**Основные вопросы для подготовки к экзамену**

**по дисциплине «Надежность программного обеспечения»**

**Надежность аппаратного обеспечения компьютерных систем**

* **Основные понятия и определения в области надежности аппаратного обеспечения компьютерных систем**

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

Под **объектом** понимается либо компьютерная система (например, компьютерная сеть), либо ее подсистема (например, компьютер, винчестер, принтер).

***Свойства систем:***

**Безотказность -** свойство объекта непрерывно сохранять работоспособное состояние в течение некоторого времени или наработки.

**Ремонтопригодность -** свойство объекта, заключающееся в приспособленности его к поддержанию и восстановлению работоспособного состояния путем технического обслуживания и ремонта.

**Долговечность -** свойство объекта сохранять работоспособное состояние до наступления предельного состояния, при установленной системе путем технического обслуживания и ремонта.

**Сохраняемость -** свойство объекта сохранять в заданных пределах значения параметров, характеризующих способность объекта выполнять требуемые функции в течении и после хранения (и транспортирования).

**Наработка -** продолжительность или объем работы объекта. Выражается чаще всего через время, может быть в другой форме: количество циклов работы, количество решенных задач и т.д.

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

**Наработка до отказа -** наработка объекта от начала эксплуатации до момента возникновения отказа.

**Сбой -** кратковременное нарушение правильной работы объекта, после которого работоспособность объекта само восстанавливается или восстанавливается оператором без проведения ремонта.

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

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

**Достоверность информации выдаваемой объектом -** свойство объекта производить безошибочно преобразование, хранение и передачу информации. Показателем достоверности обычно является вероятность искажения либо вероятность потери информации в одном знаке.

**Показатели надежности невосстанавливаемых объектов**

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

Иногда P(t) называют **функцией надежности**.

**Вероятность отказа объекта Q(t)** - вероятность того, что объект откажет хотя бы один раз в течение заданной наработки, будучи работоспособным в начальный момент времени. Вероятность отказа в течение заданной наработки (0,t).

**Плотность распределения наработки до отказа f(t)** - есть производная от функции ненадежности

f(t) = dQ(t) / dt = - dP(t) / dt

Значение функции f(t) равно числу отказавших объектов, отнесенное к общему числу объектов N0, поставленных на испытания за период Δt. Фактически f(t) представляет собой безусловную плотность вероятность возникновения отказа.

Любая функция f(x) является функцией плотности, если обладает следующими свойствами

1)

2)

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

**Средняя наработка до отказа** определяется как математическое ожидание времени до отказа (фактически это время до первого отказа).

Интегрируя по частям выражение, получим

Т.е. средняя наработка численно равна площади под кривой P(t)

**Показатели надежности восстанавливаемых объектов**

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

**Стационарный коэффициент готовности** определяется как

Другой способ определения стационарного коэффициента готовности

Где Σ tp - суммарное время работы объекта

Σ tв - суммарное время восстановления объекта

**Способы расчета эффективности компьютерных систем**

**Эффективностью системы** называется свойство системы выдавать некоторый полезный результат (эффект) при ее использовании в соответствии с назначением.

При определении **показателей эффективности** систем рассматривают результат применения системы, который почти всегда имеет элементы случайности. Поэтому в качестве показателей эффективности часто применяют **характеристики случайных событий, случайных величин или случайных процессов.**

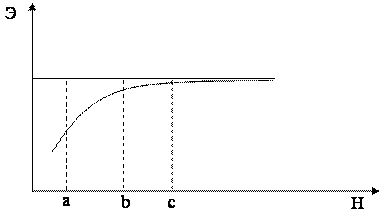
Если задачей применения системы является **осуществление некоторого события**, то за показатель эффективности принимают **вероятность выполнения** этого события.

Если результат применения системы можно рассматривать как **случайную величину**, то за показатель эффективности системы принимают **параметры закона распределения** этой случайной величины (математическое ожидание, дисперсия и т. д.).

Большинство сложных систем являются восстанавливаемыми, и в процессе эксплуатации существует возможность восстанавливать отказавшие элементы без прекращения функционирования всей системы (например, глобальная сеть Интернет, локальная сеть и т. п.). При отказах отдельных элементов сложной системы может уменьшаться эффективность функционирования всей системы.

**Эффективность системы зависит от множества факторов**, в том числе от

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



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

**Снижение эффективности из-за недостаточной надежности системы** характеризуют следующие выражения:

∆Э = Эи – Эр

∆Э / Эи

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,

**Оценка надежности ИС методом минимальных путей и сечений**

Применяется в случаях, когда структура и способы организации ИС предполагают резервирование, но его нельзя представить по схеме последовательного и параллельного включения элементов или подсистем. Метод минимальных путей и сечений относится к приближенным методам и позволяет определить граничные оценки надежности сверху и снизу.

**Путь** в сложной структуре - это последовательность элементов, обеспечивающих функционирование (работоспособность) системы.

**Сечение** - совокупность элементов, отказы которых приводят к отказу системы.

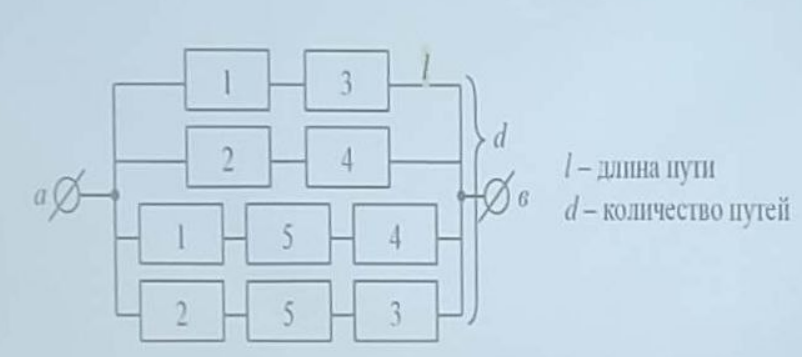
**Вероятность безотказной работы последовательно включенных** параллельных цепей **дает верхнюю оценку** для ВБР системы данной структуры.

**Вероятность безотказной работы параллельно включенных последовательных цепей** из элементов путей **дает нижнюю оценку** для ВБР системы данной структуры. Фактическое значение показателя надежности находится между верхней и нижней границами.



Набор элементов образует минимальный путь, если исключение любого элемента из набора приводит к отказу пути. В пределах одного пути элементы находятся в основном соединении, а сами пути включаются [...].

Набор минимальных путей для мостиковой схемы образован элементами: 1, 3; 2, 4; 1, 5, 4; 2, 5, 3.



Для всех элементов схемы известны ВБР P1, P2, P3, P4, P5 и соответствующие им вероятности отказа типа “обрыв” Q1÷Q5. Необходимо определить вероятность наличия цепи между точками a и b.

Поскольку один и тот же элемент включается в два параллельных пути, то в результате расчета получается оценка безотказности сверху:

Pв = 1 - Q13\*Q24\*Q154\*Q253 = 1 - (1 - P1P3)(1 - P2P4)(1 - P1P5P4)(1 - P2P5P3)

При определении минимальных сечений осуществляется подбор минимального числа элементов, перевод которых из работоспособного состояния в неработоспособное вызывает отказ системы. Возвращение любого из элементов в работоспособное состояние восстанавливает работоспособное состояние системы. Поскольку отказ каждого из сечений вызывает отказ системы, то сечения соединяются последовательно.



Pн = P12 \* P34 \* P154 \* P253 = (1 - q1q2)(1-q3q4)(1-q1q5q4)(1-q2q5q3)

В пределах каждого сечения элементы соединяются параллельно, так как для работы системы достаточно наличия работоспособного состояния любого из элементов сечения. Так как один и тот же элемент включается в два сечения, то полученная оценка является оценкой снизу.

Вероятность безотказной работы системы Pс оценивается по двойному неравенству:

Pн < Pс < Pв

**Метод позволяет представить систему с произвольной структурой в виде параллельных и последовательных цепей.**

**Логико-вероятностные методы анализа надежности ИС**

Сущность логико-вероятностных методов заключается в использовании функций алгебры логики для аналитической записи условий работоспособности системы. От функций алгебры логики следует переход к вероятностным функциям, объективно отражающим безотказность системы. Следовательно, с помощью логико-вероятностного метода можно описать ИС для расчета надежности с помощью **аппарата математической логики** с последующим использованием **теории вероятностей** для определения показателей надежности.

Предполагается, что система может находиться только в двух состояниях:

* полной работоспособности (y = 1)
* полного отказа (y = 0)

Действие системы детерминировано зависит от действия ее элементов, т. е. **y** является функцией x1, x2, …, xi, …, xn. Элементы также могут находиться только в двух несовместных состояниях:

* полной работоспособности (xi = 1)
* полного отказа (xi = 0)

Функцию алгебры логики, связывающую состояние элементов с состоянием системы **y** (x1, x2, …, xn) называют **функцией работоспособности системы F(y).**

Для оценки работоспособных состояний системы используют два понятия:

1. **Кратчайший путь успешного функционирования**, который представляет собой такую конъюнкцию ее элементов, ни одну из компонент которой нельзя изъять, не нарушив функционирования системы.

, где **i** - принадлежит множеству номеров, соответствующих данному ***l***-му пути.

**Кратчайший путь успешного функционирования** системы описывает одно из ее возможных работоспособных состояний, которое определяется минимальным набором работоспособных элементов, абсолютно необходимых для выполнения заданных для системы функции.

1. Минимальное сечение отказов системы, представляющее собой такую конъюнкцию, из отрицаний ее элементов, ни одну из компонент которой нельзя изъять, не нарушив условия неработоспособности системы.

где означает множество номеров, соответствующих данному сечению.

**Минимальное сечение отказов** системы описывает один из возможных способов нарушения работоспособности системы с помощью минимального набора отказавших элементов.

Каждая избыточная система имеет конечное число кратчайших путей () и минимальных сечений ( ).

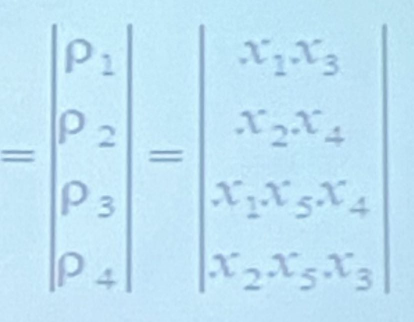
Используя эти понятия можно записать условия работоспособности системы:

1. в виде дизъюнкции всех имеющихся кратчайших путей успешного функционирования:
2. в виде конъюнкции отрицаний всех минимальных сечений отказов системы:

Таким образом, условия работоспособности реальной системы можно представить в виде условий работоспособности некоторой эквивалентной (всм надежности) систем, структура которых представляет:

* параллельное соединение кратчайших путей успешного функционирования,
* соединение отрицаний минимальных сечений.

Для мостиковой структуры ИС функция работоспособности системы **с помощью кратчайшего пути успешного функционирования** запишется следующим образом:



Функцию работоспособности этой же системы с использованием понятия **минимального сечения отказов системы** можно записать в следующем виде:

При небольшом числе элементов (не более 20) может быть использован табличный метод расчета надежности, который основан на использовании теоремы сложения вероятностей совместимых событий.

Вероятность безотказной работы системы можно вычислить по формуле (через вероятностную формулу)

Логико-вероятностные методы (разрезания, табличный, ортогонализации) широко применяются в *диагностических процедурах* при построении деревьев отказов и определении базисных (исходных) событий, вызывающих отказ системы.

Для определения показателей надежности компьютерной системы со сложной структурой резервирования может быть использован **метод статического моделирования.**

Идея метода заключается в генерировании логических переменных с заданной вероятностью , которые подставляются в логическую структурную функцию моделируемой системы в произвольной форме и затем вычисляется результат

Совокупность независимых случайных событий, образующих полную группу, характеризуется вероятностями появления каждой из событий , причем

Для моделирования этой совокупности случайных событий используется генератор случайных чисел, равномерно распределенных в интервале 0-1.

Значение выбирается в равным вероятности безотказной работы i-ой подсистемы. При этом процесс вычисления повторяется раз с новыми, независимыми случайными значениями аргументов (при этом подсчитывается кол-во единичных значений логический структурной функции).

Отношение является статической оценкой вероятности безотказной работы

где - кол-во времени безотказно работающих до момента времени объектов, при их исходном кол-ве.

**Метод дифференциальных уравнений расчета надежности восстанавливаемых систем**

Метод применяется для оценки надежности восстанавливаемых систем и основан на допущенном о показательных распределениях времени между отказами (наработки) и времени восстановления.

Параметр потока отказов и интенсивность восстановления , где - среднее время выполнения безотказной работы, - среднее время восстановления.

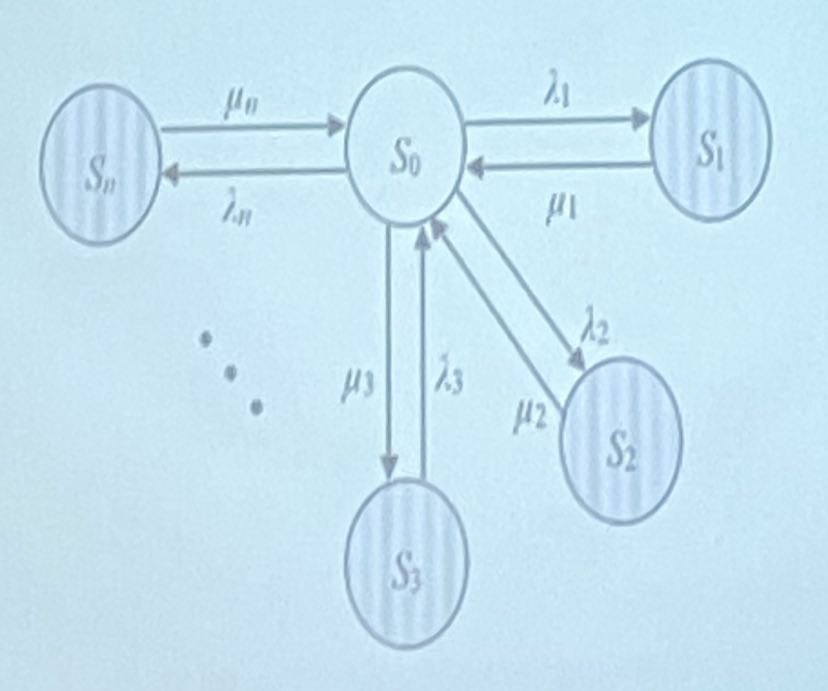
Для применения метода необходимо иметь математическую модель для множества возможных состояний системы , в которых она может находится при отказах в восстановлениях системы.

