невосстанавливаемый объект не откажет к моменту времени ***t***, т.е. в течение заданной наработки ***(0, t).***

***P(t) = P {T>= t },***

где ***T*** – случайная величина наработки до отказа.

Свойства показателя ***P(t)***:

1) ***P (0) =1***;

2)  ***.***

**Вероятность отказа объекта** ***Q(t)*** – вероятность отказа в течение заданной наработки ***(0, t).***

***Q(t) = 1 – P(t) = P {T < t },***

где: ***Q(t)*** – вероятность того, что наработка до отказа ***Т*** меньше заданной наработки ***t*.** Фактически ***Q(t)***представляет собой функцию распределения СВ ***Т***.

**Функция плотности распределения** ***f(x)*** времени до отказа ***–*** представляет собой безусловную плотность вероятности возникновения отказа.

***f(t)=P(T<t).***

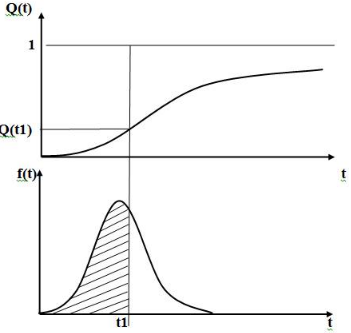


Рис. Функции распределения случайной величины и плотности распределения наработки до отказа

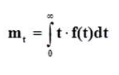
### **Показатели надежности невосстанавливаемых объектов (λ(t), m(t), D(t), σ(t)).**

**Интенсивность отказов** **λ(t)** – условная плотность вероятности возникновения отказа объекта, определяемая при условии, что до рассматриваемого момента времени отказ не возник:

.

**Статистическое определение λ(t)***–* число отказавших объектов за интервал времени ***∆t***, отнесённое к числу не отказавших объектов к моменту времени ***t*** за период ***∆t***.

**Средняя наработка до отказа m(t)** – математическое ожидание времени до отказа (фактически это среднее время до первого отказа).



*Средняя наработка* численно равна площади под кривой ***P(t)****.*

Самый естественный показатель надёжности, но он ничего не говорит о характере распределения времени до отказа.

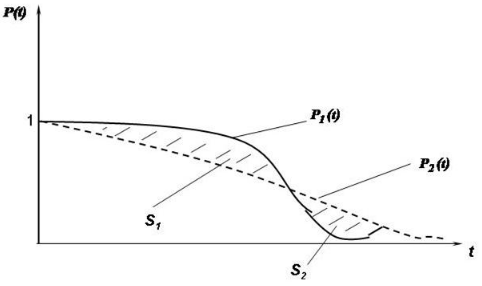


Рис. Виды функций безотказной работы(

**Среднеквадратическое отклонение наработки до отказа σ(t).**



**Дисперсия** ***D(t)*** – характеризует величину разброса наработки относительно среднего значения.

### **Показатели надежности восстанавливаемых объектов (Kr(t), Kr ст.). Экспоненциальный закон распределения времени до отказа.**

**Коэффициент (функция) готовности KГ(t)** – вероятность того, что объект окажется в работоспособном состоянии в произвольный момент времени, кроме планируемых периодов, в течение которых применение объекта по назначению не предусматривается.

Статистически:



где:

**N0** –**NB(t)** **–** количество объектов, находящихся в момент **t** в состоянии восстановления (ремонта);

**NB(t)** **–** число объектов, находящихся в рабочем состоянии в момент времени **t;**

**N0 –** общее число объектов.

*Коэффициент (функция) готовности KГ(t)* используется в том случае, когда кроме факта работоспособности необходимо учитывать и время восстановления объекта.

**Стационарный коэффициент готовности**



где: **tp** – время работы объекта; **tв** – время восстановления объекта.

**КГ(t)** и **КГ.СТ**. — комплексные показатели, учитывающие св-во безотказности и восстанавливаемости.



Рис. Зависимость коэффициента готовности от времени

Время между соседними отказами для аппаратуры компьютера является непрерывной СВ, которая характеризуется некоторым законом распределения. Зависимость надёжности от времени описывается с помощью математической модели надёжности (ММН), то есть математического выражения (формулы, уравнения, система уравнений, алгоритма), позволяющего определять показатели надёжности ММН в виде формул с эмпирическими коэффициентами. ММН в виде формул с эмпирическими коэффициентами носят название статистических моделей распределения. Наиболее распространённой статистической моделью надёжности является **экспоненциальная модель** распределения времени до отказа, по которой вероятность безотказной работы определяется выражением:



где **λ** — параметр модели.

Функция плотности:



Интенсивность отказов:



### **Общая схема расчета надежности компьютерных систем. Расчет надежности невосстанавливаемых резервируемых и не резервируемых компьютерных систем. Пример.**

Для расчета надежности необходимо иметь модель надежности сист., которая составляется на основе функциональной схемы сист.

При составлении модели надежности сист. функциональные связи между элементами сист. заменяются логическими, характеризующими безотказную работу сист. в зависимости от работоспособности или неработоспособности элементов.

Элемент, при отказе которого отказывает вся система, считается последовательно соединенным в модели надежности. В этом случае безотказная работа сист. имеет место при сохранении работоспособности всеми элементами последовательного соединения. Отказ сист. наступает при отказе хотя бы одного элемента.

Элемент, отказ которого не приводит к отказу всей сист., считается включенным параллельно в модели надежности. Безотказная работа всей сист. при параллельном соединении элементов имеет место при сохранении работоспособности хотя бы одного элемента.

Способ вычисления показателей надежности существенно зависит от типа соединения отдельных конструктивных элементов и их взаимодействия между собой с точки зрения обеспечения безотказной работы сист.

*1) Нерезервированные сист.:*

При этом отказ всей сист. происходит при отказе любого элемента.



Рис. Модель надежности последовательного соединения элементов

Надежность такой сист. определяется по теореме умножения вероятностей для независимых событий и равна произведению вероятностей безотказной работы отдельных элементов Pi(tj) в течение наработки (0, tj).

Тогда:



где **n** – количество последовательно соединенных элементов; **Pi(tj)** – вероятность безотказной работы i-го последовательно соединённого элемента.

*2) Резервированные сист.*

Отказ такой сист. происходит только в том случае, когда откажут все элементы сист.

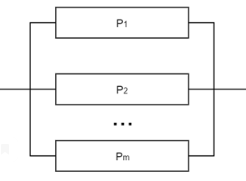


Рис. 2.10. Модель надежности параллельного соединения m элементов.

Вероятность безотказной работы равна:



где **Pi(tj)** – вероятность безотказной работы i-го параллельно соединенного элемента сист.; **m** – количество параллельно соединенных элементов.

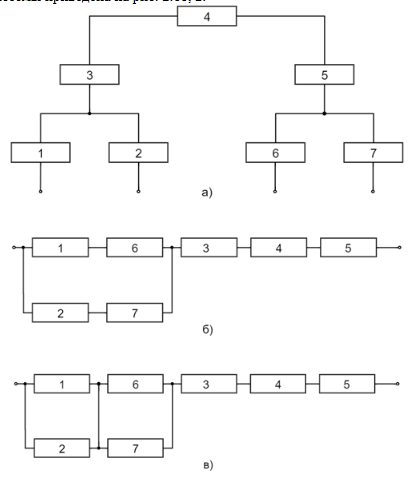


Рис. 2.11. Схемы сист. связи с использованием ИСЗ: а – схема сист. с ИСЗ; б – модель надежности при частотном разделении каналов;

в – модель надежности при временном разделении каналов; 1, 2- передатчики телевизионных программ; 3, 5 – антенные устройства; 4 – ИСЗ;

6, 7 - приемники телевизионных программ.

### **Связь эффективности и надежности компьютерных систем.**

**Эффективностью сист.** - св-во сист. выдавать некоторый полезный результат (эффект) при ее использовании в соответствии с назначением.

Эффективность сист. зависит:

* надежности;
* производительности;
* защищенности;
* точности;
* стоимости;
* и т.д.

Чем выше надежность сист., тем выше и её эффективность, но лишь до некоторого предела.

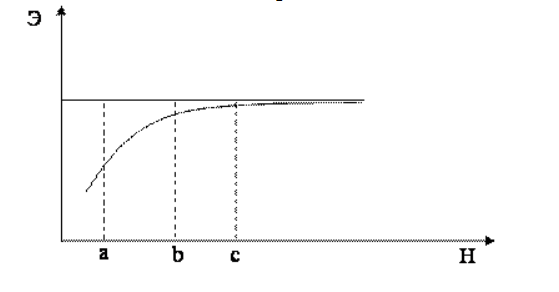


Рис. 2.12. Графическое представление связи между эффективностью и надежностью

Различают различные виды эффективности объектов (в том числе систем):

· *Эффективность номинальная (идеальная)* – эффективность объекта при безотказном его состоянии.

· *Эффективность реальная* – эффективность реального объекта, не обладающего идеальной надежностью.

· *Эффективность техническая* – технический эффект, полученный при использовании объекта (количество переданной информации, снижение затрат времени и т.д.);

· *Эффективность экономическая* – степень выгодности экономических затрат при использовании объекта.

Показатели надежности сист, или элементов сист. учитываются при вычислении значений показателя эффективностью. В общем, удается отследить снижение эффективности из-за её недостаточной надежности, при этом вычисляется идеальное значение эффективности (**Эи**) при абсолютной надежности сист, и некоторое «реальное» значение (**Эр**), учитывающее фактическую надежность сист.

Снижение эффективности из-за недостаточной надежности сист. характеризуют следующие выражения:

**∆Э = Эи – Эр;**

При оценке эффективности целесообразно выделять дискретные состояния, в которых может находиться система. Тогда значение показателя эффективности вычисляется по формуле, схожей с формулой математического ожидания:

**Э = ∑(Эj ∙ Рj) ,j ∈ G**

где:

**G** – множество состояний сист.;

**Эj** – условный показатель эффективности при нахождении сист. в j-ом состоянии;

**Pj** – вероятность пребывания сист. в j-ом состоянии.

### **Понятие требований к системам и ПО. Характеристики для оценки отдельных требований по ISO/IEC/IEEE 29148:2011.**

**Спецификация** – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание условий и эффекта действий без указания способа их достижения.

Спецификация требований к ПО не должна содержать следующих требований:

* К процессу разработки;
* К проекту.

Одна из главных причин низкого качества, разрабатываемого ПО – недостаточная детализация спецификации и ее низкая формализация. В общем случае ПО создается на основании требований из собственного видения продукта заказчиком. Очевидной причиной возможных ошибок, заложенных в спецификации, является недостаточная работа разработчика с заказчиком.

Параметры определений спецификации требований:

**Необходимость** - требование определяет существенную способность, характеристику, ограничения или показатель качества.

**Свобода реализаций** - требование должно быть независимым, от его реализации и оно определяет, что требуется, но не как это требование должно быть реализовано.

**Однозначность** - требование устанавливается таким образом, чтобы оно могло быть интерпретировано только одним способом и сформулировано просто и понятно. Его могут однозначно интерпретировать разработчик и заказчик.

**Непротиворечивость требований** - спецификация требований к ПО непротиворечива тогда и только тогда, когда каждое заявленное требование не противоречит ни одному другому требованию.

**Полнота** - требование определено достаточно полно, если оно не нуждается в дальнейшем пояснении.

**Единичность (атомарность)** - сформулированное требование должно включать только 1 требование.

**Реализуемость** - требование реализуемо, если оно с применимым уровнем риска технически достижимо в условиях существующих систем ограничений (например, ограничения по стоимости и срокам разработки).

**Трассируемость** - возможность определить связь между данным требованием, требованиями вышестоящих в иерархии и требованиями/потребностями конкретных заданных прав-дателей и др источников/артефактов (документов, модулей и т.д.). Например, должен быть обеспечен легкий доступ к каждому требованию из документации, создаваемой в процессе разработки продукта. Требование также должно позволить отследить связь между ним и требованиями в спецификации более низких уровней и другими результатами разработки. Для требований должны быть идентифицированы все родительско-дочерние связи так, чтобы можно было отобразить требования от источника до его реализации.  
**Проверяемость (верефицируемость)**- требование проверяемо, если имеется возможность проверки реализации данного требования в сист. или ПС.

### **Характеристики для оценки наборов требований по ISO/IEC/IEEE 29148:2011. Атрибуты требований.**

***Полнота набора требований -***спецификация требований к ПОявляется полной, когда она удовлетворяет следующим критериям.

1. Содержит все существенные требования, касающиеся функциональных возможностей, выполнения, ограничений проектирования, характеристик или внешних интерфейсов. В частности, любые внешние требования, накладываемые спецификацией сист., должны быть выявлены и рассмотрены.
2. Содержит все реакции программного обеспечения на все возможные типы входных данных во всех возможных ситуациях. Важно определить реакции, как на допустимые значения данных, так и на недопустимые.
3. Содержит все подписи и ссылки на все рисунки, таблицы и диаграммы, и определение всех терминов и единиц измерения.

***Непротиворечивость*** - Спецификация требований к ПОнепротиворечива тогда и только тогда, когда каждое заявленное требование не противоречит ни одному другому требованию.

***Возможность реал-ции при заданных средствах*** - при этом полный набор треб-ний, может быть реализован в условиях существования ЖЦ проекта.

***Ограниченность*** - при этом набор требований содержит только требования, необходимые для удовлетворения потребностей пользователей и не содержит избыточности требований.

Атрибут предназначен для поддержания анализа требований, при этом повышает понимание требований и описательность требований:

1. Идентификация (уникальность) (Уникальные идентификаторы помогают в транслировании требований. Однажды назначенный идентификатор никогда не изменяется, даже если само требование изменено. И не может повторно использоваться, даже если требование удалено);
2. Приоритет правообладателей (для каждого требования должен быть определен приоритет, который может быть установлен достижением консенсуса);
3. Риск (для каждого треб-я должен быть определены риски связанных с его реал-цией);
4. Источник (кажд треб-е должно вкл атрибут, указ составителя данного треб-я);
5. Обоснование (обязательный атрибут, при этом должно объяснять причины необх-ти треб-я и давать советы по его исп-нию в любых исследованиях предметной области моделирования);
6. Трудность (для каждого треб-ния должны быть отличен уровень предполагаемой трудности его реал –ции);
7. Тип (требования различают по смыслу и типу св-в, которые они представляют(для группировки треб-ний для последующего анализа и распределения).

### **Понятие ошибки и отказа ПО. Понятие надежности ПО. Особенности надежности программ по сравнению с надежностью аппаратуры.**

Майерс предложил считать, что в ПО есть ошибка, если оно не выполняет того, что пользователю от него разумно ожидать.

**Ошибка** **в ПО** - некоторый дефект (изъян) в разработке ПО, который вызывает несоответствие ожидаемых результатов выполнения ПО и фактически полученных.

*Ошибка в ПО* - его поведение не соответствует своей спецификации при условии, что спецификация корректна (правильна).

**Отказ ПО** – проявление в ходе эксплуатации скрытой ошибки разработки, приводящей к получению неприемлемого результата или времени выполнения, либо того и другого вместе.

**Надежность ПО** – св-во технического объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования.

Характеристике **надежность** в соответствии со стандартом ISO/IEC 25010:2011 соответствуют следующие подхарактеристики:

**Завершенность (стабильность**, **Maturity)** – степень соответствия сист., продукта или компонента потребностям в надежности при нормальной эксплуатации. Завершенность зависит от количества ошибок, оставшихся в системе, продукте или компоненте, и определяет возможность их безотказной работы.

**Готовность (Availability)** – степень работоспособности и доступности сист., продукта или компонента тогда, когда требуется их использование.

**Устойчивость к ошибке (Fault tolerance)** – степень функционирования сист., продукта или компонента в соответствии с предназначением, несмотря на наличие сбоев аппаратного обеспечения или ошибок в программном обеспечении.

**Восстанавливаемость (Recoverability)** – степень восстановления поврежденных данных и переустановления требуемого состояния сист. в случае прерывания или отказа продукта или сист.

Одни авторы считают, что понятие надежности ПО не аналогично этому понятию в аппаратуре.

Майерс писал: «Надежность ПО – есть вероятность работы без отказов в течении определенного периода времени, рассчитанная с учетом стоимости для пользователя каждого отказа».

Другие считают, что наблюдаемое поведение ПО при его отказах аналогично поведению при отказах аппаратуры. Это объясняется, что исходные данные в процессе эксплуатации ПО выбираются случайным образом, что приводит к случайному моменту проявления скрытых ошибок в ПО.

В соответствии со стандартом ISO/IEC 25010:2011 **надежность** – степень выполнения системой, продуктом или компонентом заданных функций в заданных условиях в течение заданного периода времени.

Снижение надежности ПО происходит из-за ошибок требований, проектирования и реализации. Отказы из-за этих ошибок зависят от способа использования ПО и контекста используемой среды, а не от времени работы.

Можно выделить следующие **особенности ПО** по сравнению с аппаратным обеспечением компьютеров.

* Ошибки в программах проявляются при их выполнении на некоторых наборах исходных данных (в аппаратуре – из-за сбоев и отказов аппаратуры).
* Компонентам программ не присущи старение и износ.
* Коррекция ошибок в программе приводит к изменению её конфигурации и обычно устраняет саму возможность повторного проявления данной ошибки.
* Тщательный анализ типа отказа и его влияние непрактичны в сложных программах.

### **Процесс разработки ПС в соответствии с СТБ ИСО/МЭК 12207-2003. Причины появления ошибок в ПО.**

**Жизненный цикл (ЖЦ)** – это структура, состоящая из процессов, работ и задач, включающая в себя разработку, эксплуатацию и сопровождение ПС.

Процесс представляет собой набор взаимосвязанных работ, которые преобразуют исходные данные в выходные результаты.

В соответствии с данным стандартом, каждый процесс ЖЦ разделен на набор работ, а каждая работа – на набор задач.

Данный стандарт описывает архитектуру процессов жизненного цикла ПС. При этом он не определяет детали реализации или выполнения работ и задач, входящих в данные процессы. Стандарт не предопределяет конкретную модель ЖЦ или метод разработки ПС.