Время от времени система скачком переходит из одного состояния в другое под действием отказов и восстановлений ее отдельных элементов.

При анализе поведения системы во времени удобно пользоваться графом состояний.

Граф состояний - это направленный граф, где кружками или прямоугольниками изображают возможные состояния системы. Он содержит столько вершин, сколько различных состояний возможно у объекта или системы. Ребра графа отражают возможные переходы из некоторого состояния во все остальные с параметрами интенсивностей отказов и восстановлений (около стрелок показаны интенсивности переходов).

Граф состояний нерезервированной восстанавливаемой системы с - элементами имеет вид

  
  
Возможны следующие состояния

* - все элементы работоспособны;
* - первый элемент неработоспособен, остальные работоспособны;
* - второй элемент неработоспособен, остальные работоспособны;
* …
* - - ый элемент неработоспособен, остальные работоспособны;

Каждой комбинации отказовых и работоспособных состояний подсистем соответствует одно состояние системы. Число состояний системы , где - кол-во подсистем (элементов).

Связь между вероятностями нахождениями системы во всех его возможных состояниях выражается системой дифференциальных уравнений Колмогорова (уравнений первого порядка).

Система уравнений Колмогорова включает столько уравнений, сколько вершин в графе состояний объекта.

Система дифференциальных уравнений дополняется нормировочным условием:

, где - вероятность нахождения системы в -ом состоянии; - число возможных состояний системы.

Решение системы уравнений при конкретных условиях дает значение искомых вероятностей .

Все множество возможных состояний системы разбивается на две части: подмножество состояний , в которых система работоспособна, и подмножество , в которых система неработоспособна.

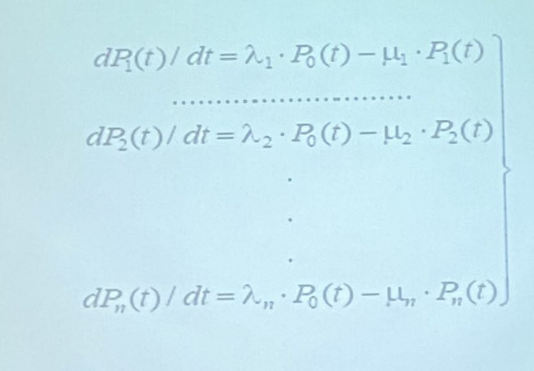
Функция готовности системы:

, где - вероятность нахождения системы системы в работоспособном состоянии. - число состояний в которых система работоспособна.

Для вычисления коэффициента готовности системы или коэффициента простоя (перерывы в работе системы допустимы), рассматривают установившийся режим эксплуатации при

При этом все производные

и система дифференциальных уравнений переходит в систему алгебраических уравнений, которая легко решается.



При установившимся режиме эксплуатации (при ):



Решив полученную систему алгебраических уравнений с учетом нормировочного условия, находим показатели надежности.

**Марковские модели для оценки надежности резервированных восстанавливаемых ИС**

Функционирование ИС и их составных частей можно представить как совокупность процессов перехода из одного состояния в другое под воздействием каких-либо причин.

С точки зрения надежности восстанавливаемых ИС их состояние в каждый момент времени характеризуется тем, какие из элементов работоспособны, а какие восстанавливаются.

Если каждому возможному множеству работоспособных (неработоспособных) элементов поставить в соответствие множество состояний объекта, то отказы и восстановления элементов будут отображаться переходом объекта из одного состояния в другое.

Если объект состоит из двух элементов, то он может находиться в одном из четырех состояний: .

* - оба элемента работоспособны
* - неработоспособен только первый элемент.
* - неработоспособен только второй элемент.
* - неработоспособны оба элемента.

Множество возможных состояний объекта: .

Полное множество состояний исследуемой системы может быть дискретным или непрерывным.

Для марковских моделей предлагается, что системы имеют дискретное пространство состояний.

Последовательность состояний такой системы и сам процесс переходов из одного состояния в другое называется **цепью.**

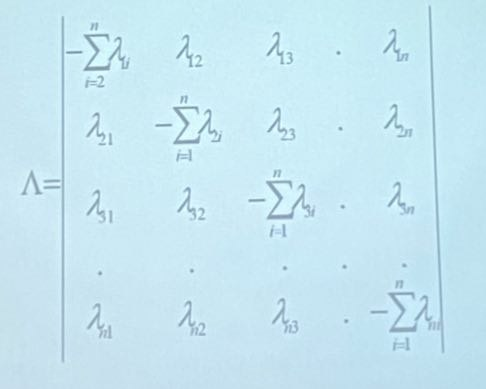
Вероятности переходов из одного состояния в другое обозначаются символами а процесс переходов называется **марковской цепью** или **цепью Маркова.**

Марковское свойство связано с отсутствием последствия. Это означает, что поведение системы в будущем зависит только от ее состояния в данный момент времени, и не зависит от того, каким образом она пришла в это состояние.

Марковские процессы позволяют описать последовательности отказов-восстановлений в системах при помощи графа состояний.

Для расчета надежности используется система дифференциальных уравнений, которая в матричной форме может быть записана в виде:

- матрица интенсивности переходов (матрица коэффициентов при вероятностях состояний):



где - интенсивности перехода системы из го состояния в ое;

вероятность того, что система находится в ом состоянии.

**Последовательность оценки надежности** резервированных восстанавливаемых систем с использованием метода марковских цепей следующая:

1. Анализируется состав системы и составляется структурная схема надежности. По схеме строится граф, в котором учитывается все возможные состояния.
2. Все вершины графа в результате анализа структурной схемы разделяются на два подмножества:
   1. Вершины, соответствующие **работоспособному** состоянию системы.
   2. Вершины, соответствующие **неработоспособному** состоянию системы.
3. С помощью графа состояний составляется система дифференциальных уравнений Колмогорова.
4. Выбираются начальные условия решения задачи.
5. Определяются вероятности нахождения системы в работоспособном состоянии в произвольный момент времени.
6. Определяется вероятность безотказной работы системы.

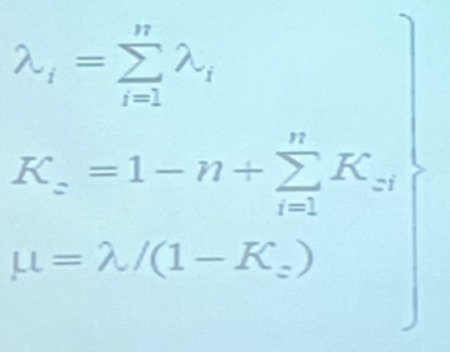
В случае необходимости определяются и другие показатели.

**Приближенный метод расчета надежности технических средств ИС**

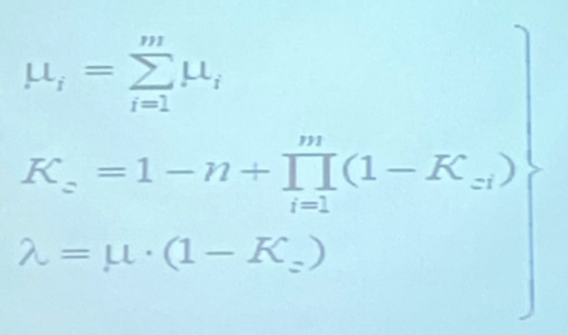
Метод основан на следующих **допущениях**:

* Время восстановления намного меньше времени безотказной работы.
* Интенсивности отказов и восстановлений системы - постоянные величины.
* Отказы и восстановления отдельных подсистем - независимые случайные события.

Для последовательного включения подсистем справедливы следующие приближенные зависимости:



Для параллельного включения подсистем справедливы следующие приближенные зависимости:



В этих формулах приняты следующие обозначения:

* - интенсивность отказов последовательной (параллельной) системы из подсистем.
* - коэффициент готовности последовательной (параллельной) подсистемы группы из .

Переменные с индексом обозначают соответствующие показатели отдельных подсистем.

Если в системе применяется скользящее резервирование, то для отделения коэффициента готовности справедливо:

где - минимально необходимое по требованиям производительности число работоспособных подсистем;

- коэффициент готовности подсистем (при скользящем резервировании все подсистемы однотипны).

Интенсивность восстановления в случае скользящего резервирования определяется:

где - интенсивность восстановления подсистем.

В случае указанных выше допущений интенсивность отказов численно равна параметру потока отказов .

**Надежность программного обеспечения компьютерных систем**

**Понятия свойств, функций и требований к программному обеспечению**

* Обычно пользователь описывает свойства программного обеспечения, которые ему необходимы. На этом уровне детализации требования к ПС обычно называются *пользовательскими требованиями* или функциями
* Функции представляют собой описание желаемого поведения разрабатываемого ПС и служат основой для формирования требований разработчика к ПС. В общем случае каждая пользовательская функция порождает одно или несколько требований разработчика
* Требования разработчика представляют собой детализированные описания пользовательских функций.
* Документом, в котором представлены требования к разрабатываемому программному средству, является спецификация требований к программному обеспечению (СТПО).
* В соответствии с ГОСТ 19781-90 спецификация представляет собой 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 является техническим документом, в котором описаны условия функционирования для отдельного программного продукта или набора программ, которые выполняют определенные задачи в определенной среде.
* В случае, когда спецификация требований к ПО является частью большой системы, у разработчика должна иметься спецификация требований к ПО, описывающая взаимодействие системы с частью программного обеспечения и содержания требования к характеристикам и функциональности данной части программного обеспечения.
* спецификация требований должна содержать:
  + требований к процессу разработки;
  + требования к проекту;

**Критерии качества требований** **к программному обеспечению**

Критерии качества требований к ПО декларируются стандартом, который выделяет следующие критерии для оценки качества спецификации требований к ПО:

Корректность(правильность) требований Требования правильны тогда и только тогда, когда все заявленные в них требования отражают реальные потребности заказчика и могут быть реализованы в будущем продукте.

Например, если А - потребности пользователя, В - сформулированные требования, то корректные требования находятся на пересечении А и В.

Недвусмысленность требований Требование является недвусмысленным, когда его могут однозначно интерпретировать разработчик и заказчик. Основными трудностями в формулировании требований часто является естественный язык, на котором они написаны. Так как естественный язык неоднозначен, это порождает множество проблем при формулировании однозначных требований. Для решения этой проблемы были придуманы специальные формализованные языки, называемые “Языками спецификации требований”, которые помогают разработчику описать требования однозначно.

Язык спецификаций - формальный язык, предназначенный для декларативного описания структуры, связей, свойств, данных и способов их преобразований (в отличие от активных языков) без явного упоминания порядка выполняемых действий и использования конкретных значений данных.

* В отличие от языков программирования, используемых при реализации компьютерных программ, языки спецификаций применяются для проведения системного анализа, анализа требований, разработке архитектуры создаваемых программных систем и формальной верификации верификации программного обеспечения.

Полнота набора требовании Спецификация к ПО является полной, когда она удовлетворяет следующим критериям:

* + - * Содержит все существенные требования, касающиеся функциональных возможностей, выполнения, ограничений проектирования, характеристик или внешних интерфейсов. В частности, любые внешние требования, накладываемые спецификацией системы, должны быть выявлены и рассмотрены.
      * 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.
      * Содержит все подписи и ссылки на все рисунки, таблицы и диаграммы и определение всех терминов и единиц измерения.

Непротиворечивость требований Спецификация требований к ПО не противоречива тогда и только тогда, когда каждое заявленное требование не противоречит ни одному другому требованию.

Упорядоченность по важности и/или стабильности. Для улучшения восприятия и последующей работы с требованиями их следует упорядочить по важности и/или стабильности. Под стабильностью понимают степень изменяемости требования во времени, а под важностью понимают степень и необходимости требования, например: обязательное требование, условное, необязательное.

Проверяемость требований Характеристика подразумевает возможность проверки требований. Например, сложно проверить требование “дружественный интерфейс”, так как каждый человек (поставщик, заказчик) за этим требованием видит свое. если в требовании существуют параметры, имеющие количественную оценку, то нужно указать либо конкретное значение, либо диапазон значений для того, чтобы можно было по результатам проверки выяснить, удовлетворяет ли продукт установленным для него требованиям.

Целостность Спецификация требований к ПО не должна противоречить документам более высокого уровня и обеспечить внутреннюю целостность, что подразумевает отсутствие описаний общих частей требований, находящихся в противоречии друг к другу.

Модифицируемость Спецификация требований к ПО изменяема тогда и только тогда, когда имеет такие структуру и стиль, при которых любые изменения требований могут быть сделаны легко, полностью, последовательно и при сохранении структуры.

Спецификация требований к ПО должна:

* **иметь последовательную и удобную в работе структуру** с оглавлением, индексом и явными взаимными ссылками;
* **не быть избыточной**; т.е. одно и то же требование не должно встречаться больше чем в одном месте в СТПО;
* **выражать каждое требование** предпочтительнее **отдельно**, чем смешанным с другими требованиями.

Трассируемость Трассируемость требований определяется возможностью отследить связь между требованием с другими артефактами проекта, например, должен быть обеспечен легкий доступ к каждому требованию из документации, создаваемой в процессе разработки продукта.

**Понятия ошибки и отказа программного обеспечения**

В соответствии со стандартами ISO/IEC 9126-1:2001

Ошибка определена как некорректное действие, процесс или определение данных в компьютерной программе.

Отказ - прекращение способности программного средства выполнять необходимую функцию или его неспособность функционировать в заданных спецификацией условиях.

Надежность - это способность ПС поддерживать заданный уровень функционирования при эксплуатации в заданных спецификацией условиях.

Характеристике надежность в соответствии со стандартом соответствуют следующие подхарактеристики:

* завершенность - способность ПС избежать отказа в результате ошибок ПО;
* устойчивость к ошибке - способность ПС поддерживать заданный уровень функционирования при наличии ошибок в ПО или при нарушении его заданного интерфейса;
* восстанавливаемость - способность ПС восстанавливать заданный уровень функционирования и восстанавливать данные, непосредственно поврежденные в случае отказа.

Особенности программного обеспечения по сравнению с аппаратным обеспечением КС

* Ошибки в программах проявляются при их выполнении на некоторых наборах исходных данных (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.

**Разработка программного обеспечения. Основные процессы жизненного цикла**

Жизненный цикл - это структура, состоящая из процессов, работ и задач, включающая в себя разработку, эксплуатацию и сопровождение.

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

В соответствии со стандартом, каждый процесс ЖЦ разделен на набор работ, а каждая работа - набор задач.