Пользователи, использующие данный стандарт, должны сами:

выбирать модель ЖЦ применительно к разрабатываемому ПС;

распределять процессы, работы и задачи, выбранные из данного стандарта, на данной модели;

выбирать и применять методы разработки программных средств и выполнять работы и задачи, соответствующие разрабатываемому ПС.

**Все процессы ЖЦ делятся на следующие группы:**

* **основные;**
* **вспомогательные;**
* **организационные.**

**Основные процессы ЖЦ:**

* **заказа** (Содержит работы и задачи, выполненные заказчиком, и состоит из определения потребностей заказчика к системе подготовки и выпуска заявки на подряд, выбор поставщика)**;**
* **поставки** (Состоит из работ и задач, выполняемых поставщиком, начиная с решения о подготовке предложения в ответ заявки на подряд, присланной заказчиком или с подписания договора и вступления с заказчиком в договорные отношения по поставке сист.)**;**
* **разработки;**
* **эксплуатации;**
* **сопровождения** (Выполняется персоналом сопровождения и реализуется при модификациях ПС. Целью данного процесса является изменение существующего ПС при сохранении целостности. Так же процесс охватывает вопрос снятия ПС с эксплуатации)**.**

**Процесс разработки содержит 13 работ**:

**1-я работа – Подготовка процесса** (Должны быть выбраны и адаптированы стандарты, инструментарии и языки программирования для выполнения работ в процессе разработки, так же должна быть выбрана модель ЖЦ).

**2-я работа – Анализ требований к системе** (Должны быть разработаны требования к системе. Требования к системе должны охватывать:

- функции и возможности сист.;

- коммерческие и организационные требования;

- требования пользователя;

- требования безопасности и защиты;

- эргономические требования;(требования к системе, чтобы чел-оператор не страдал при работе)

- требования к интерфейсам;

- эксплуатационные требования;

- требования к сопровождению.

Результатом данной работы должна быть документально оформленная спецификация требований к системе (System Requirement Specification)).

**3-я работа – Проектирование системной архитектуры.** (Должна быть определена общая архитектура всей сист., должны быть указаны объекты технических и программных средств и ручных операций. Ручные операции представляют собой те операции, которые пользователь сист. выполняет вручную, так как, в основном, все разрабатываемые сист. являются автоматизированными, а не автоматическими. Должно быть обеспечено распределение всех требований между объектами архитектуры сист. Затем на основе указанных объектов архитектуры должны быть определены объекты конфигурации технических и программных средств и ручных операций. Должна быть документально оформлена привязка системной архитектуры и требований к системе относительно установленных требований).

**4-я работа – Анализ требований к программным средствам (ПС).** (Применительно к каждому программному объекту архитектуры должна быть разработана спецификация требований. Должны быть установлены и документально оформлены следующие требования к программным средствам:

- функциональные требования;

- технические требования, включая производительность, физические характеристики и окружающие условия, под которые должен быть создан программный объект;

- требования к внешним интерфейсам программного объекта;

- квалификационные требования;

- требования безопасности и защиты;

- эргономические требования, включая требования, относящиеся к ручным операциям;

- требования к определению данных и базе данных;

- требования по вводу в действие и приёмке поставляемого ПС на объектах эксплуатации и сопровождения;

- требования к документации пользователя;

- требования к эксплуатации объекта пользователем;

- требования к обслуживанию пользователя.

Результатом данной работы должна быть спецификация требований к ПС (SoftwareRequirement Specification)).

**5-я работа – Проектирование программной архитектуры.** (Требования к программному объекту должны быть преобразованы в программную архитектуру, которая описывает общую структуру программного объекта и определяет компоненты объекта. Затем должно быть обеспечено распределение всех требований к программному объекту между его компонентами. Должны быть разработаны и документально оформлены:

- общий (эскизный) проект внешних интерфейсов программного объекта и интерфейсов между компонентами объекта;

- общий (эскизный) проект базы данных;

- предварительные версии документации пользователя;

- общие требования к тестированию программного объекта).

**6-я работа – Техническое проектирование ПС.** (Должен быть разработан технический проект для каждого компонента программного объекта. Компоненты программного объекта должны быть уточнены на уровне программных модулей. Должно быть обеспечено распределение технических требований к компонентам между программными модулями. Технический проект должен быть документально оформлен. Кроме того, должен быть разработан и документально оформлен технический проект внешнего интерфейса программного объекта, интерфейса между компонентами и интерфейса между программными модулями. Должны быть разработаны и документально оформлены:

- технический проект базы данных;

- требования к тестированию и программе испытаний программных модулей).

**7-я работа – Программирование и тестирование ПС**. (Должен быть разработан и документально оформлен каждый программный модуль и база данных. Кроме того, должен быть протестирован каждый модуль и база данных на соответствие заданным требованиям. Результаты тестирования должны быть документально оформлены).

**8-я работа – Сборка ПС.** (Все модули и компоненты должны быть собраны в единый программный объект и протестированы. Результаты сборки и тестирования должны быть документально оформлены).

**9-я работа – Квалификационные испытания ПС**. (Должны быть проведены испытания (тестирование) на соответствие квалификационных требований программному объекту. При тестировании должна быть проверена правильность выполнения каждого требования. Ошибки фиксируются и документально оформляются).

**10-я работа – Cборка сист.** (Программные объекты конфигурации должны быть собраны в единую систему вместе с объектами технической конфигурации и при необходимости с другими системами).

**11-я работа – Квалификационное испытание сист.** (Должны быть проведены в соответствии с требованиями, установленными к системе (эталон – спецификация требований к системе). Результаты квалификационных испытаний документально оформляются).

**12-я работа – Ввод ПС в действие.**

**13-я работа – Обеспечение приемки ПС.**

**Процесс эксплуатации**:

* Подготовка процесса;
* Эксплуатационные испытания;
* Эксплуатация сист.;
* Поддержка пользователя.

**Вспомогательные процессы ЖЦ ПС:**

-Документирование;

-Управление конфигурацией;(поддержанием целостности сист. на протяжении всего ЖЦ)

-Обеспечение качества;

-Верификация;(правильно ли создана с-ма) (проверяет соответствие ПО системной спецификации, в частности функциональным и нефункциональным требованиям)

-Аттестация;(правильно ли работает с-ма) (необходимо убедиться, что программный продукт соответствует ожиданиям заказчика)

- Совместный анализ;

- Аудит;

- Решение проблем;

**Документирование** – процесс формализованного описания информации, созданной в процессе или работе ЖЦ ПС.

**Управление конфигурацией** – процесс применения административных и технических процедур на всем протяжении ЖЦ программных средств.

**Обеспечение качества** – процесс обеспечения соответствующих гарантий того, что программные средства и процессы в ЖЦ проекта соответствуют установленным требованиям и утвержденным планам.

**Верификация** – процесс определения того, что программное средство функционирует в полном соответствии с требованиями или условиями, реализованными в предшествующих работах. Верификация может применяться не только к ПС, но и к любым другим результатам работы. Например, верификации могут подвергаться документация, программная архитектура и т.д.

**Аттестация** – процесс аттестации является процессом определения полноты соответствия установленных требований созданной сист, или ПС их функциональному назначению. Аттестации могут подвергаться любые промежуточные продукты процесса обработки, но чаще аттестации подвергаются продукты на завершающей стадии обработки.

**Совместный анализ** – процесс оценки состояний и результатов работ по всему проекту. Совместные анализы проводятся в течении всего ЖЦ договора и применяются как на уровне управления проектом, так и на уровне его технической реализации.

**Аудит** – процесс определения соответствия требованиям, планам и условиям договора.

**Решение проблем** – процесс анализа и решения проблем, которые обнаружены в ходе выполнения разработки, эксплуатации, сопровождения или других процессов.

**Организационные процессы ЖЦ:**

* Управления;
* Создания инфраструктуры;
* Усовершенствования;
* Обучения.

**Управление** - состоит из работ и задач, которые могут быть использованы любой стороной, управляющей соответствующим процессом (планирование, выполнение, контроль, оценки).

**Создание инфраструктуры** - является процессом установления и обеспечения (сопровождения) инфраструктуры, необходимой для любого другого процесса. Инфраструктура может содержать технические и программные средства, методики, стандарты и условия для разработки, эксплуатации и сопровождения.

**Усовершенствование** - процесс установления оценки измерения, контроля и улучшения любого процесса жизненного цикла ПС.

**Обучение** – процесс обеспечения первоначального и продолженного обучения персонала работам по заказу, поставке, разработке, эксплуатации или сопровождению ПС.

Одним из распространенных определений ошибки является расхождение между спецификацией требований и ПО. Такое определение предполагает корректность спецификации требований, что не всегда так.

Ошибка в ПО представляет некоторый дефект в его разработке, который вызвал несоответствие ожидаемым результатам выполнения ПО. Дефект может возникнуть практически во всех работах процесса разработки.

В классической теории надёжности систем, надёжность любого объекта закладывается при проектировании, реализуется при изготовлении и расходуется при эксплуатации. При этом жизненный цикл объекта состоит из трёх этапов: проектирование, изготовление и эксплуатация. В общем, ПО относится к сложным объектам. Оно имеет по сравнению с разработкой аппаратуры как схожие моменты, так и свои особенности.

Например, в 4-й работе процесса разработки ЖЦ ПС дефект – отсутствие требований или их неверная трактовка.

В 5-й работе процесса разработки ЖЦ ПС правильные требования могут быть преобразованы в неправильную программную архитектуру.

В 6-й работе процесса разработки ЖЦ ПС правильная программная архитектура может быть преобразована в неправильный технический проект, содержащий алгоритмы и модульную структуру ПС.

В 7-й работе процесса разработки ЖЦ ПС может быть внесен огромный спектр программных ошибок (зацикливание, отсутствие предварительной инициализации переменных и т.д.).

В ходе собственно тестирования в 7, 8, 9, 11-й работах процесса разработки ЖЦ ПС могут внестись новые ошибки. Статистика говорит, что исправление каждой 3-й ошибки вносит новую.

**Главные причины внесения ошибок при тестировании** – плохое документирование проекта, низкий уровень характеристики сопровождаемости ПС.

Как показывает анализ процесса разработки ПО, **важнейшей причиной появления ошибок** (дефектов, багов) в ПО является неправильный перевод информации из одного представления в другое. Например, требования к ПС преобразуется архитектуру ПС, архитектура ПС преобразуется в техническое проектирование (алгоритм), и т.п.

В общем, разработку ПО можно просто описать как ряд процессов перевода, начинающаяся с перевода исходной задачи в различные промежуточные решения, заканчивающихся большим набором команд (операторов), выполняющихся на компьютере и решающих данную задачу. Когда не удается полно и точно перевести представление исходной задачи или промежуточного решения в другое более детальное решение, тогда и возникают ошибки в ПО.

### **Методы обеспечения надежности ПО.**

**Методы обеспечения надёжности** можно разбить на три группы:

1. *Предупреждение ошибок;*
2. *Обнаружение и исправление ошибок;*
3. *Обеспечение устойчивости к ошибкам.*

1)**Предупреждение ошибок**

*Сложность* – основная причина ошибок перевода и таким образом одна из главных причин ненадёжности ПО. (Она почти не поддается ни точному определению, ни измерению).

*Мера сложности объекта* - количество интеллектуальных усилий, необходимых для понимания этого объекта.

В общем сложность является функцией взаимодействия между его компонентами. (Например, сложность структуры ПС определяется связями между компонентами ПС. Сложность отдельного модуля определяется связями между операторами (командами модуля) и т.п.)

Для **снижения структурной сложности ПС** широко используются два принципа:

1. Независимость компонентов/модулей ПС (максимально усилить независимость модулей);
2. Иерархическая структура/иерархия ПС (Иерархия позволяет декомпозировать ПО на компоненты с учетом ограничений на их сложность. Каждый вышестоящий уровень представляет собой совокупность структурных отношений между элементами нижестоящих уровней.).

В общем, чтобы уменьшить сложность ПО, нужно разбить его (декомпозировать) на множество небольших достаточно простых для понимания независимых модулей.

**Высокой степени независимости модулей** можно достичь**:**

1. Усилением внутренних связей в каждом модуле;
2. Ослаблением взаимосвязи между модулями.

Концепция уровня позволяет понять ПО, скрывая несущественные уровни детализации. (Например, иерархия в IDEFO. Функциональная модель имеет иерархическую структуру, состоящую из множества IDEFO-диаграмм. Каждая диаграмма любого уровня может быть декомпозирована на несколько диаграмм нижележащего уровня. При этом, каждая из диаграмм должна содержать не более шести блоков из-за ограничений по сложности. Таким образом сложность диаграмм должна находится в пределах понимания данной диаграммы. Иерархия и ограничение сложности на каждом уровне модели позволяют заказчику понять модель, а разработчику сделать меньше ошибок при разработке функциональной модели предметной области).

В первом методе понижения структурной сложности усиление внутренних связей в каждом модуле достигается за счет увеличения связности модуля. Связность модуля определяется как мера зависимости его частей. При этом, чем выше связность, тем больше отдельные части модуля зависят друг от друга.

Важнейшим методом увеличения независимости модулей является ослабление связи между ними. Мерой ослабления связи между модулями является сцепление модулей, которое является мерой относительной независимости модулей. Сцепление модулей является мерой взаимодействия модулей по данным и характеризуется как способом передачи данных, так и свойствами этих данных. Сцепление влияет на сохранность данных при модификациях. Слабое сцепление определяет высокий уровень независимости модулей. Независимые модули могут быть модифицированы без переделки других модулей. В общем два модуля являются полностью независимыми, если в каждом из них не используется никакая информация о другом модуле.

Таким образом максимально независимый модуль имеет максимальную связность и минимальное сцепление.

Если рассматривать программу как набор предложений, связанных между собой некоторыми отношениями (как по выполняемым функциям, так и по обрабатываемым данным), то главная задача, стоящая перед разработчиками, это распределить эти предложения между модулями, таким образом, чтобы:

1. Предложения в каждом модуле были максимально связаны;
2. Связь между любой парой предложений в разных модулях была минимальна.

При проектировании нужно стремиться, во-первых, реализовать отдельные функции отдельными модулями (высокая связность), во-вторых, ослаблять связи между модулями по данным, применяя передачу параметров (слабое сцепление модулей).

Таким образом, очевидно, что предупреждения возникновения ошибок представляет собой оптимальный путь к достижению НПО. Но гарантировать отсутствие ошибок в общем невозможно.

2)**Обнаружение и исправление ошибок**

Данная группа методов обеспечения надежности основывается на предположении о том, что, несмотря ни на что, ошибки в ПО всё же будут. Тестирование и отладка являются основным средством обнаружения и устранения ошибок для обеспечения надежности ПО. Наибольшие усилия затрачиваются во время тестирования и отладки ПО. Под отладкой понимается процесс внесения изменения в ПО с целью обеспечения его соответствия заданным требованиям. В начале отладки производится проверка ПО с помощью различных методов тестирования. Так как исчерпывающее тестирование практически невозможно, то нет полной гарантии того, что после тестирования ПО не осталось ошибок. Можно так же отметить, что само по себе тестирование не является методом повышения надёжности, т.к. только позволяет лишь выявить имеющиеся в ПО ошибки. Если же по результатам тестирования будут приняты меры к устранению найденных ошибок, то только тогда будет повышена надёжность ПО.

В общем, использование методов первой и второй групп позволяет существенно уменьшить количество ошибок в ПО, но не может их полностью ликвидировать. Поэтому важной и актуальной является задача создания ПО, устойчивого ко всем *факторам ненадёжности*:

1. Аппаратным, обусловленным влиянием сбоев и отказов аппаратуры;
2. Программным, обусловленным влиянием ошибок в программах;
3. Внешним, вследствие искажений в исходных данных, поступающих на вход программ.

3) **Устойчивость к ошибкам в ПО**

Данная группа методов обеспечения надежности также основывается на предположении о том, что, несмотря ни на что, ошибки в ПО всё же будут.

Методы этой группы ставят своей задачей обеспечить функционирование ПО при наличии в нем ошибок.

Устойчивость ПО достигается за счёт внесения в него различных форм **избыточности**:

* Временной;
* Информационной;
* Программной.

Например, программная избыточность может обеспечиваться использованием метода N-версионного программирования, когда независимо создаётся N версий разрабатываемого ПО разными коллективами. Но объективный недостаток данного метода – увеличение стоимости разработки ~ в N раз.

### **Модель Джелинского-Моранды. Определение характеристик модели с помощью метода максимального правдоподобия.**

**Допущения модели**:

1. Время до очередного отказа распределено по экспоненциальному закону;
2. Все ошибки равновероятны и их появление не зависит друг от друга;
3. Частота появления ошибок (интенсивность отказов, в англоязычной литературе так же имеет название *функция риска*) пропорциональна числу не выявленных ошибок: ***l(ti)=Kjm·[E0-(i-1)]***,

где: ***E0*** – число ошибок в ПО до начала тестирования и отладки; ***Kjm*** – коэффициент Джелинского-Моранды; ***ti*** – интервал времени между (***i-1)***-й и ***i***-й обнаруженными ошибками; ***i*** *–* число ошибок, обнаруженных к моменту отладки ***ti***;

1. ***l(ti) = const***на интервале между двумя смежными моментами появления ошибок;
2. Каждая обнаруженная ошибка в ПО немедленно устраняется и число оставшихся ошибок уменьшается на 1;
3. Ошибки корректируются без внесения новых ошибок.
4. Время ***ti*** соответствует длительности выполнения ПО на компьютере и не учитывает простои компьютера для анализа результатов и их корректировки.

|  |  |
| --- | --- |
|  | Рис. 5.1. Зависимость интенсивности отказов от времени  Так как согласно допущению 1 **время до очередного отказа** распределено по экспоненциальному закону, то вероятность безотказной работы, т.е. отсутствия следующего ***i***-го отказа равна:    **Плотность вероятности отказов** равна:    **Среднее время безошибочной работы** *или средняя наработка до следующего* ***i****-ого отказа* определяется выражением:    Параметрами модели (исходными данными) является статистика об ошибках, т.е. интервалы времени между отказами.  **Характеристиrи модели** (то, что нужно определить):   * ***P(ti)*** – вероятность отсутствия, следующего ***i***-ого отказа; * ***m(t)***– среднее время до очередного i-ого отказа; * ***Kjm*** – коэффициент Джелинского-Моранды; * ***E0*** –число ошибок в ПО до начала тестирования и отладки.   Характеристики модели определяются при помощи ***метода максимального правдоподобия***. |
|  |

**Критический анализ модели Джелинского-Моранды.**

Многие допущения модели спорны. Например:

- что все ошибки одинаково серьёзны;

- ошибка исправляется немедленно (или ПО не используется до тех пор, пока найденная ошибка не будет исправлена);

- ошибки корректируются без внесения новых ошибок;

- основное допущение - то, что для всех программ ***λ(t***i***)*** имеет вид, представленный на рис. 5.1, и ***λ(ti)***= const между появлениями ошибок, также спорно.