Стандарт описывает архитектуру процессов жизненного цикла ПС. При этом он не определяет детали реализации или выполнения работ и задач, входящих в данные процессы.

Все процессы ЖЦ делятся на следующие группы:

* + основные процессы ЖЦ;
    - процесс заказа
      * содержит работы и заказы, выполненные заказчиком. состоит из: определения потребностей заказчика к системе; выпуска заявки на подряд; Выбора поставщика
    - процесс поставки
      * Состоит из работ и задач, выполняемых поставщиком: решения о подготовке предложения в ответ на заявку на подряд, присланной заказчиком, подписания договора и вступления с заказчиком в договорные отношения по поставке системы.
    - процесс разработки
      * Содержит 13 работ и задач, выполняемых разработчиком и включает в себя работы:
        + по анализу требований
        + по проектированию
        + по программированию
        + по сборке
        + по тестированию
        + по вводу в действие
        + по приёмке программных средств

**Разработка программного обеспечения. Работы и задачи процесса разработки**

**1-я работа - подготовка процесса.**

Должны быть выбраны и адаптированы стандарты, инструментарии и языки программирования для выполнения работ в процессе разработке, также должны быть выбрана модель ЖЦ

**2-я работа - анализ требований к системе.**

* требования к системе должны охватывать:
* функции и возможности системы
* коммерческие и организационные требования
* требования пользователя
* требования безопасности и защиты
* эргономические требования
* требования к интерфейсам
* эксплуатационные требования
* требования к сопровождению

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

**3-я работа - проектирование системной архитектуры.**

Должны быть определены:

* общая архитектура всей системы,
* указаны объекты технических и программных средств и ручных операций.

Ручные операции представляют собой те операции, которые пользователь системы выполняет вручную, так как, в основном, все разрабатываемые системы являются автоматизированными, а не автоматическими.

**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 ПС:

* 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 прием 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 (Software Requirement 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.

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.**

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.

Каждый модуль и база данных должны быть протестированы 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 - сборка системы.**

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 системы.**

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.**

**Разработка программного обеспечения. Вспомогательные и организационные процессы жизненного цикла**

**Вспомогательные процессы ЖЦ**

* **Документирование -** процесс формализованного описания информации, созданной в процессе или работе ЖЦ ПС.
* **Управление конфигурацией** - процесс применения административных и технический процедур на всем протяжении ЖЦ ПС.
* **Обеспечение качества -** процесс обеспечения соответствующих гарантий того, что ПС 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** - определение полноты соответствия установленных требований созданной системы 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.
* **Процесс управления** состоит из работ и задач, которые могут быть использованы любой стороной, управляющей соответствующим процессом(планирование, выполнение, контроль, оценки).
* **Процесс создания 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.
* **Процесс усовершенствования** является процессом установления оценки измерения, контроля и улучшения любого процесса ЖЦ ПС.
* **Обучение** - процесс обеспечения первоначального и продолженного обучения персонала работам по заказу, поставке, разработке, эксплуатации или сопровождению ПС.

**Причины появления ошибок в ПО**

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.

Например, в 4-й работе **(анализ требований к программным средствам)** процесса разработки ЖЦ ПС дефект - отсутствие требований или их неверная трактовка.

В 5-й работе **(проектирование программной архитектуры)** процесса разработки ЖЦ ПС правильные требования могут быть преобразованы в неправильную программную архитектуру.

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

В ходе собственно тестирования, квалификационных испытаний ПС и квалификационного испытания системы в целом (в 7, 8, 9, 11-й работах процесса разработки ЖЦ ПС) могут быть внесены новые ошибки.

По статистике исправление каждой 3-й ошибки вносит новую.

Ошибки вносятся на разных стадиях и присутствуют в соответствующих представлениях программы (артефактах процесса проектирования).

Причина появления ошибок - **высокая** **сложность** проектирования ПО.

**Основными причинами,** вызывающими нарушения нормального функционирования программы, **являются:**

* ошибки, скрытые в самой программе;
* искажения входной информации, подлежащей обработке;
* неверные действия пользователя;
* неисправности аппаратуры, на которой реализуется вычислительный процесс.

**Скрытые ошибки программы**

Специфика создания сложных программных средств состоит в том, что в процессе их отладки практически невозможно обнаружить и ликвидировать все ошибки.

В результате в программах остается некоторое количество скрытых ошибок. Они могут вызвать неверное функционирование программ при определенных сочетаниях входных данных.

Наличие скрытых ошибок программного обеспечения является главным фактором нарушения нормальных условий его функционирования.

**Основные классы ошибок в программах:**

***Ошибка вычислений***

Ошибки данного класса содержатся в закодированных математических выражениях или в получаемых с их помощью результатах. Примерами ошибок, относящихся к данному классу, являются:

* неверное преобразование типов переменных,
* неверный знак операции, ошибка в выражении индекса,
* переполнение или потеря значимости при вычислениях.

***Логические ошибки***

Являются причиной искажения алгоритма решения задачи. Такого рода ошибки возникают в связи с:

* неверной передачей управления,
* неверным заданием диапазона изменения параметра цикла,
* неверным условием и т.д.

***Ошибки ввода-вывода:***

Связаны с такими действиями, как управление вводом-выводом, формирование выходных записей, определение размеров записей и т.д. Примерами ошибок ввода-вывода являются:

* неправильная форма ввода (вывода),
* ошибка в задании числа формируемых строк (страниц) при печати;
* отсутствие признака конца файла и т.д.

***Ошибки манипулирования данными***

Примерами таких ошибок являются:

* неверно определенное число элементов данных;
* неверные начальные значения, присвоенные данным;
* неверно указанные длина операнда, имя переменной и т.д.

***Ошибки совместимости***

Связаны с отсутствием совместимости с операционной системой или другими прикладными программами, используемыми в данной программе.

***Ошибки сопряжений***

Вызывают неверное взаимодействие программы с другими программами (подпрограммами), с системными программами, устройствами ЭВМ, входными данными и т.д.

В качестве примеров ошибок сопряжения могут быть названы:

* несовместимость аргументов и параметров подпрограммы,
* отсутствие в системе необходимой подпрограммы,
* нарушения синхронизации при асинхронном выполнении программ и т.д.

**Искажение информации, подлежащей обработке**

Вызывает нарушение функционирования программного обеспечения, когда входные данные не попадают в область допустимых значений переменных программы. В этом случае между исходной информацией и характеристиками программы возникает несоответствие.

Причинами искажения вводимой информации могут быть:

* искажение данных на первичных носителях информации;
* сбои и отказы в аппаратуре ввода данных с первичных носителей информации;
* шумы и сбои в каналах связи при передаче сообщений по линиям связи;
* сбои и отказы в аппаратуре передачи или приема информации;
* потери или искажения сообщений в буферных накопителях вычислительной системы;
* ошибки в документации, используемой для подготовки вводимых данных;
* ошибки пользователей при подготовке исходной информации.

**Неверные действия пользователя**

Приводят к отказу в процессе функционирования ПО.

Связаны с неправильной интерпретацией сообщений, с неправильными действиями пользователя в процессе диалога с вычислительным средством и т.д.

Ошибки являются следствием некачественной программной документации, т.е. неверное описание:

* возможностей программы,
* режимов работы,
* форматов входной и выходной информации,
* диагностических сообщений и т.д.

**Неисправность аппаратуры**

Неисправности, возникающие при работе аппаратуры, используемой для реализации вычислительного процесса, оказывают определенное влияние на характеристику надежности ПО. Появление отказа или сбоя в работе аппаратуры приводит к нарушению нормального хода вычислительного процесса и во многих случаях - к искажению данных и текстов программ в основной и внешней памяти.

Симптомы появления ошибок в программе

* Преждевременное (аварийное) окончание выполнения программы
* Недопустимое увеличение времени выполнения программы
* Зацикливание вычислительного средства на выполнении некоторой последовательности команд одной из программ
* Полная потеря или значительное искажение накопленных данных, необходимых для успешного выполнения решаемых задач
* Нарушение последовательности вызова отдельных программ, в результате чего происходит пропуск необходимых программ либо непредусмотренное обращение к программам
* Искажение отдельных элементов данных (входных, выходных, промежуточных) в результате обработки искаженной исходной информации

**Обеспечение надежности ПО. Предупреждение, обнаружение и исправление ошибок**

Понимание того, что именно ошибки перевода являются основной причиной ошибок в ПО, является ключом к пониманию проблемы надежности ПО. Такое понимание явилось основой к разработке подходов к обеспечению надежности ПО. Данные подходы направлены на:

* Предотвращение появления ошибок в процессе разработки ПО
* Обнаружение ошибок перевода

Методы обеспечения надежности можно разбить на три группы:

* Предупреждение ошибок
* Обнаружение и исправление ошибок
* Обеспечение устойчивости к ошибкам

Методы предупреждения ошибок можно разбить на:

1. Методы, позволяющие справиться со **сложностью**, свести её к минимуму, т. к. это **главная причина ошибок перевода**
2. Методы достижения **большей точности** перевода
3. Методы **улучшения обмена** информацией

Для снижения структурной сложности ПС широко используются два принципа:

* Независимость компонентов (модулей) ПС
* Иерархическая структура (иерархия) ПС

Чтобы уменьшить сложность ПО, нужно разбить его (декомпозировать) на множество небольших достаточно простых для понимания независимых модулей. Иерархия позволяет декомпозировать ПО на компоненты с учетом ограничений на их сложность. Каждый вышестоящий уровень представляет собой совокупность структурных отношений между элементами нижестоящих уровней. Концепция уровня позволяет понять ПО, скрывая несущественные уровни детализации.

Например, иерархия в IDEF0 (методология функционального моделирования). Функциональная модель имеет иерархическую структуру, состоящую из множества IDEF0-диаграмм. Каждая диаграмма любого уровня может быть декомпозирована на несколько диаграмм нижележащего уровня. Каждая из диаграмм должна содержать не более шести блоков из-за ограничений по сложности. Таким образом сложность диаграмм должна находиться 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.

Обнаружение и исправление ошибок

Если предполагать, что в программном обеспечении какие-то **ошибки всё же будут**, то **лучшая** (после предупреждения ошибок) **стратегия – включить средства обнаружения ошибок** в самом программном обеспечении.

**Метод исправления ошибок** – после того, как ошибка обнаружена, либо она сама, либо ее последствия должны быть исправлены программным обеспечением.

**Исправление ошибок самой системой** – плодотворный метод проектирования надежных систем аппаратного обеспечения.

Аналогичные методы **неприменимы** к программному обеспечению вследствие глубоких внутренних различий между сбоями аппаратуры и ошибками в программах. Если некоторый программный модуль содержит ошибку, идентичные «запасные» модули также будут содержать ту же ошибку.

**Обеспечение надежности ПО. Устойчивость к ошибкам в ПО**

Методы этой группы ставят своей целью обеспечение функционирования программной системы при наличии в ней ошибок. Они разбиваются на три подгруппы:

* Динамическая избыточность
* Методы отступления
* Методы изоляции ошибок

#### Динамическая избыточность

Истоки концепции динамической избыточности лежат в проектировании аппаратного обеспечения. Один из подходов к динамической избыточности – метод голосования. Данные обрабатываются независимо, несколькими идентичными устройствами и результаты сравниваются. Если большинство устройств выработало одинаковый результат, этот результат и считается правильным.

Вследствие особой природы ошибок в программном обеспечении, ошибка, имеющаяся в копии программного модуля, будет также присутствовать во всех других его копиях, поэтому идея голосования здесь неприемлема.

Второй подход к динамической избыточности – выполнять эти запасные копии только тогда, когда результаты, полученные с помощью основной копии, признаны неправильными. Если это происходит, система автоматически вызывает запасную копию. Если ее результаты неправильны, вызывается другая запасная копия и т. д.

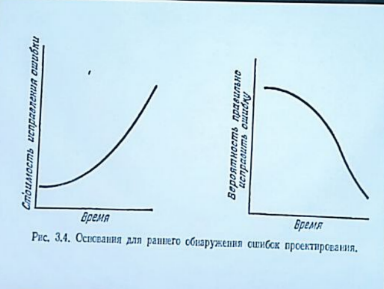
#### Методы отступления

Вторая подгруппа методов обеспечения устойчивости к ошибкам называется методом отступления или сокращенного обслуживания. Эти методы приемлемы обычно лишь тогда, когда для системы программного обеспечения существенно важно благопристойно закончить работу.

#### Методы изоляции ошибок

Основная их идея – не дать последствиям ошибки выйти за пределы как можно меньшей части системы программного обеспечения так, чтобы если ошибка возникает, то не вся система оказалось неработоспособной. Отключатся лишь отдельные функции в системе либо некоторые ее пользователи.

Из этих трех подгрупп методов обеспечения устойчивости к ошибкам только третья, изоляция ошибок, применима для большинства систем программного обеспечения.

****

**Средства и способы повышения надёжности ПО**

На основании методов обнаружения ошибок были разработаны следующие средства повышения надежности ПО:

* Средства, использующие временную избыточность
  + Авторизация доступа пользователей к системе
  + Анализ доступных пользователю ресурсов
  + Выделение ресурсов согласно ролям и уровням подготовки пользователей
  + Разграничение прав доступа пользователей к отдельным задачам, функциям управления, записям и полям баз данных
* Средства, использующие информационную избыточность
  + Целостность баз данных, обеспечиваемая за счёт системы внутренних уникальных ключей для всех информационных записей системы
  + Открытая система кодирования, позволяющая пользователю в любой момент изменять коды любых объектов классификации
  + Проверка значений контрольных сумм записей системы, обеспечивающая выявление всех несанкционированных модификаций (ошибок, сбоев) информации

Способы обеспечения и повышения надежности ПО

* Усовершенствование технологии программирования (например, формальное описание этапов программирования при помощи UML – Unified Modeling Language – унифицированный язык моделирования для графического описания бизнес-процессов, системного проектирования и отображения организационных структур)
* Выбор алгоритмов, не чувствительных к различного рода нарушениям вычислительного процесса (использование алгоритмической избыточности)
* Резервирование программ – N-версионное программирование
* Верификация программ с последующей коррекцией
* Валидация программ с последующей коррекцией

Основные проблемы исследований надежности ПО

* Разработка методов оценки и прогнозирования надежности ПО
* Определение основных факторов, влияющих на надежность ПО
* Разработка методов, обеспечивающих достижение заданного уровня надежности ПО
* Совершенствование методов повышения надежности ПО в процессе проектирования и эксплуатации

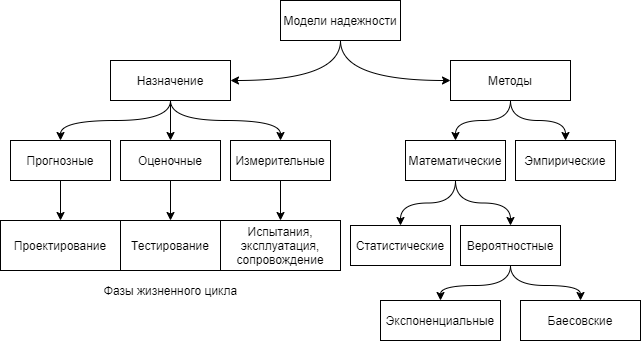
**Модели надежности программного обеспечения. Общие сведения**

Математические модели надежности ПО позволяют определить оценки показателей надежности ПО:

* Вероятность безотказной работы P(t)
* Интенсивность отказов λ(t)
* Математическое ожидание времени наработки на отказ mt
* Число ошибок в ПО до начала тестирования E0
* И др.