При этом каждая ошибка уменьшает ***λ(ti)*** на постоянную величину ***KJM***.

В общем можно утверждать, что каждая программа имеет своё собственное уникальное распределение ***λ(ti)***. Даже, при каждой установке каждой программы ***λ(ti)*** – своё. Можно также отметить, что функция ***λ(ti)*** для одной и той же программы может меняться со временем или при обнаружении каждой ошибки.

Как показывает практика, модель JM устойчиво даёт завышенное число остаточных ошибок.

### **Модель Шика-Волвертона. Определение характеристик модели с помощью метода максимального правдоподобия.**

Дополнительно к допущениям модели Джелинского–Моранды используется следующее *допущение*: частота появления ошибок пропорциональна времени отладки программы ***ti***, т.е. вероятность обнаружения ошибок с течением времени должна возрастать.



где:

***E0*** – число ошибок в ПО до начала тестирования и отладки;

***Ksw*** – коэффициент Шика-Волвертона;

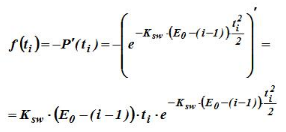
***ti*** – интервал времени между (***i-1)***-й и ***i***-й обнаруженными ошибками;

***i*** *–* число ошибок, обнаруженных к моменту отладки ***ti***.

Для вероятности безотказной работы имеем следующее выражение:



Функция плотности будет иметь вид:



Оценка характеристик модели Шика-Волвертона ***E0*** и ***Ksw*** осуществляетсяпри помощи метода максимального правдоподобия.

***Рассмотренные модели Джелинского-Моранды, Шика-Волвертона можно использовать как на этапе тестирования и отладки, так и на этапе эксплуатации.***

### **Геометрическая модель надежности ПО. Определение характеристик модели с помощью метода максимального правдоподобия.**

Основные ***исходные предпосылки*** для этой модели следующие:

1. Общее число ошибок неограниченно;
2. Обнаружение ошибок не равновероятно;
3. Обнаружение ошибок – процесс, независимый от ошибок;
4. ПО работает в условиях, близких к реальным;
5. Интенсивность обнаружения ошибок образует геометрическую прогрессию, и она в интервале между появлениями ошибок постоянна.

Интенсивность отказов имеет следующий вид:



где: ***λ*** ***(0) = D*** – исходное значение интенсивности отказов; ***K*** – константа пропорциональности ***0 < K < 1;*** ***ti*** – время между появлениями (***i – 1***)-ой и ***i***-ой обнаруженных ошибок.

### **Модель Миллса.**

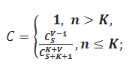
В модели не делается никаких предположений о поведении интенсивности отказов **λ(t).**

Модель строится на статистическом фундаменте.



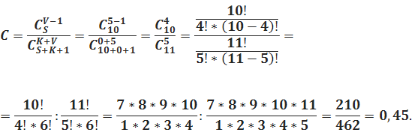
Например, если в программу внесено S = 20 ошибок и к некоторому моменту тестирования обнаружено n = 15 собственных и V = 5 внесенных ошибок, то оценка N = 20 \* 15/5 = 60 ошибок.





число сочетаний из S по V-1;

 число сочетаний из S+K+1 по K+V.



С вероятностью C=0,45 можно утверждать, что в программе нет ошибок (K=0).

***Достоинства модели:***

1. Модель Миллса одновременно математически проста и интуитивно понятна.
2. Данная модель может оказывать положительное психологическое влияние на группу тестирования, т.к. при обнаружении не всех внесенных ошибок тестировщик уверен в том, что программа содержит еще ошибки и продолжает дальнейшее тестирование.

***Недостаток модели:***

1. Процесс внесения ошибок является самым слабым местом модели, т.к. предполагается, что собственные и внесенные ошибки обнаруживаются с одинаковой вероятностью. Из этого следует, что внесенные ошибки должны быть типичными для данной программы. Но сложность состоит в том, что неизвестно, какой должна быть типичная ошибка, если тестирование проводит не тот, кто писал программу.

### **Модель надежности ПС в соответствии с ГОСТ 28195-99.**

***Оценка качества*** программного средства - совокупность операций, включающих:

1. Выбор номенклатуры показателей качества оцениваемого программного средства;
2. Определение значений показателей;
3. Сравнение показателей с базовыми значениями.

Оценка качества должна проводиться применительно ко всем работам жизненного цикла ПС при планировании показателей качества ПС, контроле качества в процессе разработки, проверке эффективности модификации ПС в процессе сопровождения.

Рекомендован метод интегральной оценки качества ПС, основанный на иерархической модели качества.

**Надежность** является одной из шести основных характеристик качества ПС. С учетом этого рассмотрим данный метод оценки в применении к надежности ПС.

Выбор номенклатуры показателей надежности для конкретного программного средства осуществляется с учетом его назначения и требований области применения в зависимости от принадлежности ПС к тому или иному подклассу, определяемому ***общесоюзным классификатором продукции (ОКП).***

Оценка надежности ПС производится с учетом фазы жизненного цикла (ЖЦ). Фазы жизненного цикла представляют собой совокупности процессов или работ стандарта *СТБ ИСО/МЭК 12207-2003*.

**Оценка надежности ПС заключается в выборе номенклатуры показателей, их оценке и сопоставлении с базовыми значениями.**

**Показатели качества продукции -** количественная характеристика одного или нескольких свойств продукции, составляющих ее качество, рассматриваемая применительно к определенным условиям ее создания и эксплуатации или потребления.

**Четырехуровневая иерархическая модель надежности:**

***Уровень 1*** *-* Факторы качества (характеристики качества, в данном случае надежности);

***Уровень 2*** *-* Критерии качества (подхарактеристики качества, в данном случае надежности);

***Уровень 3*** *-* Метрики;

***Уровень 4*** *-* Оценочные элементы или единичные показатели (в международных стандартах отсутствует).



Рис. 6.2. Модель надежности для фазы проектирования



Рис. 6.2. Модель надежности для фаз реализации, тестирования,

изготовления и сопровождения

### **Оценка надежности ПС в соответствии с ГОСТ 28195-99.**

*Оценка надежности программного средства проводится в следующей последовательности:*

1. На фазе анализа проводится выбор показателей и их базовых значений.
2. Для показателей надежности на всех уровнях принимается единая шкала оценки (от 0 до 1).
3. В процессе оценки надежности на каждом уровне (кроме уровня оценочных элементов) проводится вычисление двух величин:

* абсолютного показателя Pij;
* относительного показателя Rij

где ***j*** – порядковый номер показателя данного уровня для ***i***-го показателя вышестоящего уровня.

Относительныйпоказательявляется функцией показателяи его базового значенияи определяется по формуле



*ГОСТ 28195–99* содержит таблицу, содержащую базовые значения для показателей надежности второго уровня (критериев). Данные значения определяются подклассом ПС.

Базовые значения для надежности и ее показателей 3-го уровня формируются методом экспертного опроса с учетом назначения ПС или на основании показателей существующих аналогов или расчетного эталонного ПС. Значения базовых показателей ПС должны соответствовать значениям показателей, отражающих современный уровень качества и прогнозируемый мировой уровень.

4) Каждый показатель надежности второго и третьего уровней характеризуется двумя параметрами:

* количественным значением;
* весовым коэффициентом Vij.

Сумма весовых коэффициентов всех показателей некоторого уровня, относящихся к показателю вышестоящего уровня, постоянна и равна 1:



где: ***J*** – общее количество всех показателей ***j***-го уровня, относящихся к ***i***-му показателю вышестоящего уровня, определенных в стандарте.

*ГОСТ 28195–99* содержит таблицы, содержащие перечни весовых коэффициентов для показателей второго и третьего уровней (критериев и метрик). Количественные величины весовых коэффициентов зависят от фазы жизненного цикла ПС и подкласса ПС.

5) Определениеусредненной оценки **mkq** оценочного элемента по нескольким его значениям (измерениям) mqt осуществляется по следующей формуле (*формула для вычисления значений показателей надежности 4-го уровня*):



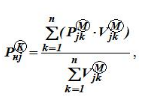
где: ***k*** – порядковый номер метрики; ***q*** – порядковый номер оценочного элемента; ***T*** – число значений (измерений) оценочного элемента; ***t*** – номер значения оценочного элемента.

6)МЕТРИКА Итоговая оценка ***k***-ой метрики ***j***-го критерия определяется по формуле (*формула для вычисления значений показателей надежности 3-го уровня*):



где: – признак метрики; ***Q*** – число оценочных элементов, реально используемых при оценке ***k***-й метрики.

7)КРИТЕРИЙ Абсолютные показатели ***j***-го критерия надежности  вычисляются по формуле (*формула для вычисления значений показателей надежности 2-го уровня*):

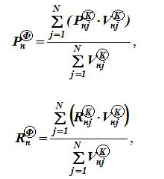


где: ***n*** – число метрик, относящихся к ***j***-му критерию, реально используемых при оценке;  – признак критерия.

8)КРИТЕРИЙ Относительные значения  ***j***-го критерия надежности  по отношению к базовому значению  определяются по формуле



9)ФАКТОР Абсолютные и относительные значения надежности определяются по формулам:



где:  – признак фактора; ***N*** – число критериев надежности, реально используемых при оценке.

### **Модель надежности ПС в соответствии с СТБ ИСО/МЭК 9126-2003.**

**Трехуровневая иерархическая модель качества**:

**1-й уровень** шесть характеристик качества;

**2-й уровень** подхарактеристики качества;

**3-й уровень** метрики качества.

**Надежность (одна из 6 характеристик)** - совокупность свойств, характеризующая способность программного средства сохранять заданный уровень пригодности в заданных условиях в течении заданного интервала времени.

*Надежность* включает подхарактеристики:

1. Стабильность;
2. Устойчивость к ошибке;
3. Восстанавливаемость.

**Стабильность (Maturity)** – способность программного продукта избегать отказов вследствие ошибок в программах.

**Устойчивость к ошибке (Fault tolerance)** – способность программного продукта поддерживать заданный уровень качества функционирования в случаях ошибок в программах или нарушения заданного интерфейса ПП.

**Восстанавливаемость (Recoverability)**– способность программного продукта восстанавливать заданный уровень качества функционирования и данные, поврежденные в случае отказа. Одним из показателей восстанавливаемости является длительность восстановления.

Следует отметить, что в данном стандарте регламентирован только первый уровень модели – уровень характеристик. Приведенные подхарактеристики носят рекомендательный характер. Примеры метрик вообще отсутствуют.

### **Модель надежности ПС в соответствии с ISO/IEC 25010-2011.**

**Части модели качества:**

1. Продукта;
2. Модель качества в использовании.

**Иерархическая структура**

1. уровень характеристик (регламентирована);
2. уровень подхарактеристик (регламентирована);
3. уровень метрик.

**Модель качества продукта** имеет отношение к:

1. Статическим свойствам программного средства (определяются с помощью внутренних метрик/мер качества);
2. Динамическим свойствам компьютерной сист. (определяются с помощью внешних метрик качества)

**Надежность (одна из 8 характеристик, Reliability)** определяется как степень выполнения системой, продуктом или компонентом заданных функций в заданных условиях в течение заданного периода времени (Ограничения надежности в процессе эксплуатации вызваны ошибками в требованиях, проектировании и кодировании).

***Подхарактеристики Надежности:***

**Завершенность (стабильность**, **Maturity)** – степень соответствия сист., продукта или компонента потребностям в надежности при нормальной эксплуатации (Завершенность зависит от количества ошибок, оставшихся в системе, продукте или компоненте, и определяет возможность их безотказной работы).

**Готовность (Availability)** – степень работоспособности и доступности сист., продукта или компонента тогда, когда требуется их использование.

Внешне Готовность может быть оценена соотношением времен, в течение которых система, продукт или компонент находится в работоспособном и неработоспособном состоянии. Поэтому Готовность представляет собой комбинацию Завершенности (обусловливающей частоту отказов), Устойчивости к ошибке и Восстанавливаемости (определяющей продолжительность времени восстановления после каждого отказа).

**Устойчивость к ошибке (Fault tolerance)** – степень функционирования сист., продукта или компонента в соответствии с предназначением, несмотря на наличие сбоев аппаратного обеспечения или ошибок в программном обеспечении.

**Восстанавливаемость (Recoverability)** – степень восстановления поврежденных данных и переустановления требуемого состояния сист. в случае прерывания или отказа продукта, или сист. (В случае отказа компьютерная система будет некоторое время находиться в неработоспособном состоянии. Продолжительность этого времени определяется ее восстанавливаемостью).

### **Процесс оценки надежности ПС в соответствии с СТБ ИСО/МЭК 9126-2003.**

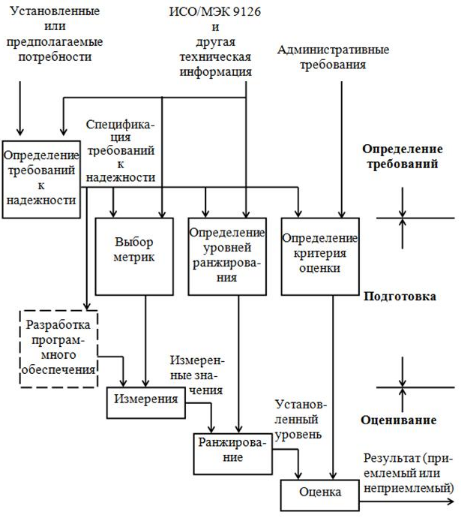


Рис. 6.4. Модель процесса оценки по СТБ ИСО/МЭК 9126–2003

**Процесс оценки** состоит из:

1. Определение требований к надежности ПС;
2. Подготовка к оцениванию;
3. Процедура оценивания.

Данный процесс может применяться после любой подходящей работы жизненного цикла для каждого компонента программного продукта.

**1. Определение требований к надежности**

Целью данной стадии является установка требований в терминах подхарактеристик надежности. Требования выражают потребности внешнего окружения ПС и должны быть определены до начала разработки. Так как ПС разделяется на компоненты, то требования для ПС в целом могут отличаться от требований для отдельных компонентов.

**2. Подготовка к оцениванию**

Целью второй стадии является подготовка основы для оценивания. Данная стадия состоит из трех этапов.

1. ***Выбор метрик надежности***

С учетом регламентированной в *СТБ ИСО/МЭК 9126–2003* иерархической модели качества уровень характеристики надежность ПС определяется уровнем входящих в нее подхарактеристик, а значения подхарактеристик в свою очередь определяются значениями входящих в них метрик.

В стандарте *СТБ ИСО/МЭК 9126–2003* набор рекомендуемых метрик отсутствует. Поэтому существует потребность в установлении метрик, которые соотносятся с подхарактеристиками надежности ПС. Каждый количественный признак и каждое количественно оцениваемое взаимодействие ПС с его окружением, которые соотносятся с надежностью, могут быть приняты в качестве ее метрики.

1. ***Определение уровней ранжирования***

Для измерения количественных признаков надежности ПС используются метрики. Измеренные значения отображаются на некоторой шкале. Данные значения не показывают уровень удовлетворения требований к надежности ПС. Для этой цели шкалы метрик должны быть разделены на диапазоны, соответствующие различным степеням удовлетворения требований.

Возможны следующие **диапазоны ранжирования** (рис. 6.5):

* Разделение шкалы на две категории: неудовлетворительно и удовлетворительно;
* Разделение шкалы на четыре категории (отлично, хорошо, удовлетворительно, неудовлетворительно), ограниченные соответственно запланированным уровнем, текущим уровнем для существующего или альтернативного продукта, и уровнем худшего случая.



Рис. 6.5. Варианты ранжирования измеренных значений метрик

Текущий уровень определяется для управления тем, чтобы новая система не становилась хуже по сравнению с существующей. Запланированный уровень определяет уровень, который считается достижимым при доступных ресурсах. Уровень худшего случая определяет границу принятия пользователем в случае, если изделие не удовлетворяет запланированному уровню. Так как надежность ПС связана с конкретными потребностями, общие уровни ранжирования невозможны и должны определяться для каждого конкретного оценивания.

1. ***Определение критерия оценки***

Для определения общей надежности ПС должна быть учтена вся совокупность результатов оценивания различных метрик. Оценщик должен подготовить для этого процедуры, используя, например, таблицы решений или средние взвешенные значения. Обычно при этом учитываются и другие аспекты, такие как время и стоимость, которые являются косвенными факторами надежности ПС.

***3. Процедура оценивания***

*Реализуется тремя этапами:*

1. ***Измерение*** *(*Для измерения выбранные метрики применяются к ПС. Результатом являются значения в масштабах метрик).
2. ***Ранжирование*** *(*На этапе ранжирования устанавливается уровень ранжирования для измеренного значения (см. рис. 6.5)).
3. ***Оценка*** *(*Оценка является последним этапом процесса оценивания ПС, на котором обобщается множество установленных уровней. Результатом является заключение о надежности ПС (приемлемый или неприемлемый уровень надежности)).

### **Внутренние метрики надежности ПС.**

В стандарте*ISO/IEC 25010:2011* определены две части **модели качества**:

1. Модель качества продукта;
2. Модель качества в использовании.