Все модели имеют допущения, которые приведены в описаниях моделей.

Классификация моделей надежности программ



Прогнозные модели

Используются для определения ожидаемого значения показателей надежности программного средства на этапе его проектирования. Примером прогнозной модели является уравнение ошибок, основанное на применении метрики Холстеда:

Где E – работа по программированию, трактуемая как общее число элементарных мысленных различений, составляющих все мысленные сравнения, которые требуются для создания программы, E0 – среднее число элементарных различений в сравнениях, которые требуются для обнаружения ошибки программирования (кодирования).

Если программный модуль пишут на диалекте английского языка, то уравнение ошибок, согласно Холстеду, преобразуется к виду:

Где V – объем программы в битах, который вычисляется по формуле:

Где N1 – общее количество операторов программы, N2 – общее количество операндов программы, η1 – число отличающихся операторов, η2 – число отличающихся операндов.

Аналитические модели

Аналитические (оценочные и измерительные) модели дают возможность рассчитать количественные показатели надежности, основываясь на данных о поведении программы в процессе тестирования. Представлены двумя группами: динамическими и статическими. В **динамических** МНПС поведение ПС (появление отказов) рассматривается во времени. В **статических** моделях появление отказов не связывают со временем, а учитывают только зависимость количества ошибок от числа тестовых прогонов (по области количества ошибок) или зависимость количества ошибок от характеристики входных данных (по области данных).

Эмпирические модели

Эмпирические модели базируются на анализе структурных особенностей программ. Они рассматривают зависимость показателей надежности от числа межмодульных связей, количества циклов в модулях, отношения количества прямолинейных участков программы к количеству точек ветвления и т. д. Эти модели целесообразно использовать на этапе проектирования ПС, когда осуществлена разбивка на модули и известна его структура. Эмпирические модели позволяют выявлять взаимосвязь между сложностью ПС и его надежностью.

Примеры моделей надежности программных средств

* Аналитические
  + Динамические
    - Дискретные
      * Модель Шика-Волвертона
      * Модель Шумана
    - Непрерывные
      * Модель Джелинского-Моранды
  + Статические
    - По области ошибок
      * Модель Миллса
      * Модель Липова
      * Простая интуитивная модель
      * Модель Коркорэна
    - По области данных
      * Модель Нельсона
* Эмпирические
  + Модель сложности
  + Модель, определяющая время доводки программ

**Модель Джелински–Моранды**

(модель роста надежности)

Допущения модели:

1. Время до очередного отказа распределено по экспоненциальному закону: P(t)=e-λt
2. Все ошибки равновероятны и их появление не зависит друг от друга
3. Частота появления ошибок (интенсивность отказов, функция риска) пропорциональна числу не выявленных ошибок:

Где E0 – число ошибок в ПО до начала тестирования и отладки, Kjm – коэффициент Джелинского-Моранды, tj – интервал времени между (i-1)-й и i-й обнаруженными ошибками, i – число ошибок, обнаруженных к моменту отладки t

1. λ(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

Время ti соответствует длительности выполнения ПО на компьютере и не учитывает простои компьютера для анализа результатов и их корректировки.

Так как согласно допущению I время до очередного отказа распределено по экспоненциальному закону, то вероятность безотказной работы, т. е. Отсутствие следующего i-го отказа равна

То плотность вероятности отказов равна:

Среднее время безотказной работы или средняя наработка до следующего i-го отказа определяется выражением:

Параметрами модели (исходными данными) является статистика об ошибках, т.е. интервалы времени между отказами.

**Характеристиками модели** (то, что нужно определить) являются:

* - вероятность отсутствия следующего i-го отказа;
* - среднее время до очередного i-го отказа;
* - коэффициент Джелинского-Моранды;
* - число ошибок в ПО до начала тестирования и отладки.

Характеристики модели определяются при помощи метода **максимального правдоподобия**.

**Пример.**

Оценка характеристик модели Джелинского-Моранды и осуществляется при помощи метода максимального правдоподобия.

Пусть для оценки характеристик модели проведено тестирование ПО. При этом фиксировать интервалы времени между отказами вследствии ошибок в ПО случайной величины **Т** - времени между отказами ПО.

Пусть интервалы времени между отказами ПО равны

=10 час.; =15 час.; =20 час.; =25 час.

Всего обнаружено n = 4 ошибки.

Введем понятие о функции правдоподобия L как о совместной n-мерной плотности распределения вероятностей события, заключающегося в том, что в процессе тестирования ПО случайная величина Т примет значение =10 час.; =15 час.; =20 час.; =25 час.

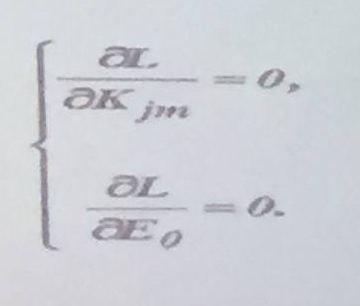
Функция правдоподобия согласно допущения 2 будет иметь вид:

или подставив значения функции плотности

В качестве оценок неизвестных параметров

и выбираются такие функции, которые максимизируют функцию правдоподобия

затем, на основании известных правил дифференциального исчисления для нахождения оценок максимального правдоподобия и составляется система уравнений



Выбирается то решение, которое обращает функция правдоподобия в максимум. Так как экстремум функций L и ln L достигается при одних и тех же значениях и то для упрощения вычислений пользуются логарифмической функцией правдоподобия.

Возьмем натуральный логарифм от функции правдоподобия, чтобы произведение функций заменить суммой:

Полагая и решая систему уравнений получаем:

1. оценка коэффициента Джелинского-Моранды
2. оценка числа ошибок в ПО до начала тестирования
3. интенсивность отказов до следующей пятой ошибки
4. средняя наработка до обнаружения следующей пятой ошибки
5. вероятность безотказной работы, т.е. отсутствие следующего пятого отказа

**Модель Шика–Волвертона**

Данная модель является модификацией модели Джелинского-Моранды и была предложена Шиком и Волвертоном. Дополнительно к допущениям модели Джелинского-Моранды используется следующее допущение: частота появления ошибки пропорционально времени отладки программы , т.е. вероятность обнаружения ошибки с течением времени должна возрастать.

где:

- число ошибок в ПО до начала тестирования и отладки

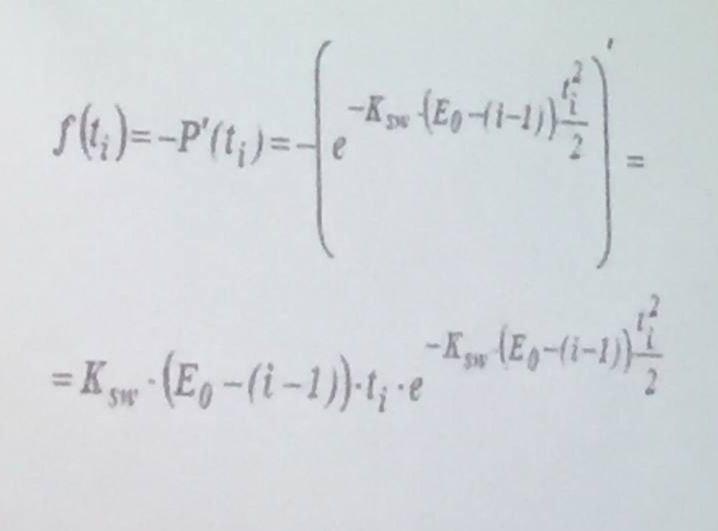
- коэффициент Шика-Волвертона

- интервал времени между (i-j)-ij и i-й обнаруженными ошибками

- число ошибок обнаруженных к моменту отладки

Для вероятности безотказной работы имеем следующее выражение:

Функция плотности будет иметь вид



**Пример**

Оценка характеристик модели Шика-Волвертона и осуществляется при помощи метода максимального правдоподобия.

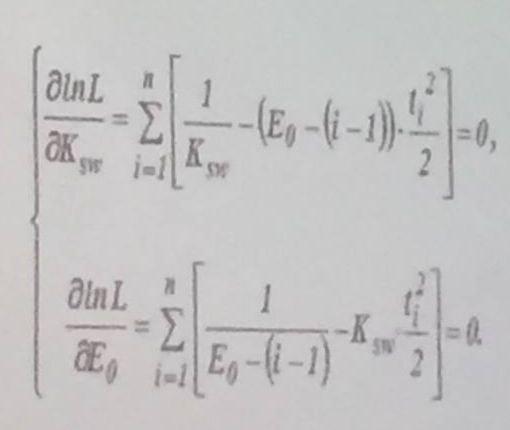
Пусть интервалы времени между отказами =7 час.; =12 час.; =23 час.; =17 час.; =10 час.

Пусть число найденных ошибок n=5

Функция правдоподобия будет иметь вид:

Возьмем натуральный логарифм от функции правдоподобия, чтобы произведение функций заменить суммой:

Возьмем частные производные по и и найдем условия экстремума:



Решая систему уравнений и полагая n=5, =, =, получаем оценку коэффициента Шика-Волвертона =0.001 и оценку числа ошибок =11

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

**Геометрическая модель**

(модель Моранды)

Основные исходные предпосылки для этой модели следующие:

* общее число ошибок неограниченно
* обнаружение ошибок не равновероятно
* обнаружение ошибок - процесс, независимый от ошибок
* ПО работает в условиях, близких к реальным
* интенсивность обнаружения ошибок образует геометрическую прогрессию, и она в интервале между появлениями ошибок постоянна

Интенсивность отказов имеет следующий вид

𝜆(0)=D - исходное значение интенсивности отказов

К - константа пропорциональности 0<K<1

- время между появлениями (i-1)-ой и i-ой обнаруженных ошибок.

**Пример**

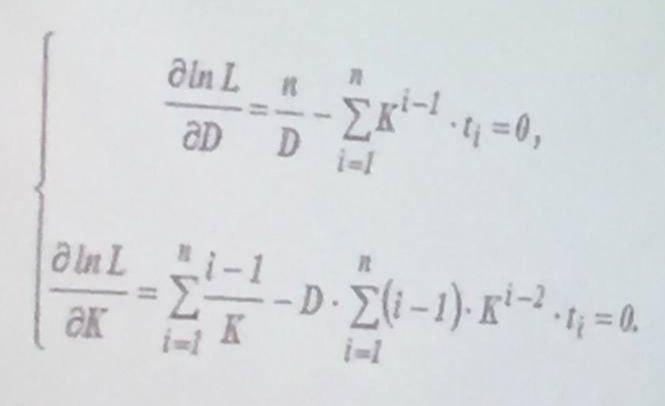
Нахождение характеристик геометрической модели методом максимального правдоподобия.

Пусть интервалы времени между отказами =7 час.; =12 час.; =23 час.; =17 час.; =10 час.

Пусть число найденных ошибок n=5

Функция правдоподобия примет вид:

Прологарифмировав и взяв частные производные по D и K, получим условие экстремума:



Полагая n=5, , и решая систему уравнений, получаем оценки максимального правдоподобия

= 0.892, =0.092

Среднее время обнаружения (n+1)-ой ошибки:

Значения интенсивности отказов на каждом интервале тестирования приведены в таблице

Значения интенсивности отказов

|  |  |
| --- | --- |
| Интервал | 𝜆() |
| 1 | 0.092 |
| 2 | 0.082 |
| 3 | 0.073 |
| 4 | 0.065 |
| 5 | 0.058 |

**Модель Шумана**

Исходные данные для модели Шумана, которая относится к динамическим моделям дискретного времени, собираются в процесс тестирования ПС в течение фиксированных или случайных временных интервалов.

Каждый интервал - это стадия, на которой выполняется последовательность тестов и фиксируется некоторое число ошибок.

В модели Шумана тестирование проводится в несколько этапов.

Для каждого этапа программа выполняется на полном комплексе разработанных тестовых данных.

Выявленные ошибки регистрируются (собирается статистика об ошибках), но не исправляются.

По завершении этапа на основе собранных данных о поведении ПС на очередном этапе тестирования модель Шумана используется для расчета количественных показателей надежности.

После этого исправляются ошибки, обнаруженные на предыдущем этапе, при необходимости корректируются тестовые наборы и проводится новый этап тестирования.

Скорость обнаружения ошибок пропорциональна числу оставшихся ошибок. Общее число машинных инструкций в рамках одного этапа постоянно.

Модель Шумана предполагает, что значение функции частоты отказов пропорционально числу ошибок, оставшихся в ПС после израсходованного на тестирование времени.

В процессе тестирования собирается информация о времени и количестве ошибок на каждом прогоне, т.е. общее время тестирования складывается из времени каждого прогона.

Предполагается, что интенсивность появления ошибок (число ошибок в единицу времени) постоянна.

Имея данные для двух различных моментов тестирования, для которых число найденных ошибок не совпадает, составляется уравнение и рассчитывается надежность программы.

Статические модели принципиально отличаются от динамических, прежде всего тем, что в них не учитывается время появления ошибок в процессе тестирования и не используется никаких предположений о поведении функции риска.

Эти модели строятся на статическом фундаменте.

**Модель Миллса**

В программу вносится некоторое количество известных ошибок s.

Эти ошибки вносятся случайным образом.

Затем делается предположение, что вероятность обнаружения при тестировании собственных и внесенных ошибок одинакова и зависит только от их количества.

Тестируя программу в течение некоторого времени и отсортировывая собственные и внесенные ошибки, можно оценить первоначальное число ошибок в программе ()

Оценка для по методу максимального правдоподобия будет следующей:

где:

s - количество внесенных в программу ошибок

n - число найденных собственных ошибок

v - число найденных внесенных ошибок

Достоинства модели:

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

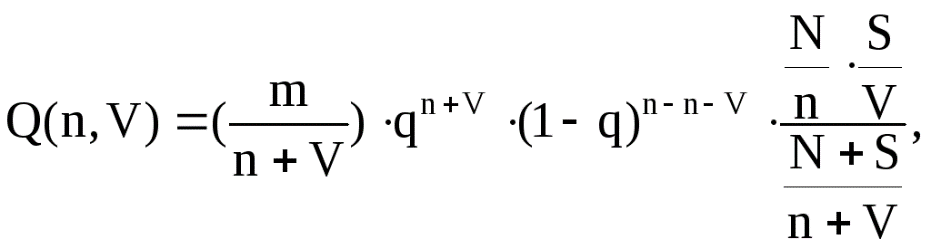
Недостаток модели:

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

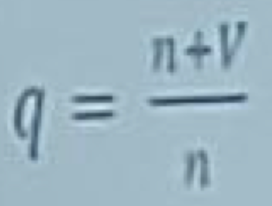
**Модель Липова**

Липов модифицировал модель Миллса, рассмотрев вероятность обнаружения ошибки при использовании различного числа тестов.

Если сделать то же предположение, что и в модели Миллса, т.е. Что собственные и искусственные ошибки имеет равную вероятность быть найденными, то вероятность обнаружения n собственных и V внесенных ошибок равна:



Где m - количество используемых тестов, q - вероятность обнаружения ошибки в каждом из m тестов, рассчитанная по формуле



S - общение количество искусственно внесенных ошибок, N - количество собственных ошибок, имеющихся в ПО до начала тестирования.

Достоинством модели Липова является то, что она позволяет оценить вероятность обнаружения некоторого количества ошибок к моменту оценки.

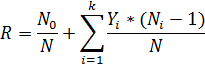
**Модель Коркорэна**

Применение модели предполагает знание следующих ее показателей:

Модель содержит изменяющуюся вероятность отказов для различных источников ошибок и соответственно разную вероятность их исправления

В модели используются такие параметры, как результат только N испытаний, в которых наблюдается Ni ошибок i-го типа

Выявление в ходе N испытаний ошибки i-го типа появляется с вероятностью ai.

Показатель уровня надежности R вычисляют по следующей формуле: 

Где N0 - число безотказных (или безуспешных) испытаний, выполненных в серии из N испытаний, k - известное число типов ошибок, Yi - вероятность появления ошибок, при Ni > 0, Yi = ai, при Ni = 0, Yi = 0.

**Модель Нельсона**

Данная модель предложена Нельсоном

В основе модели лежит представление программы в виде некоторой функции F(Ei), где Ei - набор значений данных, необходимых для выполнения прогона программы: 

Где:

E - множество всех значений наборов входных данных

N - число возможных наборов входных данных Ei.

Выполнение программы (прогон программы) приводит к получению для каждого определённого значения функции F(Ei).

Наличие дефектов в программе приводит к тому, что ей на самом деле соответствует функцию F’, отличная от заданной функции F.

Для некоторого Ei, отклонение F’ от F находится в некоторых допустимых пределах Delta i, т.е. для получения приемлемого результата необходимо следующее условие: 

Множество наборов входных данных Ei, неудовлетворяющие этому условию, т.е.

Образуют подмножество Ee множества e. Все такие случаи называются рабочими отказами

Вероятность Q того, что прогон программы приведет к рабочему отказу, равна вероятности, что набор входных данных Ei, использованный в данном прогоне, принадлежит множеству Ee:



Где:

N - число возможных наборов значений входных данных

ne - число различных наборов значения входных данных, содержащихся в Ee.

Вероятность того, что прогон программы на наборе входных данных Ei случайно выбранному из E среди равновероятных приведет к получению приемлемого результата

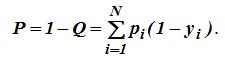
В процессе функционирования выбор входных данных из E обычно осуществляется не с одинаковыми вероятностями, а диктуется определенными условиями работы. Существует вероятность pi выбора входных данных Ei. Множество вероятностей pi называется функциональным разрезом.

Вероятность того, что прогон программы на взводных данных Ei, выбранных в соответствии с распределением вероятностей pi, закончится:

1) отказом

Где:

2) правильным выполнением единичного прогона программы:

Вероятность успешного выполнения n прогонов программы, при независимом выборе входных данных при каждом прогоне:



На практике выбор входных данных для каждого прогона нельзя считать независимым. Поэтому функциональный разрез должен быть переопределён в терминах вероятностей Pji выбора набора входных данных Ei при j-ом прогоне из некоторой последовательности прогонов.

Вероятность того, что j-ый прогон закончится отказом:



где:

i – вероятность выбора i-ого набора данных;

j – номер прогона программы.

Надежность P(n) программы равна вероятности безотказной работы n прогонов:



**Модель сложности. Использование метрик**  
Если придерживаться упрощенного понимания сложности ПС, то она может быть описана такими характеристиками, как размер ПС (количество программных модулей), количество и сложность межмодульных интерфейсов.

Под программным модулем в данном случае следует понимать программную единицу, выполняющую определенную функцию (ввод, вывод, вычисление и т.д.) и взаимосвязанную с другими модулями ПС.

Сложность модуля ПС может быть описана, если рассматривать структуру программы.

В качестве структурных характеристик модуля ПС используются: отношение действительного числа дуг к максимально возможному числу дуг, получаемому искусственным соединением каждого узла с любым другим узлом дугой; отношение числа узлов к числу дуг; отношение числа петель к общему числу дуг.

Оценка сложности производится с помощью синтаксического анализа исходного кода программы и на ее основе получают оценку надежности.

Метрики размера программ

**Метрика Холстеда**

Сложность V = N \* log2(n)

Где N - размер программы, n - размер словарая.

Число ошибок B = V / K,

Где К - некоторый коэффициент(3000)

Метрики сложности потока управления

**Метрика Маккейна**

Определяется циклометрическая сложность:

Z(G) = c - v + 2p

Где е - число вершин, v - число дуг, p - число компонентов связности ориентированного графа G.

Метрики сложности потока управления

**Метрика Джиббла**

Учитывает насыщенность программы операторами ветвления и глубину вложенности этих операторов.

Метрики сложности потока данных

**Метрика использования глобальных переменных**

Оценивает степень использования глобальных переменных.

**Метрика Чепина**

Сложность Q = a1 \* P + a2 \* M + a3 \* C + a4 \* T

Где P - входные и выходные переменные

M - модифицируемые переменные

C - управляющий переменные

T - неиспользуемые переменные

ai - некоторые коэффициенты

Используя метрики сложности можно получить оценку надежности с помощью некоторых весовых коэффициентов.

Основной проблемой является получение этих коэффициентов.

Их значение принимают исходя из опыта предыдущих проектов.

При быстром изменении технологий, методов и средств проектирования опыт предыдущих проектов учитывает свою актуальность.

**Имитационные модели**

Для сложных модулей и для больших многомодульных программ составляется имитационная модель.

Программа «засоряется» ошибками и тестируется по случайным входам.

Оценка надежности осуществляется по модели Миллса.

Полученные данные анализируются, проводится расчет показателей надежности по модели Миллса (или любой другой из описанных выше), и считается, что реальное ПС, выполняющее аналогичные функции, с подобными характеристиками и в реальных условиях должно вести себя аналогичным или похожим образом.

Преимущества оценки показателей надежности по имитационной модели, создаваемой на основе анализа структуры будущего реального ПС, заключаются в следующем:

модель позволяет на этапе проектирования ПС принимать оптимальные проектные решения, опираясь на характеристики ошибок, оцениваемые с помощью имитационной модели;

модель позволяет прогнозировать требуемые ресурсы тестирования;

модель дает возможность определить меру сложности программ и предсказать возможное число ошибок и т.д.

К недостаткам можно отнести:

• высокую стоимость метода, так как он требует дополнительных затрат на составление имитационной модели,

• приблизительный характер получаемых показателей.

**Модель, определяющая время доводки программ**

Модель используется для ПС, которые имеют иерархическую структуру, т.е. ПС как система может содержать подсистемы, которые состоят из компонентов, а те, в свою очередь, состоят из W модулей.

Таким образом, ПС может иметь W различных уровней композиции.

На любом уровне иерархии возможна взаимная зависимость между любыми парами объектов системы.

Все взаимозависимости рассматриваются в терминах зависимости между парами модулей.

Анализ модульных связей строится на том, что каждая пара модулей имеет конечную (возможно, нулевую) вероятность.

Изменения в одном модуле вызовут изменения в другом модуле.

Данная модель позволяет на этапе тестирования, а точнее при тестовой сборке системы, определять возможное число необходимых исправлений и время, необходимое для доведения ПС до рабочего состояния.

Основываясь на описанной процедуре оценки общего числа изменений, требуемых для доводки ПС, можно построить две различные стратегии корректировки ошибок:

1) фиксировать все ошибки в одном выбранном модуле и устранить все побочные эффекты, вызванные изменениями этого модуля, отрабатывая таким образом последовательно все модули;

2) фиксировать все ошибки нулевого порядка в каждом модуле, затем фиксировать все ошибки первого порядка и т.д.

Исследование этих стратегий доказывает, что время корректировки ошибок на каждом шаге тестирования определяется максимальным числом изменений, вносимых в ПС на этом шаге, а общее время — суммой максимальных времен на каждом шаге.

**Стандартизация оценки надежности программного обеспечения**

**Общие сведения о стандартизации и стандартах**

**Основной международной организацией**, **занимающейся** деятельностью по **стандартизации**, является Международная организация по стандартизации ИСО (International Standards Organization, ISO).

Стандарты, принимаемые данной организацией, получают аббревиатуру ISO и имеют ранг **международных**.

**Области электротехники, электроники, радиосвязи, приборостроения** не входят в компетенцuю ISO.

Стандартизацией в данных областях **занимаeтся** Международная электротехническая комиссия MЭK(International Electrotechnical Commission, IEC).

**В области стандартизации информационных технологий ISO и IEC объединили** свою деятельность, **создав** Объединенный технический комитет 1(Joint Technical Committee I,JTC1).

**В Республике Беларусь республиканским органом государственного управления по вопросам** технического нормирования, **стандартизации**, метрологии и оценки соответствия **является** Комитет по стандартизации, метрологии и сертификации при Совете Министров Республики Беларусь. (Госстандарт).

**В систему Госстандарта** входит Белopуccкuй госyдapcтвeнный инcтитyт cтандартuзации и cepтификaции (БeлГИСC), занимающийся вопросами технического нормирования, **стандартизации**, подтверждения соответствия, **управления качеством, нормативно-технического и информационного oбеспечения.**

*За рубежом и в Беларуси приняты следующие обозначения стандартов:*

* **вначале записывается** категория стандарта;

нaпpимep,

· **СТБ** -стандарт **Беларуси**,

·  **ГОСТ Р** - государственный стандарт **России**,

· **ГОСТ** - межгосударственный стандарт **для ряда стран СНГ, до распада ССCP**

аббревиатура ГОСТ обозначала государственный стандарт СССР,

· **ISO/IEC (ИСО/МЭК)** – международный стандарт **организаций ISO и IEC.**

· **Eсли стандарт разработан методом прямого применения** (например, является аутентичным переводом международного стандарта), то **за категорией стандарта** **следует** обозначение категории базового стандарта **(в русском именовании);**

Например,

· **CTБ ИCO/MЭK и ГOCT P ИCO/MЭK** – это **аутентичные переводы международного** стандарта **ISO/IEC**,

· **СТБ ГОСТ Р** - **стандарт Республики Беларусь, разработанный методом прямого применения** стандарта **России**.

· После категории базового стандарта **следует** номер стандарта.

· При этом, **если стандарт разработан методом прямого применения**, то **его номер совпадает с номером базового стандарта**;

например,

**базовый стандарт ISO/IEC 9126 имеет аутентичные стандарты**:

·  **Беларуси**

**CTБ ИCO/MЭK 9126**

**· и России**

**ГOCT P ИCO/MЭK 9126.**

· **Если стандарт состоит из нескольких частей,** то **после номера** стандарта **записывается номер его части;**

**например,**

**ISO/IEC 14598-1.**

· **После номера стандарта (или его части)** записывается **год его утверждения**;

например,

ISO/IEC 9126:1991,

CTБ ИCO/MЭK 9126-2003.

· Если **стандарт не находился в стадии разработки,** но **имеется необходимость в опубликовании его материалов,** то в **обозначении стандартов ISO/IEC, после категории дописывается** аббревиатура **TR (Technical report – Технический отчёт ТО)**

Например

**ISO/IEC TR 1527:1998**

**ГОСТ Р ИСО/МЭК ТО 15271-2002**

**Существующие стандарты** можно разделить на два основных типа:

* стандарты на продукты, определяющие их характеристики и требования к ним;
* Стандарты на процессы, определяющие конкретные методы создания продуктов.

**Стандарты в области оценки качества программных средств**

В настоящее время в области оценки качества ПС действуют следующие основные стандарты:

· Межгосударственный стандарт стран СНГ **ГОСТ 28195-99.** Оценка качества программных средств. Общие положения;

· Национальный стандарт Беларуси СТБ ИСО/МЭК 9126-2003. Информационные технологии. Оценка программной продукции. Характеристики качества и руководства по их применению.

· Международная серия стандартов **ISO/IEC 9126-1-4:2001-2004.** Программная инженерия – Качество продукта;

· Международная серия стандартов **ISO/IEC 14598-1-6:1998-2001**. Информационные технологии. Оценка программной продукции. Характеристики качества и руководства по их применению;

· Стандарт **СТБ ИСО/МЭК 9126-2003** представляет собой аутентичный перевод международного стандарта **ISO/IEC 9126:1991**

Данный стандарт под обозначением **ГОСТ Р ИСО/МЭК 9126-93** действует на территории России с 1994 года.

**Оценка надежности программных средств по ГОСТ 28195–99. Иерархическая модель надежности**

ГОСТ 28195-99 определяет оценку качества программного средства как совокупность операций, включающих:

* Выбор номенклатуры показателей качества оцениваемого программного средства
* Определение значений этих показателей
* Сравнение их с базовыми значениями

В соответствии с данным стандартом 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

Стандартом ГОСТ 28195-99 рекомендован метод интегральной оценки качества ПС, основанный на **иерархической модели качества.**

Надёжность является одной из шести основных характеристик качества ПС. С учётом этого рассмотрим данный метод оценки в применении к надёжности ПС.