Модель качества продукта имеет отношение к:

1. Статическим свойствам программного средства (определяются с помощью внутренних метрик (мер) качества);
2. Динамическим свойствам компьютерной сист (определяются с помощью внешних метрик (мер) качества).

***Внутренние метрики*** – это метрики, измеряющие собственные свойства ПС. Они измеряются в процессе разработки ПС на основе спецификации требований, результатов проектирования, исходного кода или другой документации ПС. Внутренние метрики дают возможность оценить качество промежуточных программных продуктов разработки, предсказывая качество конечного программного средства.

### **Внешние метрики надежности ПС.**

***Внешние метрики*** – это метрики, предназначенные для измерения качества программного продукта путем измерения поведения сист., частью которой является данный продукт. Внешние метрики могут использоваться в процессе эксплуатации и на стадиях тестирования или испытаний в процессах разработки и сопровождения ПС, когда уже созданы исполнимые коды программного продукта.

### **Введение в тестирование ПО. Понятие тестирования ПО. Что может и не может тестирование ПО. Объекты тестирования ПО.**

Одни специалисты под тестированием понимают метод проверки корректности (правильности) ПО, где под корректностью понимается степень соответствия ПО своей спецификации.

Другие специалисты считают, что тестирование не позволяет установить корректность ПО, т.к. тестирование может служить для доказательства наличия ошибок, но не доказать их отсутствия.

Майерс считает, что тестирование –процесс исполнения программ с целью обнаружения ошибок.

***Тестирование*** – это процесс анализа или эксплуатации ПО с целью выявления дефектов.

*Тестирование может:*

1. Обнаруживать ошибки;
2. Показывать соответствие функций программы ее назначению, т. е. проверять правильность работы (реализацию) функциональных требований;
3. Отображать надежность как индикатор качества программы.

*Тестирование не может:*

1. Показывать отсутствие ошибок.

**Можно тестировать:**

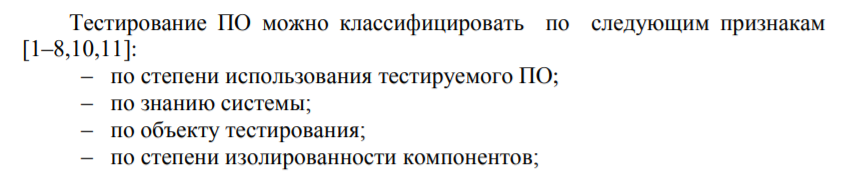
1. Исполняемые коды программ при их непосредственном запуске и исполнении;
2. Исходные коды программ без их запуска и исполнения;
3. Требования к ПО;
4. План проекта;
5. Тестовый план;
6. Документацию описывающею программную архитектуру;
7. Тестовые сценарии;
8. Документацию для конечных пользователей;
9. И т.п.

Существующие стандарты определяют ПО как совокупность программ (кодов) и программной документации. Поэтому тестировать в общем можно как собственно ПО, так и программную документацию. Практически можно тестировать любой промежуточный продукт разрабатываемого программного средства. В дальнейшем под объектом тестирования будет пониматься исполняемое ПО и иные случаи будут оговариваться особо.

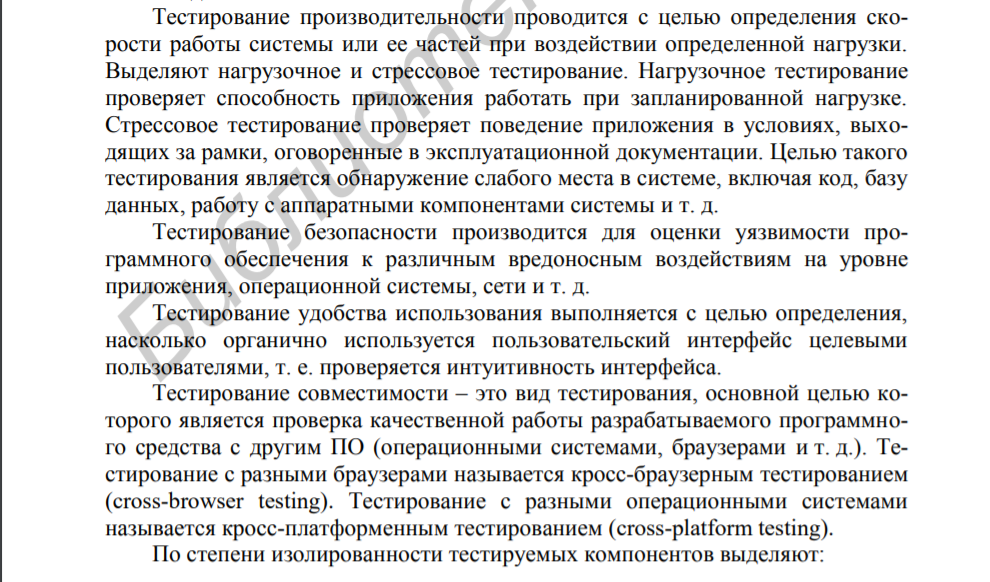
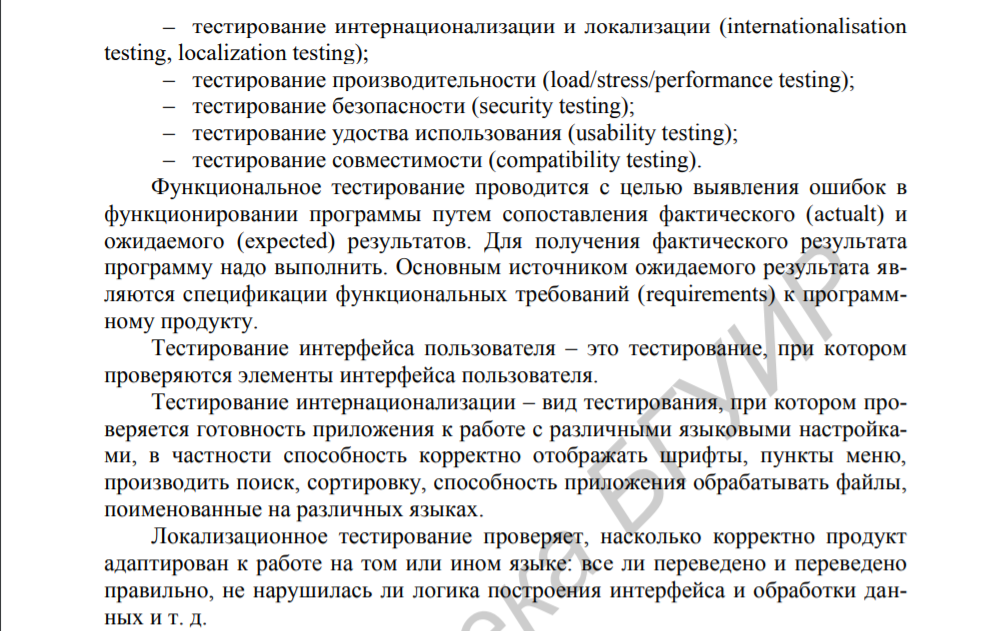
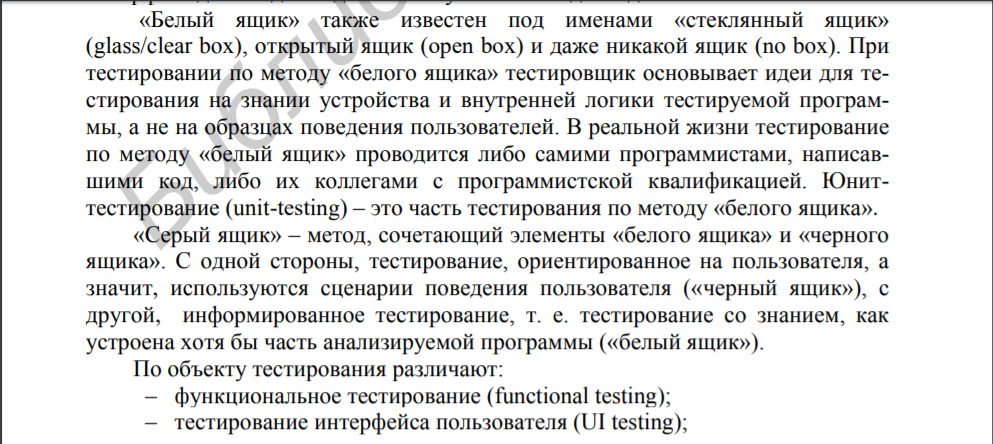
Программы как объекты тестирования имеют две основные ***особенности***:

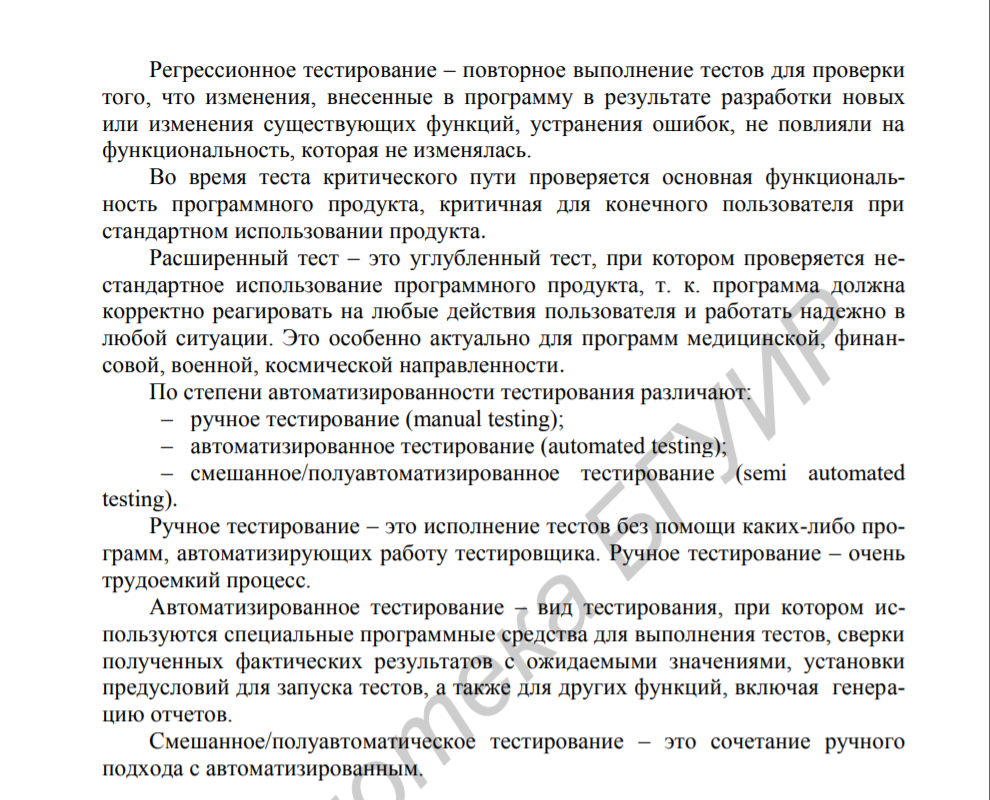
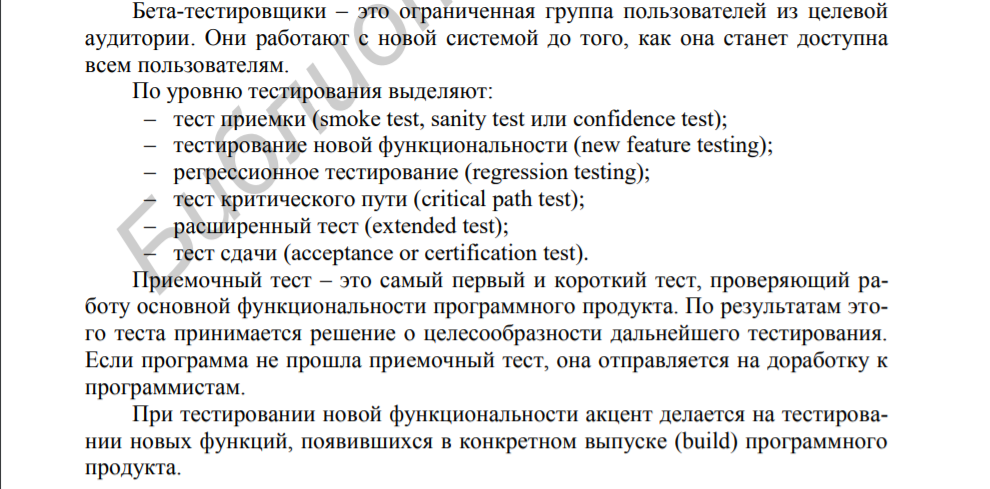
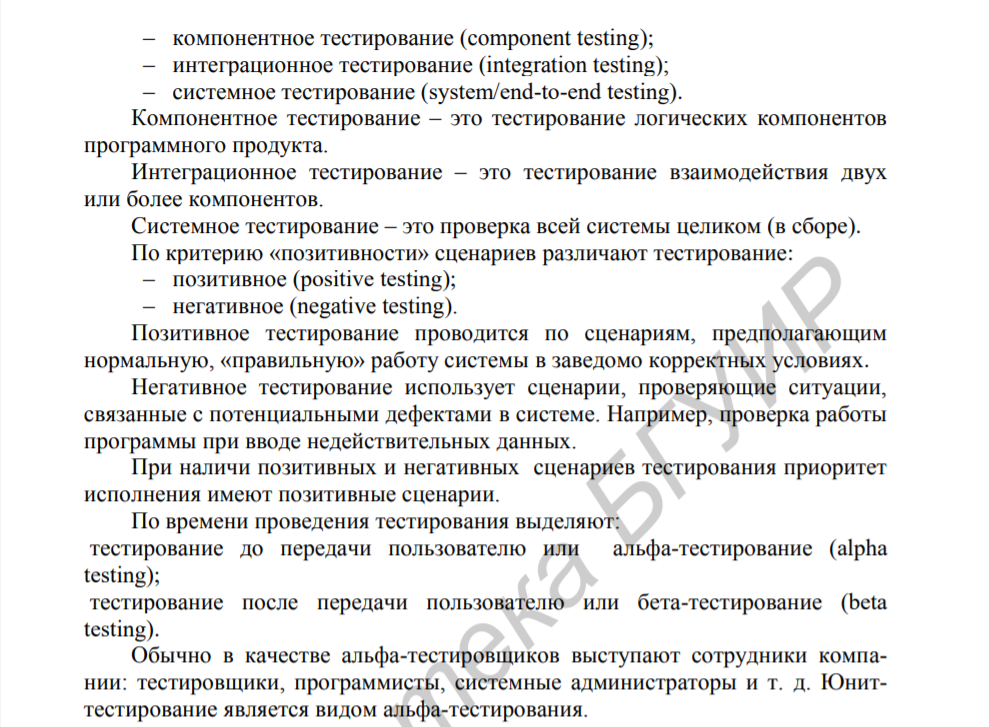
1. Отсутствие полностью определённого эталона, которому должны соответствовать все результаты проверяемого программного обеспечения. Для тестирования программ в качестве эталонов чаще всего используются косвенные данные, которые не полностью отражают функции тестируемой программы.
2. Высокая сложность ПО и принципиальная невозможность построения тестовых наборов, достаточных для исчерпывающего тестирования программ.

### **Виды тестирования ПО.**









### **Принципы разработки тестов.**



Рис. 8.1. Принципы разработки тестов

**Первый подход.**

Данный подход соответствует левой границе спектра, и заключается в том, что ПО рассматривается как метод «*черный ящик»*, когда в процессе проектирования тестов структура ПО не интересует тестировщика.



Рис. 8.2. Тестирование по принципу “Черный ящик”

Методы, которые рассматривают ПО как «черный ящик», называются ***функциональными методами***. В данном случае известны функции, которые должно выполнять ПО. В идеале должна тестироваться работа каждой такой функции по всей области ее определения. Основное место применения тестов «черного ящика» - проверка правильности выполнения функций ПО на уровне интерфейса.

В общем случае тестировщика не интересует прохождение всех путей или ветвей программы. Его удовлетворяет тот факт, что ПО ведет себя так, как указано в спецификации.

В общем случае данный подход практически не осуществим, т.к. проверка работы каждой функции для всей области ее определения практически неосуществима. Поэтому тестирование функций осуществляется с помощью специальных методов за счёт выбора отдельных значений тестовых данных.

**Достоинства метода «чёрного ящика»:**

* тестирование выполняется тестировщиками, а не программистами и поэтому оно достаточно объективно;
* тестировщику не обязательно знать программный код тестируемого ПО;
* тестирование выполняется с точки зрения конечного пользователя, что важно для эксплуатации ПО пользователем;
* тест-кейсы могут разрабатываться сразу после разработки спецификации требований и до начала кодирования;

**Второй подход.**

Данный подход заключается в изучении структуры ПО и ее логики. Данный метод называют методом «белого ящика».

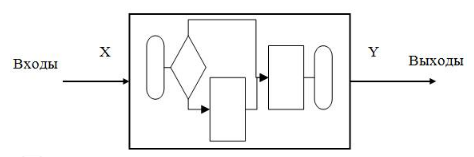


Рис. 8.3. Тестирование по принципу “Белый ящик”

Методы, которые рассматривают ПО как «белый ящик», называют структурными методами.

Объектом тестирования в данном случаи является не внешнее, а внутреннее поведение ПО. Чаще всего анализируются управляющие связи элементов структуры.

Например, была ли выполнена каждая ветвь программы, либо каждый переход условного оператора был выполнен, либо был выполнен каждый путь программы, либо каждый оператор был выполнен хотя бы один раз. Тестирование по данному принципу характеризуется степенью, с которой тесты покрывают исходный текст программы. При использовании второго подхода исчерпывающее тестирование по критерию путей для всех входных данных также практически неосуществимо.

Анализ данных методов показывает, что исчерпывающее тестирование практически невозможно и представляет собой в значительной степени экономическую проблему. Поэтому на практике ограничиваются чем-то меньшим, то есть так проектируют тесты, чтобы они обеспечивали максимальную отдачу при ограниченных ресурсах (время, средства, люди).

Достоинством метода «белого ящика» является то, что метод «чёрного ящика» может скрыть проблемы, которые метод «белого ящика» отображает. Так, метод «чёрного ящика» может не сообщить о неправильном функционировании объекта, потому что проблемы в работе оказались незаметны. Метод «белого ящика» может обнаружить некорректный объект обеспечив прохождение вычислительного процесса по нужному пути исполнения кода за счёт выбора нужных тестовых данных.

Существует также метод «серого ящика», сочетающий в себе нечто среднее между методами «белого ящика» и «чёрного ящика».

### **Информационные потоки процесса тестирования ПО. Аксиомы тестирования ПО.**

На входе процесса тестирования находятся потоки:

1. Тестируемая программа;
2. Исходные данные (тесты);
3. Ожидаемые результаты.

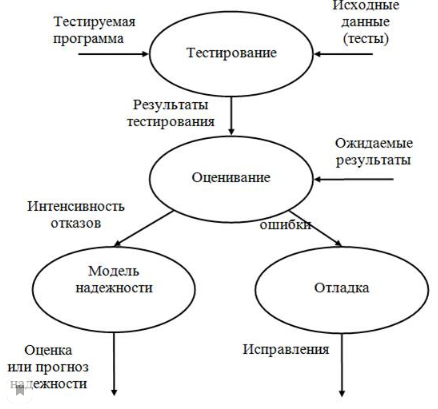


Рис 8.4. Информационные потоки процесса тестирования

Тесты выполняются, и все полученные результаты оцениваются, то есть реальные результаты сравниваются с ожидаемыми результатами.

Когда наблюдается несовпадение с ожидаемым результатом, то фиксируется отказ и начинается отладка, которая должна привести к исправлению ошибки, приведшей к данному отказу. Кроме того, фиксируется статистика о фактах отказов, которая позволяет оценить или спрогнозировать надежность разрабатываемого ПО.

**Принципы организации тестирования:**

1. Хорошим является то тест, у которого большая вероятность обнаружения ошибки, а не тот, который демонстрирует правильную работу ПО.
2. Нецелесообразно тестировать ПО своей собственной разработки, т.к. тестирование должно быть разрушительным процессом. Тестирование по возможности всегда должна выполнять внешняя группа (еще лучше внешняя организация), которая стоит отдельно от программистов и проекта.
3. Необходимо готовить тесты как для правильных (в соответствии спецификацией), так и не правильных исходных данных.
4. Одна из сложных проблем при тестировании – решить, когда нужно закончить тестирование. Необходимо разработать критерии, позволяющие обоснованно принять решение об окончании тестирования.
5. Необходимо детально изучать результаты каждого теста, чтобы не пропустить ошибку.
6. Любое тестирование должно быть воспроизводимым. Для чего тесты необходимо документировать и хранить в виде удобном для повторного использования.
7. Необходимой частью всего теста является заранее подготовленное описание выходных результатов. Лучше всего разрабатывать самопроверяющиеся тесты использующие, например, либо контрольное соотношение, либо попадание в заданную область, позволяющие автоматически сверять ожидаемые результаты с фактическими.
8. Нельзя изменять программу, чтобы облегчить ее тестирование. Программа должна тестироваться «как есть».
9. Тестирование, как почти всякая другая деятельность, должно начинаться с постановки целей. Как уже неоднократно отмечалось, тесты должны быть спроектированы, реализованы, проверены и, наконец, выполнены.

### **Структурное тестирование ПО. Понятие потокового графа, пути, базового пути, ветви, цикломатической сложности.**

Все методы тестирования ПО делятся на:

1. Методы структурного тестирования ПО;
2. Методы функционального тестирования ПО.

Структурное тестирование используется на ранних этапах тестирования, то есть на уровне тестирования модулей (7-я работа процесса разработки, см. подразд. 4.1). Функциональное тестирование используется практически во всех работах процесса разработки, связанных с тестированием (работы 7, 8, 9, 11 процесса разработки).

Структурное тестирование основано на понятиях:

1. Потокового графа;
2. Пути;
3. Цикломатической сложности ПО.

***Потоковый граф*** - структурная модель, показывающую связь между отдельными его элементами.

В общем случае граф

***G = (V, E)***

состоит из множества вершин ***V*** и множества дуг ***E:***

Дуги отображают потоки управления в программе (модуле), то есть передачи управления между операторами. Дугу можно представить в виде упорядоченной пары вершин



или записать в таком виде

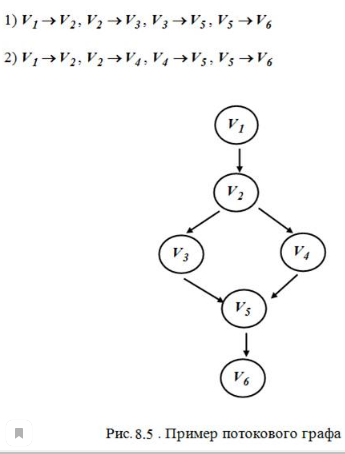
Различают ***операторные*** и ***предикатные вершины***. Из операторной вершины выходит только одна дуга, из предикатной – две.

Предикатные вершины соответствуют простым условиям (аналог – условный оператор (оператор ветвления) или оператор цикла в языках программирования). Операторной вершине соответствует или линейный участок программы или один оператор с одним входом и одним выходом.

Если представлять некоторую программу (модуль) в виде потокового графа, то

***Путь*** - последовательность вершин для которой существуют дуги ***V1 →Vk , Vk → Vl , …, Vi → Vj , …, Vm → Vn*** . Путь начинается в вершине ***V1***, проходит через вершины ***Vk, Vl*** ,**…, *Vi , Vj , …, Vm*** изаканчивается вершиной ***Vn*** . ***V1*** соответствует первому выполняемому оператору программы (модуля) и называется начальной вершиной графа. ***Vn*** соответствует последнему выполненному оператору программы (модуля) и называется конечной вершиной графа.

В нашем примере возможны пути, соответствующие следующим последовательностям дуг:



***Ветвь*** графа - часть пути, начинающаяся от начальной или предикатной вершины графа и заканчивающаяся предикатной или конечной вершиной графа.

Для графа, приведенного на рис. 8.5, можно выделить следующие ветви:

***1)*** ***1 – 2***

***2)*** ***2 – 3 – 5 – 6***

***3)*** ***2 – 4 – 5 – 6***

Одной из мер сложности ПО является *цикломатическое число*.

**Цикломатическое число** **(число МакКейба)** – структурная сложность графа по сравнению с линейным графом и позволяет оценить трудоемкость тестирования программы (модуля).

*Цикломатическое число* модуля можно определить *следующими способами:*

1. *Определение через количество вершин и количество связывающих вершины дуг графа*.

***Z(G)=E-V+2=6-6+2=2.***

В данном выражении ***Е*** – число дуг графа, ***V*** – число вершин графа. Для рис. 8.5 ***Е = 6***, ***V = 6***.

В выражение ***Z(G), Z*** обозначает цикломатическое число, а ***G*** обозначает, что сложность является функцией графа.

1. *Определение через количество регионов графа.*

***Z(G)=R=1+1=2,***

где ***R*** представляет собой число регионов графа.

***Регионы*** - замкнутые области на графе, образованные дугами и вершинами. Окружающая граф среда является дополнительным регионом. Например, для примера, приведенного на рис. 8.5, имеется один регион, образованный вершинами ***V2***, ***V3*** , ***V4*** , ***V5*** и связывающими их дугами. Дополнительным регионом является окружающая среда графа. Таким образом, общее количество регионов данного графа равно 2.

1. *Определение через количество предикатных вершин графа*.

Для предикатных вершин, из которых выходит две дуги, цикломатическая сложность

***Z(G)=P+1***,

где ***Р*** – число предикатных вершин.

**Цикломатическое число** определяет:

* Количество базовых независимых путей в модуле.
* Количество тестов, гарантирующее полное покрытие всех ветвей графа модуля.

***Базовый (Независимый***) путь – путь который вводит новый оператор обработки или новое условие, то есть независимый путь содержит дугу, не входящую в ранее определенные пути.

Все независимые пути графа образуют базовое множество.

### **Метод тестирования базовых путей. Тестирование циклов.**

Данный метод позволяет:

1) получить оценку сложности модуля;

2) на основе полученной оценки сложности определить необходимое количество тестов для определения набора базовых путей через тестируемый модуль. Полученные тесты гарантируют выполнение каждой ветви модуля или каждого оператора модуля хотя бы один раз.

***Суть метода (Том-Маккейб)*** заключается в следующем:

1. На основе схемы алгоритма модуля формируется потоковый граф.

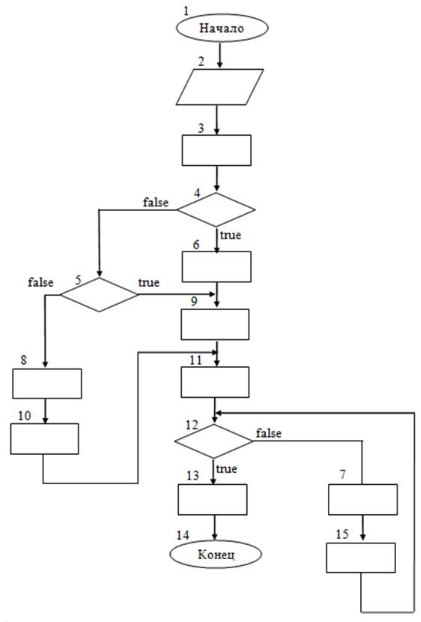


Рис. 8.9. Алгоритм работы модуля

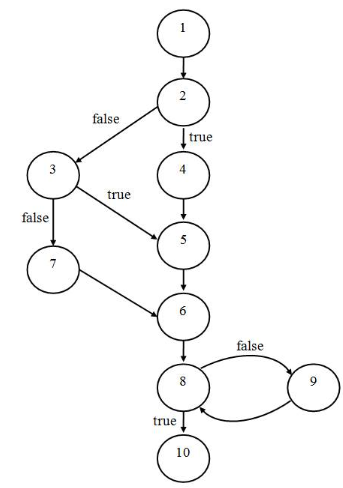


Рис. 8.10. Потоковый граф модуля

1. Определяется цикломатическая сложность потокового графа
2. Определяется базовое множество независимых путей. Для этого в начале выбирается любой один путь от входа в модуль к его выходу. Данный путь назовем *первым базовым путем*.

После того как все операторы в модуле выполнены в каждом направлении, набор тестовых путей будет закончен.

Достоинством метода является его независимость от вида потокового графа, т.е. метод применим как для циклических, так и для ациклических графов.

**Тестирование циклов**

Тестирование циклов производится по принципу “белого ящика”, при этом основное внимание уделяется правильности конструкции программы.

Возможны следующие ***типы циклов***:

1. *Простые*;
2. *Вложенные*;
3. *Неструктурированные* (go to).

Для проверки простых циклов с числом повторений ***n*** можно использовать следующий возможный набор тестов:

* ни одного прохода цикла;
* один проход цикла;
* ***m < n*** проходов цикла, где ***m*** – достаточно малое число проходов, если ***n*** достаточно велико;
* ***n*** – 1, ***n***, ***n*** + 1 проходов цикла, если ***n*** достаточно мало.

Если цикл простой и число повторений не определено, то должны быть тесты с однократным выполнением тела цикла и без выполнения тела цикла.

В случае вложенных циклов общее число путей может быть достаточно большим, поэтому для сокращения числа тестов применяется следующая *методика*:

1)вначале выбирается самый внутренний цикл и устанавливается минимальное значение параметров для всех охватывающих циклов;

2)для самого внутреннего цикла проводятся тесты простого цикла;

3)переходят в следующий по порядку объемлющий цикл и проводят с ним тесты простого цикла, при этом устанавливают минимальные значения параметров всех объемлющих и внутренних циклов;

4)шаги методики повторяют до тех пор, пока не будут протестированы все циклы программы.

Неструктурированные циклы тестировать нельзя, и они должны быть приведены к структурированному виду.

### **Функциональное тестирование ПО. Общие сведения. Цели функционального тестирования. Уровни функционального тестирования.**

Функциональное тестирование является основным видом тестирования ПО. Каждая функция программы тестируется и при этом делается вывод об ее правильности. Очевидно, что по всей области определения проверить функцию невозможно и поэтому каждая функция проверяется на правильность в некоторых точках области её определения.

Функциональное тестирование не является альтернативой структурному тестированию – это дополняющий подход, позволяющий обнаружить другие классы ошибок.

Например:

1. Обнаружение некорректных или отсутствующих функций;
2. Обнаружение ошибок интерфейса;
3. Обнаружение ошибок во внешних структурах данных (файлы, базы данных).

В отличие от структурного, функциональное тестирование используется на более поздних этапах тестирования.

В зависимости от того используются или нет средства автоматизации тестирования функциональное тестирование подразделяется на:

1. Ручное тестирование;
2. Автоматическое или автоматизированное тестирование (исп. инструментальных средств).

**Цели функционального тестирования**:

1. Поиск тестировщиками в тестируемом ПО ошибок и, чем больше будет их найдено и серьёзнее они будут, тем лучше. При этом очевидно, что даже для небольших программ провести 100% исчерпывающее тестирование невозможно в силу ограниченных компьютерных, материальных и временных ресурсов. Поэтому на практике тестирования всегда недостаточно, и в тестируемом ПО остаются не выявленные ошибки;
2. Документирование тестировщиками найденных ошибок с целью дальнейшего их исправления программистом. При этом необходимо, чтобы абсолютно каждая найденная ошибка была задокументирована;
3. Определение соответствия, тестируемого ПО предъявляемым к нему требованиям и принятие объективного заключения о возможности поставки протестированного ПО заказчику. Причём это заключение, как правило, делает менеджер проекта. При этом тестировщик путём предоставления максимально полной и объективной информации о качестве тестируемого ПО влияет на принятие данного заключения.

**Классификация функционального тестирования**:

1. Приёмочный тест (smoke test);
2. Критический тест (critical path test);
3. Расширенный тест (extended test).

Приёмочный тест является самым первым и коротким тестом, проверяющим работу основной функциональности тестируемого ПО. В зависимости от сложности ПО он длится от получаса до 2-3-х часов максимум, и затем принимается решение о целесообразности дальнейшего тестирования. При этом, если ПО не прошло приёмочный тест, то оно отправляется на доработку программистам, так как нет смысла выполнять дальнейшее тестирование.

Критический тест является основным по объёму тестирования видом теста. Во время критического теста проверяется основная функциональность тестируемого ПО, критичная для конечного пользователя при стандартном использовании ПО. В рамках данного тестирования проверяется большинство основных требований, предъявляемых к тестируемому ПО.

Расширенный тест представляет собой углубленный тест, при котором проверяется нестандартное использование тестируемого ПО. При этом прогоняются различные сложные, логически запутанные сценарии и совершаются действия, которые пользователь будет совершать редко. Расширенный тест используется в тех предметных областях, где риски велики и где отказ ПО может привести к большим финансовым потерям и(или) гибели людей. Протестированное ПО при этом должно работать надёжно в любой ситуации. Для тех предметных областей, где риски не велики, ПО может быть и не столь надёжным, и поэтому нет смысла тратить временные и материальные ресурсы на разработку и прогон сложных и запутанных тестовых сценариев.

**Методы тестирования ПО**:

1. Метод эквивалентного разбиения;
2. Метод анализа граничных условий.

### **Понятие класса эквивалентности. Примеры.**

Разбиение на классы эквивалентности (КЭ) представляет собой технологию проектирования тестов, ориентированную на снижение числа тестов необходимых для подтверждения корректности функциональных возможностей ПО. Основная идея, стоящая за разбиением на классы эквивалентности, заключается в том, чтобы разбить область ввода ПО на классы данных. При этом, очевидно, если проектировать тесты для каждого класса данных, но не для каждого элемента класса, то общее количество требуемых тестов уменьшится.

Принадлежность двух элементов данных к одному и тому же классу эквивалентности приводит к выбору общего набора операторов в тестируемом модуле.

Принадлежность двух элементов данных к различным классам эквивалентности означает, что существует по крайней мере одна строка кода, требуемая для обработки одного элемента данных, которая не будет использоваться при обработке другого элемента данных.

Если от выполнения двух тестов ожидается один и тот же результат, то с точки зрения надёжности ПО эти тесты считаются **эквивалентными**.

В общем группа тестов представляет собой класс эквивалентности, если выполняются следующие условия:

1. Все тесты предназначены для выявления одной и той же ошибки;
2. Если один из тестов выявляет ошибку, то остальные тесты скорее всего выявят эту же ошибку (с высокой вероятностью);
3. Если один из тестов не выявит ошибку, то и остальные тесты скорее всего не выявят эту ошибку (с высокой вероятностью).

Классы эквивалентности выделяются путем выбора каждого входного условия.

В общем, поиск КЭ – процесс субъективный. Два тестировщика, анализирующих одну и ту же программу составят различные КЭ. При этом, надо выявить как можно больше КЭ, так как это сделает тестирование более эффективным.

Нужно спроектировать тесты, которые выполняли бы проверку по крайней мере по одному представителю каждого **правильного класса эквивалентности (ПКЭ)** и **неправильного класса эквивалентности (НКЭ)**. В случае ввода недопустимых данных должно быть выдано соответствующее сообщение об ошибке, если оно определено в требованиях спецификации. При этом, при вводе недопустимых данных (НКЭ) программа, по меньшей мере, не должна завершиться аварийно, вызывать искажения данных или вести себя непредсказуемым образом.

### **Метод эквивалентного разбиения. Метод анализа граничных значений. Примеры.**

В соответствии с данным методом проектирование тестов проводится в *два этапа*.

***На первом этапе*** в соответствии со спецификацией область входных данных делится на классы эквивалентности. Для каждого входного условия из спецификации формируется два или более класса эквивалентности. Это так называемые *ПКЭ (правильный класс эквивалентности)* и *НКЭ (неправильный класс эквивалентности)*.