В соответствии с данным методом выбор номенклатуры показателей надёжности для конкретного программного средства осуществляется с учётом значения и требований области применения в зависимости от принадлежности ПС к тому или иному подклассу, определяемому общесоюзным классификатором продукции (ОКП).

В стандарте ГОСТ 28195-99 приведена классификация ПС на подклассы в соответствии с ОКП.

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

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

Под показателем качества продукции в ГОСТ 28195-99 подразумевается количественная характеристика одного или нескольких свойств продукции, составляющих её качество, рассматриваемая применительно к определённым условиям её создания и эксплуатации или потребления.

Обеспечение и оценка качества ПС должны выполняться в течение жизненного цикла программных средств.

Основу описываемого метода оценки надёжности составляет четырёхуровневая иерархическая модель надёжности.

ГОСТ 28195-99 предлагает следующую терминологию для показателей качества (в том числе и надёжности) каждого уровня:

1. Факторы качества (в терминологии, принятой в международных стандартах, соответствуют характеристики качества, в данном случае это относится к надёжности

2. Критерии качества (в международной терминологии – подхарактеристики качества, в данном случае надёжности)

3. Метрики (соответствуют международной терминологии)

4. Оценочные элементы или единичные показатели (данный уровень в международных стандартах отсутствует)

Для фактора надёжности составляется четырёхуровневая иерархическая модель, отражающая её взаимосвязь с критериями, метриками и оценочными элементами. Вид данной модели зависит от фазы жизненного цикла ПС.

В качестве примера на рис 6.1 – 6.3 приведены три верхних уровня иерархической модели фактора надёжности для различных фаз жизненного цикла. Номера на данных рисунках соответствуют номерам метрик данного фактора.

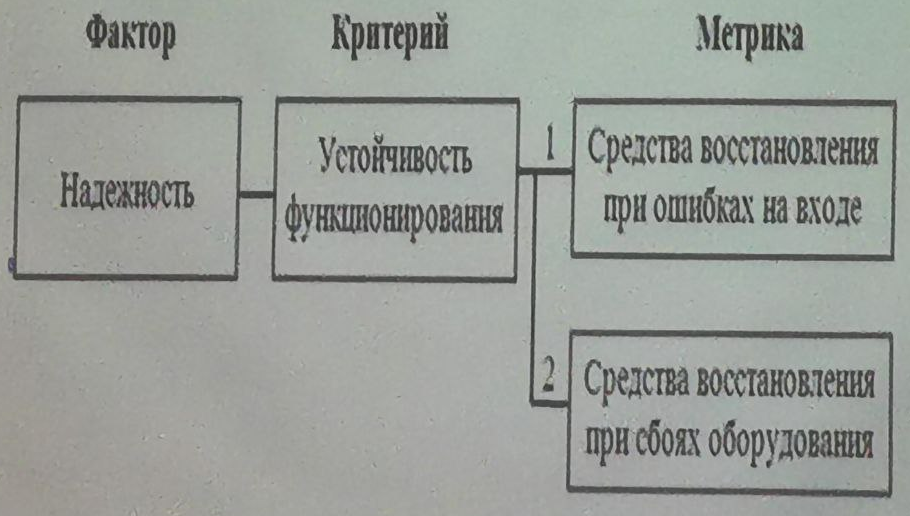
****

Рис 6.1 Модель надёжности для фазы анализа

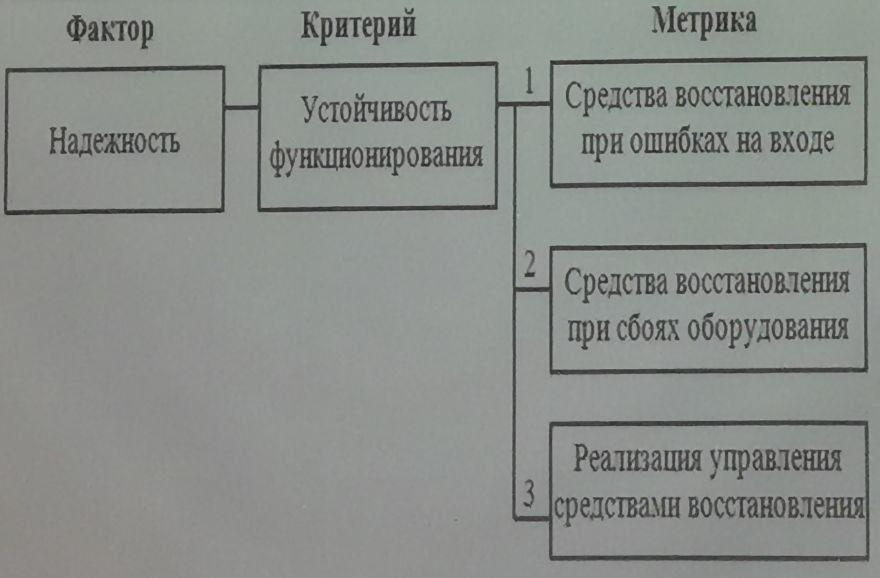
****

Рис 6.2 Модель надёжности для фазы проектирования

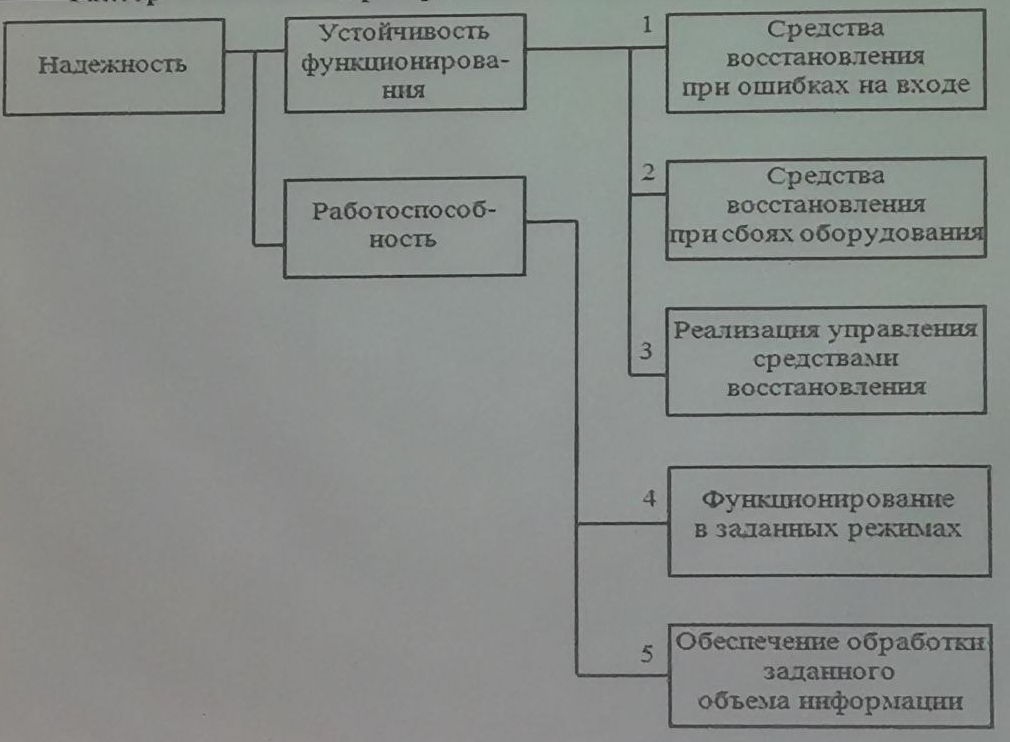


Рис 6.3 Модель надёжности для фаз реализации, тестирования, изготовления и сопровождения

**Оценка надежности программных средств по ГОСТ 28195–99. Оценочные элементы фактора надежности**

Выбор оценочных элементов в метрике зависит от функционального назначения ПС и формируется с учётом данных, ранее полученных при проведении испытаний ПС и эксплуатации аналогичных программ.

Для выбора оценочных элементов ГОСТ 28195-99 предлагает перечень таблиц, содержащих наименование элемента, метод оценки и применяемость элемента для различных подклассов ПС.

Таблица содержит перечень оценочных элементов для фактора надёжности.

В таблице код оценочного элемента состоит из пяти символов.

Первый символ (буква) указывает на принадлежности элемента фактору (Н – Надёжность)

Два следующих символа – номер метрики, который принадлежит оценочный элемент (для надёжности номера метрик обозначены на рис 6.1-6.3).

Четвертый и пятый символы – порядковый номер данного оценочного элемента в метрике.

Оценочные элементы фактора надёжности

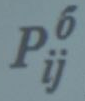
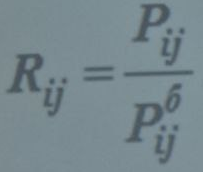
|  |  |  |  |
| --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка |
| Н0101 | Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных | Экспертный | 0 - 1 |
| Н0102 | Возможность обработки ошибочных ситуаций | -||- | 0-1 |
| Н0103 | возможность обработки ошибочных ситуаций. | -||- | 0-1 |
| Н0104 | наличие тестов для проверки допустимых значений входных данных | -||- | 0-1 |
| Н0105 | Наличие системы контроля полноты входных данных | -||- | 0-1 |
| H0106 | наличие средств контроля корректности входных данных | -||- | 0-1 |
| H0107 | наличие средств контроля непротиворечивости входных данных | -||- | 0-1 |
| H0108 | наличие проверки параметров и адресов по диапазону их значений | -||- | 0-1 |
| H0109 | наличие обработки граничных результатов | -||- | 0-1 |
| H0110 | наличие обработки неопределенностей | -||- | 0 - 1 |
| H0201 | наличие требований к программе по восстановлению процесса выполнения в случае сбоя операционной системы, процессора, внешних устройств | -||- | 0 - 1 |
| H0202 | наличие требований к программе по восстановлению результатов пи отказа процессора, операционной системы | -||- | 0 - 1 |
| H0304 | наличие средств, обеспечивающих выполнение программы в сокращённом объёме в случае ошибок и помех | -||- | 0 - 1 |
| H0305 | Показатель устойчивости к искажающим воздействиям P(Y) | Регистрационный + Расчётный | P(Y) = 1 - D/K  где: D - число экспериментов, в которых искажающие воздействия приводят к отказу;  K - число экспериментов, в которых имитируются искажающие воздействия;  Y - вид искажающего воздействия |
| H0401 | Вероятность безотказной работы Р | Регистрационный + Расчётный | P = 1 - Q/N.  где: N - число экспериментов; Q - число зарегистрированных отказов |
| H0501 | Оценка по среднему времени восстановления Q | Измерительный + Расчётный | Q = {1, если Tв <= Tвдоп;  {Tвдоп/Tв, если Tв > Tвдоп;  где: Tв - среднее время восстановления;    N - число восстановлений;  Tвi - время восстановления после i отказа  Tвдоп - допустимое среднее время восстановления |
| H0502 | Оценка по продолжительности преобразования i-го входного набора данных в выходной | Измерительный + Расчётный | где: Tniдоп - допустимое время преобразования i-го входного набора данных;  Tni - фактическая продолжительность преобразования i-го входного набора данных. |

**Оценка надежности программных средств по ГОСТ 28195–99. Последовательность оценки**

Оценка надёжности программного средства проводится в следующей последовательности:

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

* абсолютного показателя Pij;
* относительного показателя Rij

Относительный показатель Rij является функцией показателя Pij и его базового значения  и определяется по формуле 

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

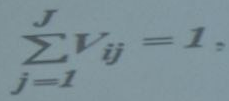
Базовые значения для надёжности и её показателей третьего уровня формируются методом экспертного опроса с учётом назначения ПС или на основании показателей существующих аналогов или расчётного эталонного ПС.

Значения базовых показателей ПС должны соответствовать значениям показателей, отражающих современный уровень качества и прогнозируемый мировой уровень.

4) Каждый показатель надёжности 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:

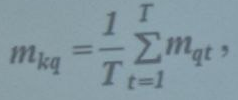


где: J - общее количество всех показателей J-го уровня, относящихся к i-му показателю вышестоящего уровня, определённых в стандарте.

ГОСТ 28195-99 содержит таблицы, содержащие перечни весовых коэффициентов для показателей второго и третьего уровней (критериев и метрик)

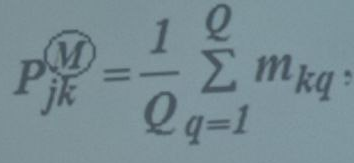
Количественные величины весовых коэффициентов зависят от фазы жизненного цикла ПС и подкласса ПС

5)Определение усреднённой оценки Mkq оценочного элемента по нескольким его значениям (измерениям) Mqt осуществляется по следующей формуле (формула для вычисления значений показателей надёжности 4 уровня):



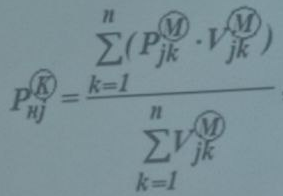
Где: k - порядковый номер метрики; q - порядковый номер оценочного элемента; T - число значений (измерений) оценочного элемента; t - номер значений оценочного элемента

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



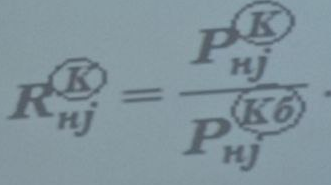
Где: М - признак метрики; Q - число оценочных элементов, реально используемых при оценке k-й метрики

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

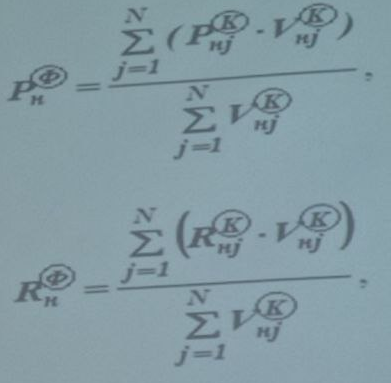


где: n - число метрик, относящихся к j-му критерию, реально используемых при оценке; K - признак критерия

8) Относительные значения  j-го критерия надёжности  по отношению к базовому значению к6 определяется по формуле



9) Абсолютные и относительные значения надёжности определяются по формулам:

****

где: Ф - признак фактора; N - число критериев надёжности, реально используемых при оценке

* **Оценка надежности программных средств по СТБ ИСО/МЭК 9126–2003. Модель процесса оценки надежности**

**Стандарт СТБ ИСО/МЭК 9126-2003 регламентирует метод оценки** качества ПС, **основанный на трёхуровневой иерархической модели качества**

На **первом уровне** модели находятся **6 характеристик качества**

**Второй уровень** составляют **подхарактеристики** и **третий** - **метрики качества**

* **Надёжности** включает три подхарактеристики
* 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, повреждённые в случае отказа. Одним из показателей восстанавливаемости является длительность восстановления.

Метод оценки качества, регламентированный в СТБ ИСО/МЭК 9126-2003, в применении к надёжности ПС

Процесс оценки состоит из трёх стадий:

определение требований к надёжности ПС,

подготовка к оцениванию,

процедура оценивания.

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

**Оценка надежности программных средств по СТБ ИСО/МЭК 9126–2003. Определение требований к надёжности. Подготовка к оцениванию**

1. Определение требований к надёжности

Целью данной стадии является установка требований в терминах подхарактеристик надёжности.

Требования выражают потребности внешнего окружения ПС и должны быть определены до начала разработки.

Так как ПС разделяется на компоненты, требования для ПС в целом могут отличаться от требований для отдельных компонентов.

1. Подготовка к оцениванию

Целью второй стадии является подготовка основы для оценивания

Данная стадия состоит из трёх этапов

* выбор метрик надёжности
* определение уровней ранжирования
* определение критерия оценки

2.1 Выбор метрик надёжности

С учётом регламентрованной в СТБ ИСО/МЭК 9126-2003 иерархической модели качества уровень характеристики надёжность ПС определяется уровнем входящих в неё подхарактеристик, а значения подхарактеристик в свою очередь, определяются значениями входящих в них метрик

В стандарте СТБ ИСО/МЭК 9126-2003 набор рекомендуемых метрик отсутствует. КАждый количественный признак и каждое количественно оцениваемое взаимодействие ПС с его окружением, которые соотносятся с надёжностью, могут быть приняты в качестве её метрики.

2.2 Определение уровней ранжирования

Для измерения количественных признаков надёжности ПС используются метрики.

Измеренные значения отображаются на некоторой шкале.

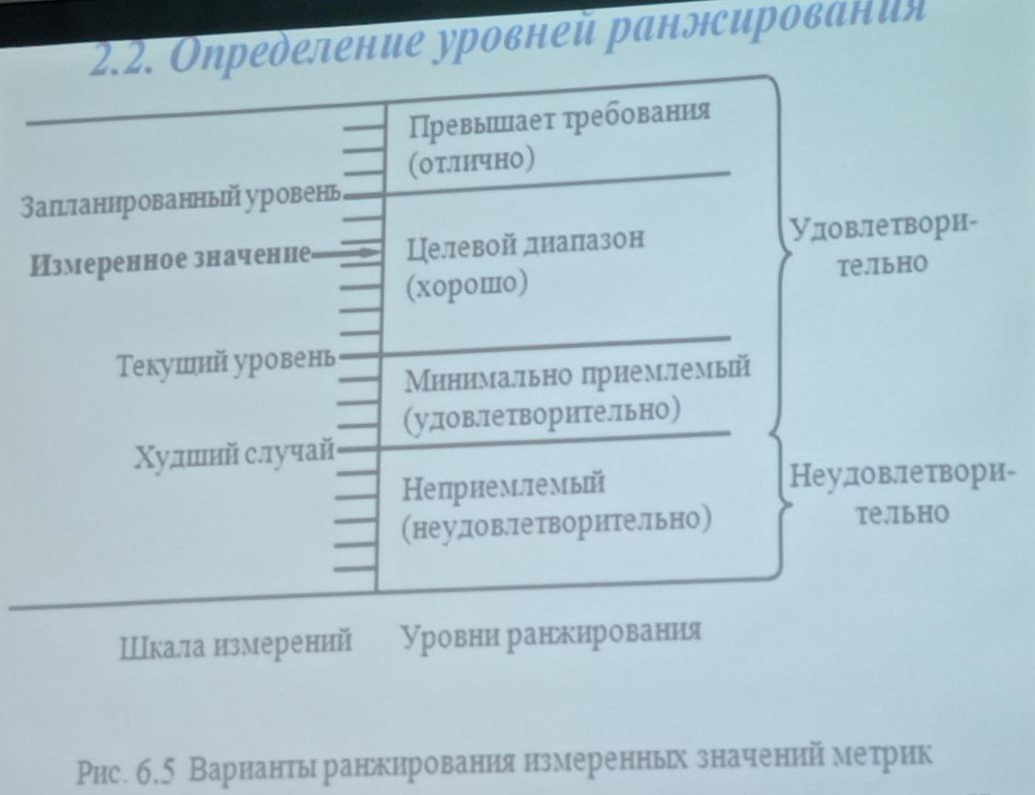
Данные значения не показывают уровень удовлетворения требований к надёжности ПС. Для этой цели шкалы метри должны быть разделены на диапазоны, соответствующие различным степеням удоволетворения требований.

В стандарте ISO/IEC 14598-1:1999 приведён пример следующих диапазонов ранжирования:

* разделение шкалы на **2 категории**:

неудовлетворительно и удовлетворительно

* разделение шкалы на 4 категории (отлично, хорошо, удоволетворительно, неудоволетворительно), ограниченные соответственно **запланированным уровнем, текущим уровнем** для существующего или альтернативного продукта, и **уровнем худшего случая**.



**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.

Для определения общей надёжности 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 косвенным факторами надёжности ПС.

**Оценка надежности программных средств по СТБ ИСО/МЭК 9126–2003. Процедура оценивания**

Последняя стадия модели процесса оценивания реализуется **тремя этапами:**

3.1 Измерение

Для измерения выбранные метрики применяются к ПС. Результатом являются значения в масштабах метрик.

3.2 Ранжирование

На этапе ранжирования устанавливается уровень ранжирования для измеренного значения.

3.3 Оценка

Результатом является заключение о надёжности ПС (приемлемы ии неприемлемы уровень надёжности

**Общие сведения о международных стандартах в области качества ПС**

В настоящее время в области оценки качества ПС за рубежом действуют следующие основные стандарты:

**Международная серия стандартов SQuaRE** – Системная и программная инженерия – Требования к качеству и оценка программного продукта

Модель качества ПС регламентирует относящийся к данной серии стандарт ISO/IEC 25010:2011 – Системная и программная инженерия – Требования к качеству и оценка программного продукта (SQuaRE) – Модели качества систем и программных средств

* + **Международный стандарт ISO/IEC 9126–2:2003** – Программная инженерия – Качество продукта – Часть 2: Внешние метрики
  + **Международный стандарт ISO/IEC 9126–2:2003** – Программная инженерия – Качество продукта – Часть 3: Внутренние метрики
  + **Международный стандарт ISO/IEC 9126–2:2003** – Программная инженерия – Качество продукта – Часть 4: Метрики

Данные стандарты регламентируют **иерархическую модель качества программных средств**. На верхнем уровне модели находятся **характеристики.** Характеристики разделяются на **подхарактеристики.** Подхарактеристики определяются **мерами** (**метриками**). Метрики измеряют **атрибуты** (**свойства**) ПС.

В стандартах серии ISO/IEC 9126 регламентированы следующие виды метрик:

* + Внутренние метрики
  + Внешние метрики
  + Метрики качества в использовании

При этом под **метрикой** понимается метод измерения и шкала измерения некоторого свойства ПС.

**Внешние метрики** – это метрики, предназначенные для измерения качества программного продукта путем измерения поведения системы, 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.

**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.

**Метрики качества в использовании** – это метрики, измеряющие соответствие продукта потребностям заданных пользователей в достижении заданных целей с результативностью, продуктивностью, безопасностью и удовлетворением в заданных контекстах использования. Данные метрики могут использоваться только в процессе эксплуатации ПС в реальной среде окружения. Метрики качества в использовании основаны на измерении поведения типичных пользователей и системы, содержащей данное программное средство.

* **Модель надежности программных средств по стандарту ISO/IEC 25010:2011**

В стандарте определены две модели качества:

· Модель качества продукта

· Модель качества в использовании

Данные модели имеют трехуровневую иерархическую структуру (уровни характеристик, подхарактеристик и метрик). Уровни характеристик и подхарактеристик являются регламентированными.

В соответствии с ISO/IEC 25010:2011 **надёжность** (**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. **Подхарактеристиками надёжности** являются:

* + Завершенность
  + Готовность
  + Устойчивость к ошибке
  + Восстанавливаемость

**Завершённость** (**стабильность, Maturity**) – степень соответствия системы, 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 системы, продукта или компонента тогда, когда требуется их использование. Готовность может быть оценена соотношением времён, в течение которых система, продукт или компонент находится в работоспособном или неработоспособном состоянии. Готовность представляет собой комбинацию завершенности (обуславливающей частоту отказов), устойчивости к ошибке и восстанавливаемости (определяющей продолжительность времени восстановления после каждого отказа).

**Устойчивость к ошибке (Fault tolerance)** – степень функционирования системы, 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 переустановки требуемого состояния системы в случае прерывания или отказа продукта или системы. Продолжительность этого времени определяется ее восстанавливаемостью.

**Внутренние метрики надежности программных средств по стандарту ISO/IEC 9126-3:2003**

**Внутренние метрики надежности** используются во время разработки программного средства для предсказания того, удовлетворяет ли ПС заявленным потребностям в надежности.

**Внутренние метрики завершённости** определяют набор атрибутов для оценки завершенности ПС.

**Внутренние метрики устойчивости к ошибке** определяют набор атрибутов для оценки способности программного продукта поддерживать желаемый уровень качества функционирования в случае ошибок выполнения или нарушения заданного интерфейса.

**Внутренние метрики восстанавливаемости** определяют набор атрибутов для оценки способности программного продукта восстанавливать соответствующий уровень качества функционирования и данные, поврежденные в случае отказа.

**Внутренние метрики соответствия** **надежности** определяют набор атрибутов для оценки способности программного продукта соответствовать стандартам, соглашениям или нормативам организации пользователя, связанным с надежностью.

|  |  |  |  |
| --- | --- | --- | --- |
| Название метрики | Формула | Исходные данные для вычисления метрики по соответствующей формуле | Источники данных для измерения |
| **Завершенность** | | | |
| Обнаружение ошибок | X = A / B | А – количество ошибок, обнаруженных при проверке, В – количество предполагаемых ошибок, которое должно быть обнаружено при проверке (используя предыдущую историю или соответствующую модель) | А – отчёт о проверке, В – база данных организации |
| Устранение ошибок | X = A / B | А – количество исправленных ошибок проектирования/ кодирования, В – общее количество ошибок, обнаруженных при проверке | А – отчет об устраненных ошибок, В – отчет о проверке |
| Полнота тестирования | X = A / B | А – количество тестовых вариантов, спроектированных в плане тестирования и подтвержденных при проверке, В – количество требуемых тестовых вариантов | А – план тестирования, В – спецификация требований |
| **Устойчивость к ошибке** | | | |
| Предотвращение отказов | X = A / B | А – количество типовых ошибок, обработка которых предусмотрена при проектировании/ кодировании, В – количество типовых ошибок, которые должны быть учтены | А – отчет о проверке, В – спецификация требований |
| Предотвращение некорректных действий | X = A / B | А – количество функций, реализованных с предотвращением некорректных действий, В – количество типичных некорректных действий, которое должно быть учтено | А – отчет о проверке, В – спецификация требований |
| **Восстанавливаемость** | | | |
| Способность к восстановлению | X = A / B | А – количество реализованных требований к восстановлению, подтвержденное при проверке, В – общее количество требований к восстановлению, определённых в спецификации | А – отчёт о проверке, В – спецификация требований |
| Эффективность восстановления | X = A / B | А – количество реализованных требований к восстановлению, удовлетворяющих заданному временному ограничению, В – количество требований к восстановлению, определенных в спецификации временными ограничениями | А – отчёт о проверке, В – спецификация требований |

**Внешние метрики надежности программных средств по стандарту ISO/IEC 9126-2:2003**

Внешние метрики надежности должны измерять свойства, связанные с поведением системы, содержащей ПС, во время тестирования, чтобы показать степень надежности ПС в этой системе в процессе эксплуатации.

**Внешние метрики завершенности** должны позволять измерять такие атрибуты, как отсутствие отказов в ПС, вызываемых ошибками, существующими в самом ПС.

**Внешние метрики устойчивости к ошибке** должны быть связаны со способностью ПС поддерживать заданный уровень качества функционирования в случаях ошибок выполнения или нарушения заданного интерфейса.

**Внешние метрики восстанавливаемости** должны позволять измерять такие атрибуты, как способность ПС в системе восстанавливать соответствующий уровень качества функционирования и данные, поврежденные в случае отказа.