***На втором этапе*** осуществляется собственно проектирование тестов с помощью сформированных классов эквивалентности. Для ПКЭ тесты проектируются таким образом, чтобы каждый тест покрывал как можно больше еще не покрытых ПКЭ. Процесс продолжается до тех пор, пока не будут покрыты все ПКЭ.

Затем проектируются тесты для каждого НКЭ, при этом они проектируются таким образом, чтобы каждый тест покрывал один и только один НКЭ. Процесс продолжается до тех пор, пока не будут покрыты все НКЭ, но только не общими тестами.

#### **Метод анализа граничных значений**

На практике ошибки ПО часто проявляются на границах области ввода.

Под граничными значениями понимаем ситуацию, возникшую на границе определенного спецификацией входного или выходного условия.

**Суть метода** заключается в выполнении следующих правил:

**1-е правило.** Если входное условие, то есть условие ввода, задает диапазон чисел ***X* [*1, 100*],** то необходимо создать тесты:

1. Для значений ***1*** и ***100***;
2. Для значения чуть левее ***1*** и чуть правее ***100***.

**2-е правило.** Если условие ввода задает дискретное множество значений, то необходимо создать тесты:

1. Для проверки минимального и максимального значений;
2. Для значения чуть меньше минимума и чуть больше максимума.

**3-е правило.** Ранее названные правила 1 и 2 применяются к условиям области вывода.

**4-е правило.** Если внутренняя структура данных программы является упорядоченным множеством (массив, список), то разрабатываются тесты, проверяющие эти структуры на границах.

Анализ граничных значений отличается от эквивалентных разбиений двумя моментами:

1. Выбор любого элемента в классе эквивалентности в качестве представительного при анализе граничных значений осуществляется таким образом, чтобы проверить тестом каждую границу этого класса;
2. При разработке тестов рассматривают не только входные условия (пространство входов), но и выходные КЭ (пространство результатов).

### **Документирование тестов. Структура Test Case. Примеры.**

Под тест-кейсом понимается:

1. Набор тестовых входных данных,
2. Условий выполнения,
3. Ожидаемых результатов,

Разработанных с целью проверки некоторого пути выполнения программы или проверки соответствия некоторому требованию.

Тест-кейс представляется в виде документа, который может включать следующие поля:

1. Идентификатор тест-кейса;
2. Связанные с тест-кейсом требования;
3. Приоритет тест-кейса;
4. Модуль и подмодуль приложения, к которым относится тест-кейс;
5. Название тест-кейса;
6. Исходные данные, необходимые для тест-кейса;
7. Шаги для выполнения тест-кейса;
8. Описание ожидаемых результатов по каждому шагу выполнения тест-кейса;
9. Поле статуса об успешном или ошибочном выполнении тест-кейса;
10. Автор тест-кейса;
11. Время последнего выполнения тест-кейса;
12. Последний полученный результат;
13. Связанный с тест-кейсом дефект(баг), если он есть.

***Достоинствами использования тест-кейсов*** является то, что они:

1. Дают структурированный системный подход;
2. Представляют собой один из способов тестирования проектной документации ещё до появления первой промежуточной версии программного продукта;
3. Значительно ускоряют процесс регрессионного тестирования;
4. Позволяют отслеживать статистику проводимого тестирования;

### **Методика тестирования ПО. Тестирование модулей.**

В соответствии с *СТБ ИСО/МЭК 12207-2003* процесс разработки программных средств включает 13 работ. Из них тестированию соответствуют 7, 8, 9, 11-я работы.

**Методика тестирования ПС** включает следующую *последовательность шагов*:

1. ***Тестирование модулей*** (7-я работа).
2. ***Тестирование сборки*** (интеграции) (8-я работа).
3. ***Тестирование правильности*** (9-я работа).
4. ***Системное тестирование*** (11-я работа).

Процесс тестирования как правило начинается с тестирования отдельных модулей, а затем продолжается при объединении модулей в единое целое и завершается тестированием, при котором проверяется соответствие разработанного ПС требованиям заказчика.

Методика тестирования ПС может быть представлена в виде разворачивающейся спирали.

На первом шаге осуществляется тестирование модулей, проверяющее результаты этапа кодирования. Целью данного шага является проверка каждого отдельного модуля. Используется как структурное, так и функциональное тестирование.

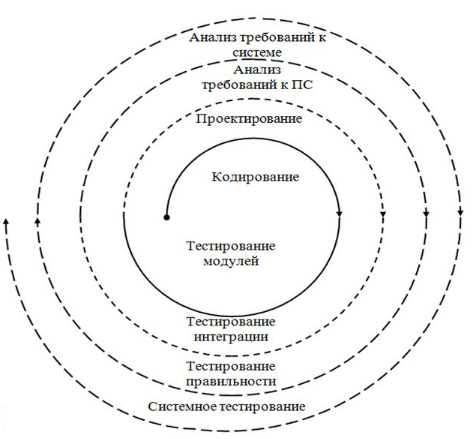


Рис. 8.14. Методика тестирования ПС

На втором шаге выполняется тестирование интеграции, направленное на выявление ошибок этапа проектирования ПС. Целью второго шага является сборка модулей в единое ПС и его тестирование. В основном используется функциональное тестирование и частично структурное тестирование.

На третьем шаге производится тестирование правильности, проверяется правильность этапа анализа требований к ПС. Целью третьего шага является проверка правильности реализации в ПС всех функциональных требований, а также требований по качеству к ПС. Проводится только функциональное тестирование.

На четвёртом шаге проводится системное тестирование, проверяется правильность этапа анализа требований к системе. Целью четвёртого шага является проверка реализации в ПС всех системных требований, а также требований по качеству к системе. Проводятся различные типы системного тестирования.

Тестирование модулей

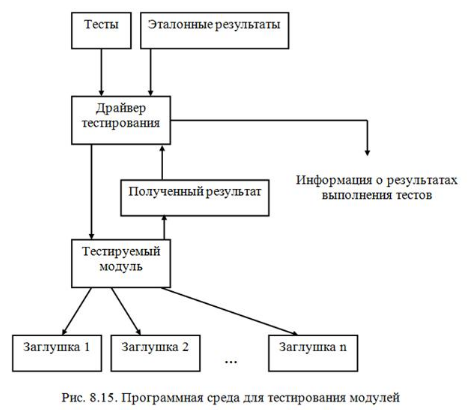
Проводится во время работы № 7 «Программирование и тестирование ПС» процесса разработки (см. подраздел 4.1). На данном шаге проверяется правильность функционирования наименьшего отдельно взятого элемента ПС. Что взято в качестве элемента (модуля) определяется контекстом проекта. Это может быть либо отдельно взятая функция, либо набор функций, либо отдельно взятый класс, либо набор классов и т.п. [4]

Вначале, с целью проектирования тестов, рекомендуется рассматривать модуль ПС как “чёрный ящик”, а затем для подготовки дополнительных тестов исследовать модуль как “белый ящик”.

**Методика тестирования модулей** включает следующие *шаги*:

1. Руководствуясь спецификацией требований модуля, необходимо подготовить тесты для каждого входного условия (используются методы эквивалентного разбиения и граничных условий);
2. Проверить текст модуля, чтобы убедиться, что все условные переходы будут выполнены в каждом направлении, т.е. каждая ветвь модуля будет выполнена. При необходимости спроектировать дополнительные тесты;
3. Проверить по тексту модуля тот факт, что тесты охватывают достаточное количество путей (для каждого цикла должны быть составлены пути без выполнения тела цикла, с однократным и максимальным числом повторений;
4. Проверить по тексту модуля его чувствительность к отдельным особым значениям входных данных и при необходимости спроектировать дополнительные тесты;
5. Следует также спроектировать тесты, пытаясь выйти за границы упорядоченных структур данных (массивов, списков, файлов и т.д.).

Так как модуль является частью всего ПС, то для проведения его тестирования необходимо дополнительное ПС - драйвер и заглушка).



*Драйвер* представляет собой управляющую программу, которая:

1. Принимает тесты и ожидаемые результаты тестов (эталонные результаты);
2. Осуществляет запуск в работу тестируемого модуля;
3. Получает из модуля реальные результаты выполнения модуля;
4. Формирует информацию о результатах тестирования.

*Заглушки* представляют собой фиктивные программы, замещающие модули, которые вызываются тестируемым модулем. Заглушка реализует интерфейс подчиненного модуля и может осуществлять минимальную обработку данных и имитировать прием и возврат данных.

В общем случае не все модули могут быть адекватно протестированы с помощью драйвера и заглушки. В таких случаях окончательное тестирование модуля может быть отложено до шага тестирования интеграции (сборки).

### **Тестирование сборки (интеграции).**

Тестирование сборки проверяет правильность взаимодействия между программными компонентами или модулями. Проводится во время работы № 8 «Сборка ПС» процесса разработки. В данной работе разрабатывается план сборки для объединения модулей и компонентов в единый программный объект.

План включает требования к тестированию, а также процедуры и методы тестирования. План должен быть документально оформлен. Затем, в соответствии с планом, модули и компоненты собираются в единое целое и тестируются. Выбор варианта тестирования сборки определяется той стратегией сборки, которая была выбрана при проектировании ПС.

**Стратегии сборки**:

1. Сборка модулей и компонент по мере движения по управляющей иерархии “снизу-вверх” (восходящее тестирование сборки, bottom-up);
2. Сборка модулей и компонент по мере движения по управляющей иерархии “сверху-вниз” (нисходящее тестирование сборки, top-down).

При выполнении данной работы в основном находятся ошибки интерфейса, например,:

1. Потеря данных при прохождении через интерфейс;
2. «Неблагоприятное» влияние одного модуля на другой;
3. Проблемы при работе с глобальными переменными.

### **Тестирование правильности. Системное тестирование. Типы системного тестирования.**

**Тестирование правильности**

На данном этапе (работа № 9 процесса разработки по стандарту СТБ *ИСО/МЭК 12207-2003*) проводится тестирование на соответствие требований ко всему программному объекту. Данные требования прописаны в спецификации требований к ПС (работа № 4). Используется только функциональное тестирование, которое должно подтвердить, что функции, прописанные спецификацией, работают правильно.

В общем, разработчик не может знать, как заказчик будет реально использовать разработанное ПС. Для обнаружения дополнительных ошибок, которые может найти заказчик или конечный пользователь, используется Альфа- и Бета-тестирование.

**Альфа-тестирование** проводится заказчиком или конечным пользователем в организации разработчика с документацией всех выявленных ошибок. Это могут быть сотрудники организации разработчика, но не участвовавшие в разработке ПС.

**Бета-тестирование** проводится конечным пользователем, а также заказчиком. Заказчик проводит тестирование в своей организации, при этом фиксируются и документируются все ошибки. Затем по результатам всех выявленных ошибок разработчик корректирует ПС, повторно его тестирует и (при положительных результатах тестирования) отдает его в эксплуатацию заказчику или конечному пользователю

**Системное тестирование**

В работе № 10 процесса разработки (Сборка сист., см. подраздел 4.1) объекты программной конфигурации собираются в единую систему вместе с объектами технической конфигурации и при необходимости с другими системами. Система тестируется, начиная с работы № 11, и проводится в соответствии с требованиями, установленными к системе. При этом должна быть обеспечена проверка каждого требования к системе на соответствие заданным требованиям. Только при системном тестировании можно оценить характеристики ПС, которые нельзя оценить на более ранних этапах тестирования.

**Основные** **типы системного тестирования**:

1. *Тестирование производительности*. Существует огромное количество систем реального времени, в которых должны жестко выполняться требования по производительности. Например, сист., управляющие технологическими процессами, обучающие программы, игры и т.п. Только системное тестирование проверяет истинную производительность всей сист. в целом.
2. *Тестирование надежности*. Проверяется правильность работы всей сист. при заданных условиях для заданного интервала времени. Надёжность можно измерить метриками:

* средняя наработка на отказ;
* вероятность безотказной работы и т.п.

1. *Тестирование восстановления*. Проверяется отказоустойчивость сист. Система должна быть быстро восстанавливаема после сбоев и отказов. При этом отказы не должны быть причиной прекращения работы всей сист., т. к. отказ для отдельных систем может привести к огромному ущербу.
2. *Стрессовое тестирование*. Стрессовые тесты проектируются таким образом, чтобы проверить работу всей сист. в ненормальных ситуациях. Например, при ненормальных запросах на ресурсы сист.:

* по количеству запросов;
* по частоте запросов;
* по размеру используемых ресурсов и т.п.

Целью стрессового тестирования является нахождение ошибок, появление которых спровоцировано дефицитом ресурсов, т.е. нехваткой свободной оперативной памяти или свободного места на внешнем носителе, или нехваткой пропускной способности сети.

1. *Тестирование безопасности*. При этом можно подвергнуть систему попыткам НСД (несанкционированного доступа) и при этом измерять среднее время взлома. Взлом лучше всего организовать с помощью внешних средств.
2. *Тестирование практичности*. Оценивается “дружелюбие'' разработанного ПС к пользователю и фиксируются те операции, которые могут вызвать трудности у пользователя.

Тесты практичности включают тесты на:

* Наличие и качество оперативной и контекстной помощи;
* Наличие руководств и учебных материалов;
* Проверку на соответствие общепринятым стандартам графического интерфейса и т.п.

### **Регрессионное тестирование.**

Основной работа тестировщика.

Оно основано на повторном использовании ранее разработанных тестов.

Тестировщик после обнаружения ошибки передает отчет о ней программисту. В отчете приводятся симптомы проявления ошибки. Программист анализирует отчет, исправляет данную ошибку и отдает исправленное ПС назад тестировщику. Тестировщик затем опять проверяет ПС на том же самом тесте, который ранее приводил к ошибке, так как должен убедиться в том, что ошибка исправлена и больше не проявляется [25].

Реально программист может исправить только описанные в отчете симптомы, и фактически могут исправляться только отдельные причины проявления ошибки. Кроме того, программист может неправильно понять отчет об ошибке и исправить не то, что надо. Таким образом, после исправления одни ошибки могут устраняться, а другие оставаться.

Статистика показывает, что в среднем каждое третье исправление ошибки приводит к внесению новых ошибок. Поэтому очевидно, что после исправления программистом каждой ошибки, тестировщик должен спроектировать дополнительные тесты. Целью данных тестов является проверка того, чтобы убедиться, что при исправлении ошибки не внесены новые ошибки. Для этого необходимо проанализировать те части ПС, которые могут быть затронуты внесением изменений.

Кроме того, рекомендуется проверить целостность всего ПС, то есть правильность его работы, на следующих тестах:

* Перечисленные тесты в начальном плане тестирования;
* Тесты, которые выявляли все предыдущие обнаруженные ошибки;
* Тесты, сгенерированные случайным образом.

*Таким образом, повторное выполнение тестов (регрессионное тестирование) необходимо для проверки того, что изменения, внесенные в программу в результате устранения ошибок или изменения существующей функциональности, не повлияли на правильность функционирования тестируемой программы.*

### **Документирование ошибок. Структура отчета об ошибке.**

**Программная ошибка/баг** - некоторый дефект в разработке программного продукта, который вызывает несоответствие ожидаемых результатов выполнения программного продукта и фактически полученных результатов.

**Задокументировать ошибку** может практически любой разработчик или пользователь, обнаруживший некорректное поведение программы. Но собственно написание отчёта об ошибке («bugreport») является одним из основных результатов работы тестировщика. Основная цель написания отчёта об ошибке - устранение ошибки.

Цели отчёта об ошибке:

* Предоставить информацию о возникшей проблеме, её свойствах и последствиях;
* Назначить приоритет возникшей проблеме с учётом её важности и скорости устранения;
* Помочь программистам обнаружить и устранить источник проблемы.

Для формирования отчёта об ошибке используются так называемые багтрэкинговые сист.

Часто данные сист. интегрируются с системами управления проектами, так как ошибка в разрабатываемом приложении является угрозой для качественного выполнения проекта в целом.

**Основные поля**:

1. **Краткое описание**(Summary). Содержит краткое описание ошибки, возникшей в приложении. Чаще всего указывается место, где произошла ошибка, и вкратце её суть.
2. **Версия приложения**, в котором была обнаружена ошибка (Build found).
3. **Симптом** (Symptom) ошибки. Это поле показывает вид ошибки, т.е. как она проявляется.

*Наиболее широко распространённые симптомы:*

* Некорректная операция (incorrect operation). Например, неверная арифметическая операция;
* Повреждение/потеря данных (data corruption/loss). В результате ошибки данные повреждаются или теряются.
* Крах сист. (system crash). Например, приложение или операционная система виснет, перезагружается или закрывается.
* Косметический дефект (cosmetic flaw). Например, опечатки, повреждённые картинки, не тот цвет и т.п.
* Проблема в документации (documentation issue). Если ошибка описывает проблему не в приложении, а в документации.
* Нереализованная функциональность (missing feature).

1. **Серьёзность** (Severity) ошибки. Это поле показывает, насколько серьёзна найденная ошибка.

Обычно выделяют следующие уровни серьёзности:

- критическая (critical) ошибка. Это самые серьёзные ошибки, ведущие к краху приложения или операционной сист., серьёзному повреждению базы данных, падению веб-сервера или сервера приложений;

- высокая (major) ошибка. Это также серьёзные ошибки, ведущие к потере/искажению данных, падению значительной части функциональности приложения, падению браузера и т.п.;

- средняя (medium) ошибка. Это ошибки, которые не приводят к краху сист, или потере данных. Они обычно затрагивают небольшой набор функциональности, чаще всего это некорректная обработка вводимых значений. Как правило, такие ошибки можно «обойти», т.е. выполнить требуемое действие иным способом, не приводящим к возникновению ошибки;

-низкая (minor) ошибка. Как правило, это ошибки «косметического» плана. Они непосредственно не мешают работе с приложением.

1. **Приоритет** (Priority) ошибки. Это поле показывает, как быстро необходимо исправить ошибку.

Обычно выделяют следующие уровни приоритета:

- наивысший (ASAP, as soon as possible). При этом ошибка должна быть исправлена немедленно, так как её наличие делает невозможным дальнейшую работу над проектом;

- высокий (high). Обычно присваивается ошибкам, которые нужно исправить как можно быстрее;

- обычный (normal). Ошибка должна быть исправлена в ближайшем будущем после исправления всех ошибок с более высоким приоритетом;

- низкий (low). Это самые низкоприоритетные ошибки, которые исправляются в последнюю очередь.

Если серьёзность ошибки выставляет тестировщик, то приоритет ошибки выставляет менеджер проекта после просмотра отчёт об ошибке.

1. **Воспроизводимость** (Reprodusible) ошибки. Это поле показывает, как воспроизводится ошибка. Возможны два уровня воспроизводимости ошибки:

-всегда (always), когда ошибка воспроизводится постоянно;

-иногда (sometimes), когда ошибка воспроизводится при определённых условиях.

1. **Шаги** для воспроизведения ошибки (Steps to reproduce). Данное поле приводится для того, чтобы воспроизвести ошибку в кратчайшие сроки. Это поле следует заполнять максимально подробно.
2. **Подробное описание ошибки** (Description). Приводится для того, чтобы программист чётко понял суть проблемы. Чем подробнее описание, тем быстрее причина ошибки будет обнаружена и тем быстрее ошибка будет исправлена.

Кроме словесного описания в поле Discription тестировщик может «прикрепить» Screen Shot, изображающий полученную ошибку.

Хорошее описание содержит необходимую информацию об ошибке, описание ожидаемого результата, описание полученного результата и ссылку на требование.

### **Жизненный цикл дефекта.**

Баг может находится в одном из представленных на рисунке состояний.

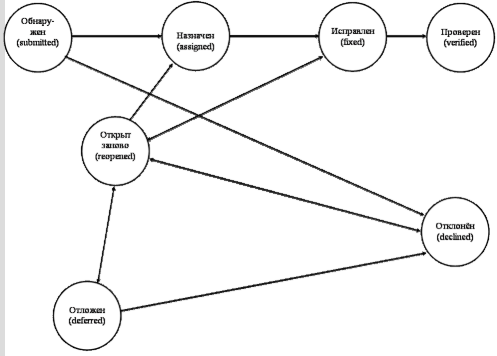


Рис. 8.19. Жизненный цикл ошибки

После обнаружения тестировщиком дефекта он вносится в баг-трэкинговую систему и ему присваивается статус **Обнаружен (submitted)**.

Далее ведущий разработчик, например, менеджер, рассматривает дефект, присваивает ему приоритет и назначает его исправление кому-то из команды разработчиков (программистов) – дефекту присваивается статус **Назначен (assigned).**

Разработчик, которому было назначено исправление дефекта, исправляет его и сообщает о том, что задание выполнено. Дефекту присваивается статус **Исправлен (fixed).**

Тестировщик, который обнаружил ошибку, проверяет на новой версии продукта (в которой исправление данной ошибки заявлено), исправлен ли дефект на самом деле. И только в том случае, если ошибка не проявится в новой версии, тестировщик меняет статус бага на **Проверен (verified)** и закрывает дефект.

Если баг проявляется в новой версии, тестировщик снова открывает этот дефект. Баг приобретает статус **Открыт заново (reopened).**

Баг может быть отклонён. Во-первых, потому, что для заказчиков какие-то ошибки перестают быть актуальными. Во-вторых, это может случиться по вине тестировщика из-за плохого знания продукта, требований (дефекта на самом деле нет). Дефекту присваивается статус **Отклонён (declined).**

Если исправление конкретного бага сейчас не очень важно или заказчик пока думает, или мы ждём какую-то информацию, от которой зависит исправление бага, тогда баг приобретает статус **Отложен (deferred).**

**Закрытым** считается баг в состоянии **Проверен (verified)** и **Отклонён (declined)**.

**Открытыми** являются баги в состоянии **Обнаружен (submitted), Назначен (assigned), Открыт заново (reopened)**. Иногда к открытым относят и баги в состояниях **Исправлен (fixed)** и **Отложен (deferred).**

### **Проектирование тестов, основанных на случайных наборах исходных данных. Получение закона распределения по заданным моментам исходного распределения.**

В процессе разработки интерес представляет оценка надежности промежуточного продукта. На ранних этапах разработки это может быть спецификация требований, архитектура или технический проект программного продукта, исходные коды модулей. Для них выполняется оценка надежности по внутренней модели надежности с целью прогноза внешнего уровня надежности.

На последующих этапах процесса разработки промежуточными продуктами являются исполнимые коды модулей и промежуточных продуктов сборки, а также конечный программный продукт. Для них выполняется оценка надежности по внешней модели в моделируемой среде с моделируемыми исходными данными с целью прогноза уровня надежности программного продукта в среде эксплуатации. При этом моделируемые данные должны быть адекватными реальным данным в заданных условиях эксплуатации.

*Метод моделирования случайных наборов данных основан на получении закона распределения в виде функции плотности.*

**Получение закона распределения по заданным моментам исходного распределения**

Методы генерации значений случайных величин используют заданный закон распределения в виде известной функции распределения или функции плотности. Но часто аналитический вид функции распределения или функции плотности неизвестен, а известен ряд моментов исходного распределения (или некоторая ограниченная *выборка*). В этом случае плотность или функцию исходного распределения можно представить в виде разложения по полиномам.

Наиболее общим способом представления скалярных и векторных случайных величин является представление их в форме линейных комбинаций некоторых эталонных функций.

Разложение искомой функции плотности обычно ищут в виде ряда



где: ***Сn*** - некоторый коэффициент разложения; ф- весовая (эталонная) функция плотности; ***Qn(х)***- система ортогональных полиномов.

Выбор определяется характером исходного распределения. Если распределение исходных данных на заданном интервале близко к нормальному, то хорошие результаты дает разложение, основанное на нормальном законе. В качестве обычно берется нормальная функция плотности нормированной случайной величины (- для удобства расчета)



В качестве ***Qn(x)***используется ***полином Эрмита n-порядка***, который определяется как

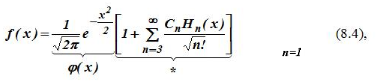


где:

***n*** = 0, 1, 2, ...

производная ***n***-порядка

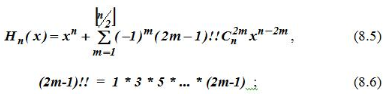
Выражение аппроксимирующей функции (8.1) имеет вид



где:

\* - ряд, учитывающий асимметрию распределения;

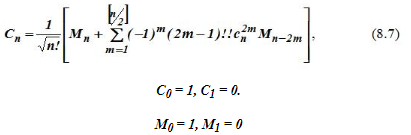
***Hn(х)***- полином Эрмита ***n***-порядка, определяемый как



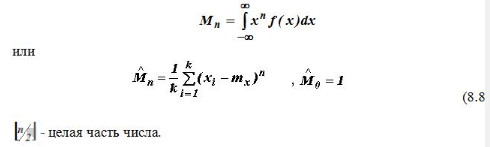
в выражении (8.5) число сочетаний из ***n*** по ***2m*** определяется формулой:



***Cn*** - *коэффициент разложения*, определяемый как



В выражении (8.7) ***Mn*** *- центральный момент* ***n****-порядка:*

**

Обычно ряд (8.4) называется *рядом Грама-Шарлье типа А*. В том случае, когда моменты исходного распределения не заданы, а известна выборка значений исходных распределений, в качестве центральных моментов берутся их оценки (8.8).

Из анализа выражения (8.7) для вычисления ***Cn*** видно, что необходимо знание центральных моментов исходного распределения до ***n***-го порядка включительно.

Вопрос о сходимости ряда (8.4) не имеет значения; важно быть уверенным, что первые слагаемые данного ряда дают достаточно точное приближение.

Как показывает анализ литературы, аппроксимация исходного распределения с помощью *ряда Грама-Шарлье типа А* тем лучше, чем ближе исходное распределение к нормальному.

### **Проектирование тестов, основанных на случайных наборах исходных данных. Генерация случайных значений исходных данных.**

Процедура генерации случайных значений в соответствии с заданной функцией плотности ***f(x)*** заключается в следующем.

Функция плотности ***f(x)*** аппроксимируется ступенчатой функцией и при этом отрезок **[*a, b*]** разбивается на ***n*** отрезков (рис. 8.20).

1)Генерируется равномерно распределенное случайное число



и определяется число**,** равномерно распределенное на отрезке [a, b]:

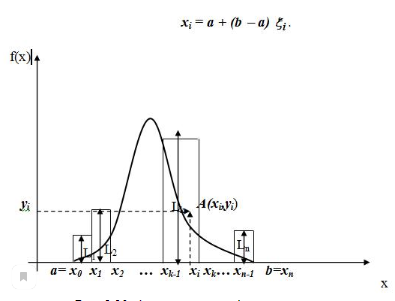


Рис. 8.20. Аппроксимация функции плотности

2) Значение индекса ***k*** соответствует попаданию случайного числа ***xi*** в некоторый интервал



значение ***Lk*** равно высоте выбранной ступеньки.

3) Генерируется следующее случайное число, которое принадлежит отрезку [0, 1],



и определяется случайное число



4) Если точка ***А*** с координатами ***А (xi, yi)*** находится под кривой ***f(x)***, то ***xi*** является реализацией случайной величины ***X***, имеющей заданное распределение ***f(x).*** А если точка ***А*** с координатами ***А (xi, yi)***находится над кривой ***f(x)***, то ***xi*** не является реализацией случайной величины ***X***, имеющей заданное распределение ***f(x).***

5) Шаги 1, 2, 3, 4 повторяются до тех пор, пока не получится требуемое число реализаций случайной величины ***X***, имеющей заданное распределение ***f(x).***

### **Понятие верификации. Понятие высказывания, высказывательной формы и предиката. Операция импликации. Примеры.**

**Верификацией** - подтверждение экспертизой и представлением объективных доказательств того, что конкретные требования полностью реализованы

*Верификацией* - процесс проверки правильности какой-то работы ЖЦ.

Чем раньше начать верификацию работ, тем выше будет качество будущего ПО, тем раньше будут найдены все дефекты при разработке ПО.

Верификация программ состоит в формальном доказательстве их правильности. В отличие от тестирования, имеющего дело со свойствами отдельных процессов программы, верификация имеет дело со свойствами всей программы и для всей совокупности исходных данных.

**Элементы математической логики**

**Исчисление высказываний**

***Исчисление высказываний*** - это совокупность правил для определения истинности или ложности высказываний.

***Высказывание*** - это предложение, которое либо истинно, либо ложно.

Предложение ***x2 = 4***, например, не является высказыванием, т.к. чтобы говорить об истинности или ложности высказывания, нужны дополнительные сведения (чему равно число ***x***). В этом предложении ***x*** - некоторая переменная, вместо которой можно подставить элементы некоторого множества, называемые значениями этой переменной.

Предложение, которое содержит хотя бы одну переменную и становится высказыванием при подстановке вместо всех переменных их значений, называется ***высказывательной формой*** (**ВФ**).

В математической логике всякое предложение, составленное из высказываний с помощью логических связок (и; или; если то; тогда и только тогда, когда и т.п.), становится высказыванием.

**Операции над высказываниями**

1. Конъюнкция
2. Дизъюнкция
3. Отрицание
4. Импликация

Определение импликации соответствует употреблению союза "*если-то*" не только в математике, но и в обыденной речи. По определению логической операции смысл составляющих высказываний не учитывается. Составляющие высказывания рассматриваются как объекты, обладающие единственным свойством: *быть истинными или ложными*.

**Исчисление предикатов**

Исчисление предикатов является расширением логики высказываний.

Рассмотрим высказывательную форму ***sin(x)=1.*** Данная форма каждому ***x*** на множестве действительных чисел ставит в соответствие некоторое высказывание и тем самым одно из значений истинности. Таким образом, данная высказывательная форма задаёт отображение множества действительных чисел ***R*** на множество значений {***Истина, Ложь***}, то есть задаёт функцию с областью определения ***R*** и множеством значений ***{Истина, Ложь}***.

Функция, все значения которой принадлежат множеству ***{Истина, Ложь},*** называется ***предикатом***. Чаще всего предикаты задаются с помощью высказывательных форм.

Например:

1. Одноместная высказывательная форма,
2. Предикат с двумя неизвестными,
3. Трехместная высказывательная форма.

**Операции над высказывательными формами**

**Конъюнкцией** высказывательных форм ***Ф1*** и ***Ф2*** называется высказывательная форма, истинная при тех и только при тех значениях, входящих в неё переменных, которые обращают обе формы в истинное высказывание.

**Дизъюнкцией** высказывательных форм ***Ф1*** и ***Ф2*** называется высказывательная форма, ложная при тех и только при тех значениях, входящих в неё переменных, которые обращают обе формы в ложное высказывание.

**Отрицанием** высказывательной формы ***Ф*** называется высказывательная форма, ложная при тех наборах значений переменных, которые обращают высказывательную форму в истинное высказывание и наоборот.

**Импликацией** высказывательных форм ***Ф1*** и ***Ф2*** называется высказывательная форма ***Ф1Ф2***, ложная при тех и только при тех значениях, входящих в неё переменных, которые обращают ***Ф1*** в истинное высказывание, а ***Ф2*** - в ложное.

### **Метод индуктивных утверждений. Общие сведения.**

***Верификация*** ПО заключается в выполнении формального доказательства того, что ПО удовлетворяет своей спецификации.

Общими для различных методов ***верификации*** ПО является представление программ в виде схемы алгоритма, с каждой дугой которой соотносится некоторый предикат, называемый утверждением.

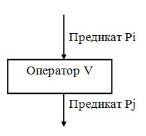


Рис. 9.1. Пример фрагмента схемы алгоритма с предикатами

Если ***Pi***есть утверждение, связанное с входящей дугой оператора ***V,*** а ***Pj*** – утверждение, связанное с исходящей дугой того же оператора, то тогда необходимо доказать правильность следующего оператора: если ***Pi*** истинно и если оператор ***V*** выполнен, то утверждение ***Pj*** истинно. Такие теоремы имеют следующий вид: ***Pi→Pj***.

Обычный метод их доказательства сводится, например, к тому, чтобы показать, что***Pj*** истинно всякий раз, когда ***Pi***истинно, т.е. для доказательства всей теоремы необходимо доказать, например, истинность ***Pj***. Подобный процесс может быть повторен для каждого оператора программы.

На рис. 9.2 приведена схема алгоритма некоторой программы с предикатами ***P1*** и ***Pn***.

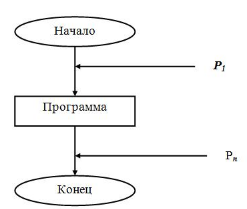


Рис. 9.2. Пример схемы алгоритма некоторой программы с предикатами

На данном рисунке ***P1*** – утверждение, непосредственно предшествующее входному оператору программы (начальное утверждение), а ***Pn*** – утверждение, соответствующее выходному оператору программы (конечное утверждение).

Тогда оператор «если ***P1*** истинно и программа выполнена, то ***Pn*** истинно», представляет собой теорему, доказательство которой устанавливает правильность того, что ПО соответствует своим спецификациям (т.е. теорема имеет вид ***P1→Pn***). Таким образом, доказательство правильности программ сводится к доказательству теорем методами исчисления предикатов.

Одним из известных методов доказательства правильности программ является *метод индуктивных утверждений*, предложенный Флойдом.

Данный метод применим для доказательства правильности или корректности схем алгоритмов.

**Аспекты понятия корректности программ:**

1. Частичная корректность.
2. Завершаемость.

***Частичная корректность*** - соответствие программы своей спецификации.

***Завершаемость -*** выполнение программы закончится при вводе определенных данных в соответствии со спецификацией. Нет общего метода доказательства завершаемости программ, но обычно бывает достаточно неформальных рассуждений.

В методе индуктивных утверждений первый шаг состоит в записи утверждений относительно свойств входных и выходных данных программ, а также результатов в ряде промежуточных точек называемых точками разреза. Эти утверждения формулируются в логике исчисления предикатов. На основании этих утверждений и семантики операторной схемы программы путем определенных преобразований формулируются верификационные условия, правильность которых необходимо доказать. И если доказывается истинность сформулированных условий, то программа будет корректна относительно входного и выходного утверждений. Если доказать истинность не удается, то либо в ПС есть ошибка, либо в процедуре доказательства есть ошибка, например, неверное утверждение в некоторой точке разреза.

### **Процесс верификации в жизненном цикле ПО.**

**Процесс верификации** **(The Verification Process)** - процесс определения того, что программные продукты функционируют в полном соответствии с требованиями и условиями, реализованными в предшествующих работах.

Верификация может применяться к любым промежуточным программным продуктам и к любым другим результатам работы. Например, верификации могут подвергаться требования, системная и программная архитектура, документация, методы, планы и т.п.

Процесс верификации может включать анализ, проверку и тестирование. Объектами анализа и проверки могут являться, например, документация и исходные тексты программных модулей. Объектами тестирования могут являться исполнимые коды программных модулей, компонентов, промежуточных и конечного программных продуктов.

**Процессом независимой верификации** - если организация–исполнитель не зависит от поставщика, разработчика, оператора или персонала сопровождения.

Процесс верификации состоит из *двух работ* (рис. 9.5). Общее число задач по данным работам равно 13.

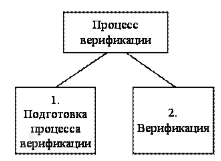


Рис. 9.5. Структура процесса верификации

**Задачи работ процесса верификации:**

* ***Подготовка процесса верификации:***
  1. Определение необходимости в проекте работ по верификации и степени их организационной независимости, анализ критичности проектных требований;
  2. Установка процесса верификации (при необходимости);
  3. Выбор соответствующей независимой квалифицированной организации (при необходимости);
  4. Определение верифицируемых работ и продуктов, выбор работ и задач верификации;
  5. Разработка плана верификации;
  6. Реализация плана проведения верификации, обеспечение доступности его результатов заказчику, при необходимости организация связи с процессом решения проблем (см. п. 2.3.8 пособия).
* ***Верификация***
  1. Верификация договора по критериям;
  2. Верификация процесса по критериям;
  3. Верификация требований по критериям;
  4. Верификация результатов проектирования по критериям;
  5. Верификация исходных текстов программных модулей по критериям;
  6. Верификация сборки по критериям;
  7. Верификация документации по критериям.

***Договор*** должен быть верифицирован по следующим *критериям*:

* Возможность поставщика удовлетворять установленным требованиям;
* Непротиворечивость требований и охват ими потребностей пользователя;
* Наличие соответствующих процедур для внесения изменений в установленные требования и для решения проблем;
* Наличие процедур по взаимодействию и кооперации между участниками договора;
* Наличие критериев и процедур, предусмотренных в соответствии с установленными требованиями.

***Процесс*** должен быть верифицирован по следующим *критериям*:

* Соответствие и своевременность установления требований к планированию проекта;
* Пригодность, реализуемость, выполнимость в соответствии с планом и условиями договора выбранных для проекта процессов;
* Применимость стандартов, процедур и условий к процессам проекта;
* Укомплектованность и обученность персонала в соответствии с условиями договора.

***Требования*** должны быть верифицированы по следующим *критериям*:

* Непротиворечивость, выполнимость и тестируемость требований к системе;
* Распределение требований к системе между объектами технических и программных средств и ручных операций в соответствии с критериями проектирования;
* Непротиворечивость, выполнимость, тестируемость и точность отражения требований к системе в требованиях к программным средствам;
* Правильность (подтвержденная соответствующими методами) критических требований к программным средствам, в том числе по безопасности и защите.

***Результаты проектирования*** должны быть верифицированы по следующим *критериям*:

* Правильность, соответствие установленным требованиям и учет этих требований;
* Реализация соответствующей последовательности событий, исходных данных, выходных результатов, интерфейсов, логики; соответствие временным ограничениям и ограничениям размера; обнаружение, локализация и устранение ошибок;
* Возможность дальнейшего использования с учетом требований;
* Правильность, подтвержденная соответствующими методами, реализации требований безопасности, защиты и других критических требований.

***Исходные тексты программных модулей*** должны быть верифицированы по следующим *критериям*:

* Трассируемость с результатами проектирования и требованиями; тестируемость, правильность и соответствие требованиям и стандартам программирования;
* Реализация соответствующей последовательности событий, соответствующих интерфейсов, правильных данных и логики управления; завершенность; соответствие временным ограничениям и ограничениям размера; обнаружение, локализация и устранение ошибок;
* Соответствие результатам проектирования и требованиям;
* Возможность дальнейшего использования с учетом результатов тестирования и требований;
* Правильность, подтвержденная соответствующими методами, реализации требований безопасности, защиты и других критических требований.

***Сборка*** должна быть верифицирована по следующим *критериям*:

* Полнота и правильность сборки программных компонентов и модулей каждого программного объекта в соответствующий программный объект;
* Полнота и правильность сборки технических и программных объектов и ручных операций в систему;
* Выполнение задач сборки в соответствии с планом сборки.

***Документация*** должна быть верифицирована по следующим *критериям*:

* Соответствие, полнота и непротиворечивость документации;
* Своевременность подготовки документации;
* Соблюдение установленных процедур управления конфигурацией документов.

### **N-версионное программирование.**

Очевидно, что использование технологий, минимизирующих ошибки в ПО, методов верификации, методов тестирования и отладки, позволяет существенно уменьшить количество ошибок ПО, но к сожалению все программные ошибки не устраняются. Оставшиеся скрытые ошибки ПО могут при определенном сочетании исходных данных привести к отказам в ходе эксплуатации.

Естественным является стремление повысить отказоустойчивость ПО, применяемого в критических областях путём использования различных форм **избыточности**. Введение форм избыточности является классическим приемом обеспечения, например, надежной работы аппаратуры. Специфика программной избыточности определяется разной природой неисправности. Если в аппаратуре сбой и отказы имеют очевидно случайный характер вследствие, например,:

* Локального перегрева;
* Окисления контактов;
* Пробой **p-n** перехода;
* Старение полупроводника;
* Электромагнитные помехи и т. д.,

то наработка на отказ для аппаратуры представляет собой случайную величину (СВ), имеющую некоторый закон. В ПО ошибки определяются в первую очередь его сложностью и вносятся как будто неслучайно, практически во всех работах жизненного цикла ПО. Но проявляются ошибки в процессе эксплуатации в случайный момент времени при случайном сочетании исходных данных, и таким образом, можно считать, что наработка на отказ так же представляет собой СВ. Но природа ошибки в ПО совершенно другая.

Под ***N–версионным программированием*** понимается независимое создание ***N ≥ 2*** функционально–эквивалентных программ на основе общего исходного описания.

Можно сформулировать следующие ***требования к программной избыточности*** (т.е. к ***N***–версионномупрограммированию):

1. Для выполнения одной и той же задачи рекомендуется разработать ***N*** независимых программ. При этом независимое создание программ подразумевает отсутствие контактов в процессе работы между ***N*** программистами или группами программистов. При этом насколько возможно, желательно использование различных алгоритмов, языков программирования и средств разработки.
2. В спецификации ПО необходимо так же предусмотреть дополнительно:

а) разработку алгоритма сравнения по совпадению при ***N = 2*** и мажоритарные при ***N > 2***;

б) точки контроля;

в) для оценки правильности результатов в алгоритме сравнения устанавливается допустимый диапазон расхождения числовых результатов.

1. ***N*** независимо разработанных программ должны работать параллельно на ***N*** независимых компьютерах. Затем результаты вычисления на каждом компьютере сравниваются между собой, и в случае расхождения наиболее вероятный результат находится голосованием (по большинству) для ***N > 2***. При этом на аппаратном уровне:

а) компьютеры должны иметь эффективные средства взаимодействия с целью быстрого сравнения результатов;

б) архитектура сист. должна быть такова, чтобы отказ одного из компьютеров не мог вызвать отказа всей сист. в целом.

Данная схема сравнения результатов вычислений соответствует *мажоритарному резервированию* или *резервированию по принципу голосования*.

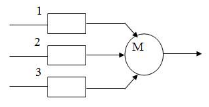


Рис. 9.5. Схема голосования “два из трех”

На данном рисунке: 1, 2, 3 – идентичные сист. обработки информации (компьютер); М – мажоритарный орган.

Чем больше параллельно работающих элементов, тем выше уровень надежности и отказоустойчивости. Чем больше число элементов, результаты которых должны совпадать, тем выше достоверность. Для схемы 2 из 3 вместо одного элемента включается три идентичных элемента, выходы которых подаются на мажоритарный орган, который осуществляет операцию выбора по большинству. Условием безотказной работы данной сист. является безотказная работа любых двух элементов из трех и мажоритарного органа. Вероятность безотказной работы данной сист. равна:



при условии равнонадежности элементов 1, 2, 3. Здесь: ***Р*** – вероятность безотказной работы каждого из элементов 1, 2, 3; ***q*** – вероятность отказа каждого из элементов 1, 2, 3; ***РМ*** – вероятность безотказной работы мажоритарного органа.

Такая схема не позволяет достичь вероятности безотказной работы сист. выше, чем вероятность мажоритарного органа **PМ**, но вероятность **PМ** в силу простоты мажоритарного органа намного выше, чем вероятность безотказной работы элементов 1, 2, 3. В общем сам мажоритарный орган можно зарезервировать. Например, идентичные программы в центральном вычислительном комплексе космического челнока Space Shuttle выполняются на трёх и более процессорных блоках с последующим мажоритарным выборам.

Очевидно, что N-версионное программирование позволяет бороться с ошибками как на программном, так и на аппаратном уровне. Объективный недостаток – стоимость разработки увеличивается в **N** раз.

### **Отказоустойчивость компьютерных систем. Граф процесса восстановления типичной отказоустойчивой сист.**

**Отказоустойчивость** – св-во архитектуры компьютерных систем, позволяющее пользователю или программе продолжить работу и тогда, когда в аппаратных или программных средствах возникли отказы.

**По способу реализации отказоустойчивость** подразделяется на:

1. Активную;
2. Пассивную.

**Активная отказоустойчивость** базируется на отдельно выделенных процессах обнаружения отказа, локализации отказа и реконфигурации сист.

Отказы обнаруживаются при помощи средств контроля, локализуются при помощи средств диагностирования и устраняются автоматически реконфигурацией сист., которая заключается в перестройке структуры компьютерных систем таким образом, чтобы её отказавшие части были устранены от участия в работе.

**Пассивная отказоустойчивость** заключается в свойстве сист. не потерять свои функциональные свойства в случае отказа отдельных элементов сист. Иногда говорят, что отказ маскируется системой.

Пример пассивной отказоустойчивости систем - сист. с мажоритарным органом. Пассивная отказоустойчивость связана с увеличением аппаратуры в несколько раз.

Пассивная отказоустойчивость применяется в случае особенно ответственных компьютерных систем, в случаях, когда не допустимы даже кратковременные перерывы в работе компьютерных систем, а также для обеспечения отказоустойчивости важнейших подсистем компьютерных систем.

Применение активной отказоустойчивости характеризуется более экономным расходом аппаратных средств, чем применение пассивной отказоустойчивости. Но оно связано с восстановлением работы сист. после отказа, а также возможными потерями некоторой части данных.

Активная отказоустойчивость реализована только в многопроцессорных системах.

В то же время применение пассивной отказоустойчивости гарантирует практически безостановочную работу компьютерных систем и сохранение всей информации.

Введение отказоустойчивости является одним из методов повышения надежности компьютерных систем. Вопрос о построении и применении отказоустойчивых систем возникает тогда, когда другие пути повышения надежности не могут обеспечить требуемого уровня надежности или тогда, когда они оказываются экономически не оправдываемыми.

Отказ компьютерных систем может быть вызван различными причинами. Кроме отказов встречаются сбои, вызванные случайными помехами и разрушающие обрабатываемую информацию в течении одного или нескольких тактов работы. Отказы и сбои вызывают ошибки в работе элементов аппаратуры. Кроме ошибок аппаратуры встречаются ошибки оператора (входные данные) и ошибки программ, которые также могут исказить обрабатываемую информацию. Множество выше перечисленных ошибок может привести к невыполнению компьютерными системами своих функций. Отказом компьютерных систем также может быть понижение производительности сист. ниже допустимого уровня, вследствие отказов определенных подсистем.

В общем отказоустойчивость сист. обеспечивается введением избыточности, т.е. созданием **определенных резервов или запасов в системе**.

В компьютерных системах может быть использована **избыточность**:

1. Временная;
2. Алгоритмическая;
3. Структурная.

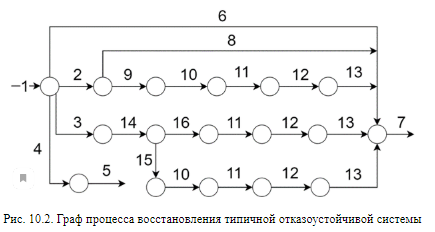
**Временная избыточность** - наличие дополнительного времени для решения задачи, с тем, чтобы в случае возникновения сбоя или других ошибок можно было исправлять их путем повторения вычислений. Она использует некоторую часть производительности компьютера для контроля исполнения ПО и восстановления вычислительного процесса. Поэтому при проектировании должен быть предусмотрен запас производительности.

**Алгоритмическая избыточность** - применение таких алгоритмов, которые обеспечивают удовлетворительные результаты в случае наличия или возникновения ошибок в процессе вычислений.

Например, свойствами избыточных алгоритмов обладают итерационные алгоритмы, обеспечивающие сходимость при случайных отклонениях промежуточных результатов. При возникновении ошибки вычислительный процесс занимает больше времени.

Алгоритмическая избыточность и временная избыточность увеличивают устойчивость сист. к сбоям.

**Структурная избыточность** – наличие дополнительных подсистем в структуре компьютерных систем, предназначенных для автоматической замены отказавших элементов, узлов, устройств и подсистем.



На рис. 10.2 ошибка 1 может обнаруживаться либо аппаратными 2 средствами контроля, либо программными 3 средствами контроля, либо не обнаруживаются средствами контроля 4 и в последнем случае результатом является отказ сист. 5. В зависимости от степени применения пассивной отказоустойчивости в компьютерных системах ошибка может быть замаскирована 6. В последнем случае вычислительный процесс продолжается без задержки 7.

При обнаружении ошибки аппаратными средствами 2 в большинстве систем проводится повторение выполняемой операции заданное число раз. Если повторение было успешным, т.е. имел место сбой последствия которого при повторении операции,, исчезли, вычислительный процесс продолжается 8. Для повторения операции необходимо чтобы аппаратные средства сохранили операнды до окончания контроля над выполненной операцией.

Если повторение операции было безуспешным 9, то это говорит об устойчивой ошибке в аппаратуре и поэтому проводится автоматическая реконфигурация 10.

Реконфигурация может заключаться либо в замене отказавшей подсистемы за счет резервов, либо в ее простом отключении. В последнем случае имеет место постепенная деградация сист. После реконфигурации производится восстановление информации 12. Для этого по ходу вычислительного процесса предусматривают контрольные точки 11, в которых состояние сист, и вычислительного процесса подвергаются контролю. В случае положительного результата контроля состояние данной программы и данного процессора (промежуточные результаты, содержание регистров и др.) записываются либо в оперативную память другого процессора, либо в общей оперативной памяти, либо на внешнем носителе.

В ходе восстановления информации содержание этих дублирующих записей переписывается в тот процессор, который после реконфигурации берет на себя функции отказавшего. Затем, начиная с контрольной точки вычислительный процесс возобновляется 13.

Аналогичные процедуры проводятся в случае, когда ошибка обнаружена программными средствами 3. Но при этом повторение операции не имеет смысла, так как программные средства обнаруживают отказ с запозданием, за которое были выполнены уже другие операции и поэтому первоначальные операнды вряд ли сохранятся.

Следовательно, после обнаружения ошибки программными средствами могут быть задействованы тесты 14. Если тесты подтверждают наличие устойчивого отказа 15, то следует реконфигурация 10, возврат к контрольной точке 11, восстановление данных 12 и повторение вычислений 13.

Если устойчивого отказа нет 16, то повторяются перечисленные операции без реконфигурации.

Восстановление может оказаться безуспешным также в случае наличия ошибки в программах, разрушения информации в контрольных точках, исчерпания резервов или снижения производительности сист. из-за отказов ниже допустимого уровня.

Описанный выше процесс может варьироваться в конкретных системах, особенно, что касается способов обнаружения отказов.

### **Оптимальное распределение ресурсов в отказоустойчивых компьютерных системах.**

В отказоустойчивых системах существует ряд параметров, от которых зависит отказоустойчивость сист.: X = (x1, …, xn). Этими параметрами могут быть конкретное количество резервных элементов сист., количество версий ПО, временные периоды контроля и восстановления и т. д.

Надежность КС зависит от некоторых параметров монотонно, например, увеличение количества резервов ведет к росту надежности.

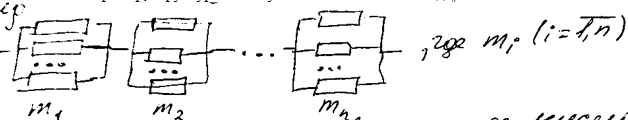
**Возможны 2 постановки задачи:**

1. Нахождение экстремума показателей надежности ПС, произведенное при некоторых ограничениях: min(max) П(x), где x G
2. Но если целью оптимизации является обеспечение не максимально, а заданной надежности сист., то решается обратная задача: min(max) Ф(x), где П(х) Н

**Методы оптимизации:**

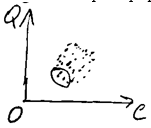
1. ***Метод перебора:***

Численные методы определения оптимального резерва позволяют найти сколь угодно точное решение и применимы для очень сложных моделей надежности. Простейшим численным методом оптимизации является метод перебора, когда сравниваются между собой все возможные варианты структуры КС, и выбирают тот вариант, который лучше всего отвечает установленным требованиям. Например, структура отказоустойчивой КС имеет вид:



Представляет собой максимально возможное число параллельных подсистем i-го типа, тогда число конкурирующих вариантов определяется следующим образом: **N =**

Из анализа данной формулы видно, что **число конкурирующих вариантов** возможных решений может быть **очень большим**, и поэтому данный метод может применяться только для **очень простых моделей надежности**. Поэтому для сокращения числа вариантов моделей перебора вводится понятие **доминирующей последовательности**. Рассмотрим график вариантов технических решений:



Из анализа 1-ой и 2-ой постановки задачи видно, что наибольший интерес представляют те решения, которые обладают **минимальной стоимостью и максимальной надежностью**, т. е. область **снизу слева**. Подмножество вариантов этих решений перспективно с точки зрения поиска оптимального варианта решения. **Последовательность таких решений называется доминирующей**. Обычно мощность доминирующей последовательности намного меньше мощности всех вариантов, и поэтому не составляет труда выбрать оптимальное решение из вариантов, входящих в доминирующую последовательность.

Фактически **метод полного перебора** **позволяет найти глобальный экстремум**, но применим **для простых систем**.

1. **Градиентный метод поиска локального экстремума**:

Является **простым и достаточно эффективным** методом. В общем случае решение задач градиентным методом заключается в том, что **отыскивается значение** **экстремума целевой функции путем последовательных шагов из начальной точки по направлению градиента**, т. е. максимального увеличения или уменьшения целевой функции. Целевая функция позволяет сравнивать альтернативные решения.

Сначала рассматривается исходная нерезервированная система:

На 1-ом шаге **отыскивается элемент** сист., добавление к которому дает **наибольшее отношение прироста показателя надежности к приросту затрат** (стоимости).

На 2-ом шаге отыскивается **следующий элемент** сист., который даст максимальное отношение прироста надежности к приросту затрат.

В общем градиентный метод позволяет определить **только часть элементов доминирующей последовательности**.

**Итерации** повторяются до тех пор, пока **надежность не достигнет заданной**. Однако результат данного метода **не является самым оптимальным**.

Данный подход может быть применим при оптимизации конфигурации компьютерных сетей. Самые приемлемые решения достигаются наращиванием сети т. о., чтобы прирост надежности, отнесенный к единице расходов был максимальным.