**Внешние метрики соответствия надежности** должны позволять измерять такие атрибуты, как количество функций или случаев с проблемами соответствия, вызывающими неудовлетворение требований стандартов, соглашений и нормативов, связанных с надежностью.

|  |  |  |  |
| --- | --- | --- | --- |
| Название метрики | Формула | Исходные данные для вычисления метрики по соответствующей формуле | Источники данных для измерения |
| **Завершенность** | | | |
| Оценочная плотность скрытых ошибок | X = (Abs (A1 – A2)) / B | А1 – общее количество прогнозируемых скрытых ошибок в программном продукте, А2 – общее количество реально обнаруженных в течение определенного испытательного срока ошибок, В – размер продукта | А1 – модель предсказания роста надежности, А2 – отчет о тестировании, отчет об эксплуатации, отчет о проблеме |
| Плотность отказов по тестовым вариантам | X = A / B | А – количество обнаруженных в течение определенного испытательного срока отказов, В – количество выполненных тестовых вариантов | А, В – отчет о тестировании, отчет об эксплуатации, отчет о проблеме |
| Устранение отказов | X = A / B | А – количество устраненных в течение определенного испытательного срока отказов, В – общее количество реально обнаруженных отказов | А, В – отчет о тестировании, отчет об эксплуатации |
| Плотность ошибок | X = A / B | А – количество ошибок, обнаруженных в течение определенного испытательного срока, В – размер продукта | А – отчет о тестировании, отчет об эксплуатации, отчет о проблеме |
| Устранение ошибок | X = A / B или Y = A / C | А – количество исправленных ошибок, В – общее количество реально обнаруженных при тестировании ошибок, С – общее количество прогнозируемых скрытых ошибок в программном продукте | А, В – отчет о тестировании, С – база данных организации |
| Тестовое покрытие | X = A / B | А – количество реально выполненных тестовых вариантов, представляющих сценарий эксплуатации во время тестирования, В – количество тестовых вариантов, которые должны быть выполнены, чтобы покрыть требования | А – отчет о тестировании, отчет об эксплуатации, В – спецификация требований, спецификация тестирования |
| Тестовая завершенность | X = A / B | А – количество прогонявшихся тестовых вариантов во время тестирования или эксплуатации, В – количество тестовых вариантов, которые должны быть выполнены, чтобы покрыть требования | А – отчет о тестировании, отчет об эксплуатации, В – спецификация требований, спецификация тестирования |
| **Устойчивость к ошибке** | | | |
| Предотвращение аварийных отказов | X = 1 – A / B | А – количество аварийных отказов, В – общее количество отказов | А, В – отчет о тестировании, отчет об эксплуатации |
| Предотвращение отказов | X = A / B | А – количество случаев предотвращения типичных и серьезных отказов во время выполнения тестовых вариантов, направленных на проверку типичных ошибок, которые могут привести к отказу, В – количество выполненных тестовых вариантов, направленных на проверку типичных ошибок, которые могут привести к отказу | А, В – отчет о тестировании, отчет об эксплуатации |
| Предотвращение некорректных действий | X = A / B | А – количество случаев предотвращения критических и серьезных отказов, В – количество выполненных при тестировании тестовых вариантов, направленных на проверку типовых некорректных действий, которые могут привести к отказу | А, В – отчет о тестировании, отчет об эксплуатации |
| **Восстанавливаемость** | | | |
| Среднее время простоя | X = T / B | Т – общее время простоя, В – количество отмеченных аварийных отказов | Т, В – отчет о тестировании, отчет об эксплуатации |
| Среднее время восстановления | X = Sum (T) / B | Т – время восстановления неработающей системы программного обеспечения в каждом соответствующем случае, В – количество случаев, когда наблюдаемая система программного обеспечения восстановлена | Т, В – отчет о тестировании, отчет об эксплуатации |
| Способность к повторному запуску | X = A / B | А – число повторных запусков, удовлетворяющих требуемому времени, во время тестирования или поддержки пользовательской эксплуатации, В – общее число повторных запусков во время тестирования или поддержки пользовательской эксплуатации | А, В – отчет о тестировании, отчет об эксплуатации |
| Способность к восстановлению | X = A / B | А – количество случаев успешного восстановления, В – количество случаев восстановления, протестированных согласно требованиям | А – отчет о тестировании, отчет об эксплуатации, В – спецификация требований, спецификация тестирования |
| Эффективность восстановления | X = A / B | А – количество случаев успешного восстановления, удовлетворяющих заданному времени, В – количество выполненных случаев восстановления, в которых должно быть удовлетворено заданное время восстановления | А, В – отчет о тестировании, отчет об эксплуатации |
| **Готовность** | | | |
| Готовность | X = Tэ / (Tэ + Tв) или Y = A / B | Tэ – время эксплуатации, Tв – время восстановления, А – общее число случаев успешного использования программного средства пользователем, В – общее число попыток использования ПС пользователем за время наблюдения | Tэ, Tв, А, В – отчет о тестировании, отчет об эксплуатации |

**Тестирование программных средств**

**Основные определения тестирования программного обеспечения. Этапы тестирования**

**Тестирование ПО (software testing)** - процесс анализа ПС и сопутствующей документации для выявления дефектов и повышения качества продукта

**Тестирование ПО является процессом** в силу того факта, что оно выполняется большим кол-вом людей на протяжении длительного периода времени.

Тестировать нужно:

* Программы при их непосредственном запуске и исполнении (software)
* Код программ без запуска и исполнения (code)
* Прототип программного продукта (product prototype)
* Проектную документацию (project documentation)
  + Требование к программному продукту (product requirements)
  + Функциональные спецификации к программному продукту (functional specification)
  + Архитектуру (architecture )и дизайн (design)
  + План проекта (project plan) и тестовый план (test plan)
  + Тестовые случаи сценарии (test cases)
* Сопроводительную документацию (и документацию для пользователей)
  + Интерактивную помощь (on-line help)
  + Руководства по установке (Installation guide) и использованию программного продукта (user manual)

**Дефект (баг, defect, bug) -** любое несоответствие фактического и ожидаемого результата (согласно требованиям и здравому смыслу).

**Ожидаемый результат (expected result) -** такое поведение ПС, которое мы ожидаем в ответ на наши действия.

**Тест-кейс (test case) -** набор входных данных, условий выполнения и ожидаемых результатов, разработанных с целью проверки того или иного свойства или поведения ПС

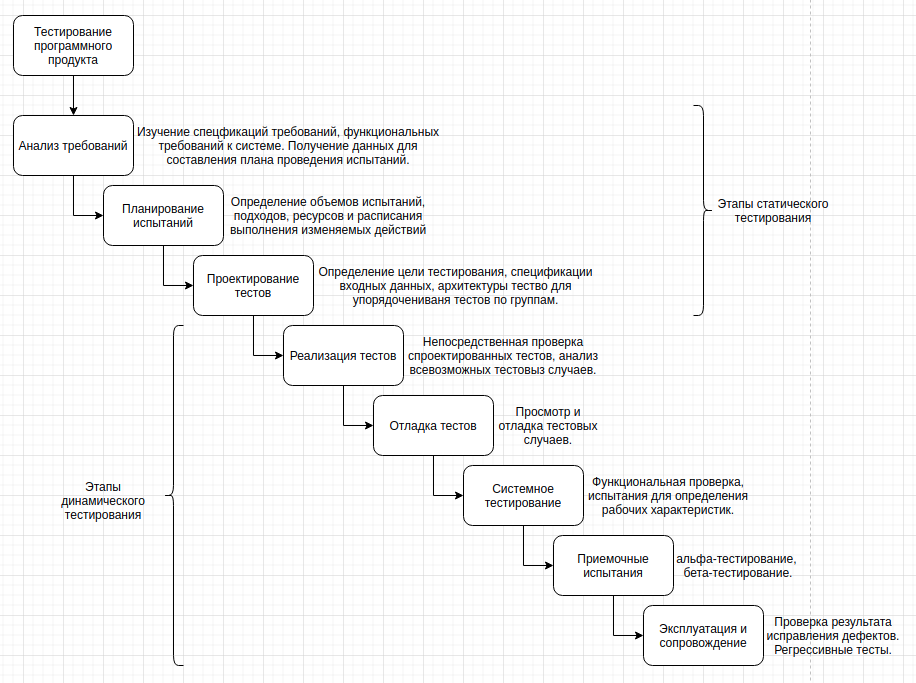
**Тест-план (test plan) -** часть проектной документации, описывающая и регламентирующая процесс тестирования.

**Билд (build) -** промежуточная версия ПС (финальный билд часто называют релизом (release)).

**Статическое тестирование (static testing) -** это процесс анализа самой разработки ПО, иными словами, это тестирование без запуска программы (проверка кода, требований, функциональной спецификации, архитектуры. дизайна и т.д.)

**Динамическое тестирование (dynamic testing) -** это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Этапы тестирования:



**Уровни и виды тестирования**

**Метод белого ящика (white-box testing, glass-box testing)** - используется для тестирования программного кода без запуска.

Тестировщик имеет доступ к исходному коду ПС. Тесты основаны на знании кода приложения и его внутренних механизмов.

Метод белого ящика часто используется на стадии, когда приложение еще не собрано воедино, но необходимо проверить каждый из его компонентов, модулей, процедур и подпрограмм. Как правило, выполняется разработчиками ПС.

**Метод черного ящика (black-box testing)** заключается в том, что тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь.

Тестирование черного ящика ведется с использованием спецификаций или иных документов, описывающих требования к системе на основе применения пользовательского интерфейса для ввода I/O данных. **Цель данного метода** - проверить работу всех функций приложения на соответствие функциональным требованиям.

**Метод серого ящика (gray box testing)** - совокупность подходом из белого и черного ящика.

Этот метод, как правило, используется при тестировании веб-приложений, когда тестировщик знает принципы функционирования технологий, на которых построено приложение, но может не видеть кода самого приложения.

## 4. УРОВНИ И ВИДЫ ТЕСТИРОВАНИЯ

**Компоненты тестирования (component testing, unit testing)** - тестирование отдельного модуля ПС (под модулем может пониматься отдельный класс, метод и тд)

**Интеграционное тестирование (integration testing)** - проверка того, как отдельные компоненты, проверенные на предыдущем уровне, взаимодействуют друг с другом.

**Системное тестирование (system testing)** - полная проверка приложения: проверяются как функциональные, так и модульные требования.

**Функциональное тестирование** - процесс проверки ПО, сконцентрированный на анализе соответствия ПО требованиям и спецификациям. Функциональное тестирование может быть **ручным** (manual) или **автоматизированным** (automatic).

**Приемочный тест (smoke test)** - самый первый и быстрый уровень. Если тесты этого уровня не проходят, тестирование прекращается.

**Тест критического пути (critical path test)** - проверка таких функций приложения, с которыми обычный пользователь работает в повседневной жизни.

**Расширенный тест (extended test)** - полная проверка приложения на соответствие всем требованиям к нему.

**Инсталляционное тестирование** - проверка всего того, что связано с инсталляцией продукта в систему и удалением продукта из системы.

**Регрессионное тестирование** - проверка того, что внесенные в приложение изменения не привели к потере работоспособности того, что ранее работало, и/или привели к потере работоспособности того, что ранее не работало.

**Тестирование нового функционала** - проверка того, что заявленный в данном билде новый функционал работает должным образом.

**Конфигурационное тестирование** - проверка того, как приложение работает с различным оборудованием и (или) собственными настройками.

**Тестирование совместимости** - проверка того, как приложение взаимодействует с другими приложениями и ОС. В случае веб-ориентированных приложений особое внимание уделяется совместимости с различными браузерами.

**Тестирование удобства использования** - проверка того, насколько пользователю удобно и приятно работать с приложением.

**Тестирование интернационализации** - проверка готовности продукта к переводу на различные языки.

**Тестирование локализации** - проверка качества перевода продукта на конкретный язык.

**Позитивное тестирование** - проверка того, как приложение работает в заведомо “тепличных условиях” (корректных данных)

**Негативное тестирование** - проверка того, как приложение реагирует на заведомо невалидные данные.

**Исследовательское тестирование** - редкий вид тестирования, основанный на профессиональной интуиции тестировщика, которая может подсказать опасные и малоисследованные способы работы с приложением.

**Работа с документацией. Типы требований. Уровни требований. Документирование требований**

Именно в требованиях берет начало большинство багов *(а не в коде, как думает этот долбаеб)*

Тестировать нужно:

* Проектную документацию (project documentation)
  + Требования к программному продукту
  + Функциональные спецификации к программному продукту
  + Архитектуру и дизайн
  + План проекта
  + Тестовые случаи сценария
* Сопроводительную документацию (и документацию для пользователей):
  + Интерактивную помощь
  + Руководства по установке и использованию программного продукта

5.2 Основные типы требований

Чаще всего выделяют два типа требований:

1. Функциональные требования - определяют, “что система должна делать”
2. Нефункциональные требования - определяют, “с соблюдением каких условий” должно происходить это действие (например, скорость отклика при выполнении заданной операции)

5.3 УРОВНИ МАТЬ ВАШУ ТРЕБОВАНИЙ

|  |  |  |
| --- | --- | --- |
| Уровень | Требования (В чем/как выражаются) | Что описывает |
| 1 | Бизнес требования (Общее видения и обзорная документация) | Зачем вообще нужен ПП и что с его помощью будет делаться. Например, “Нам нужен инструмент, извлекающий бизнес-информацию из различных источников и представляющий ее в виде диаграмм и таблиц”. |
| 2 | Пользовательские требования  (Use cases) | Задачи, которые пользователь может выполнять с помощью ПП, и реакцию ПП. Например: “Когда пользователь заходит в систему, должен появляться текст приветствия”. |
| 3 | Функциональные и нефункциональные требования  (Требования к ПО) | **Функциональные требования** описывают поведение системы, например: “В процессе инсталляции программа должна проверять остаток свободного места на всех несъемных носителях”.  **Нефункциональные требования** описывают специфические свойства системы, такие как: требования к удобству использования, безопасности, надежности, расширяемости и т.д. Например: “При одновременной работе с системой 1000 пользователей, минимальное время между возникновением сбоя должно быть более или равно десяти тысячам часов”. |

5.4 ВАЖНОСТЬ ТЕСТИРОВАНИЯ ТРЕБОВАНИЙ

Хорошо проработанные требования позволяют:

* Выработать общее понимание м/у заказчиком и разработчиком
* Определить рамки проекта
* Более точно определить финансовые и временные характеристики проекта.
* Обезопасить заказчика от риска получить продукт, в котором он не сможет работать.
* Обезопасить разработчика от риска попасть в ситуацию “неконтролируемого размытия границ”, которое может привести к непредвиденным затратам ресурсов сверх начальных ожиданий

Основными путями выполнения требований являются:

* Интервью
* Наблюдения
* Самостоятельное описание
* Семинары
* Прототипирование

5.6 ДОКУМЕНТИРОВАНИЕ ТРЕБОВАНИЙ

Стандарты содержат развернутое описание требований, которое может быть оптимизировано для нужд конкретной организации.

В описании требований рекомендуется включать:

1. Введение
2. Назначение документации
3. Поддерживаемые соглашения
4. Предполагаемая аудитория и рекомендации по последовательности работы с документом для каждого класса читателей.
5. Границы проекта. Здесь содержится ссылка на документ “Концепция”, если таковой имеется, либо краткое резюме продукта.
6. Ссылки.

**Работа с документацией. Проблемы с требованиями. Работа с требованиями**

**Проблемы незавершенности (неполноты)**

Хорошо, когда вся важная информация присутствует. Только вот вопрос — а как можно найти то, чего нет, но должно быть? Как догадаться, что что-то отсутствует?

Следует обратить внимание на такие типичные случаи.

**Отсутствуют нефункциональные требования или нефункциональные составляющие требования.** Мы знаем, что система должна делать. А про то, как (как быстро, как безопасно) в лучшем случае узнаем только в самом конце.

**Проблемы с требованиями**

**Проблемы незавершенности (неполноты) (продолжение)**

Итак, мы уточнили какие-то требования, и заказчик высказал свои предпочтения. Например, ему нужно, чтобы приложение работало «надёжно и быстро».

Думаем, как мы сможем проверить эти требования? Проверить, что приложение работает быстро. А как проверить? И что проверять?

**Вывод: кроме самого нефункционального требования, нам обязательно нужны критерии - как это требование проверить (и измерить).** Чем важнее требование для заказчика, тем точнее должны быть эти критерии.

**Чтобы легче было понять суть потребностей заказчика, рекомендуется задать общие вопросы.**

Их преимущества:

* Они **универсальные** - большая часть их применима к любому проекту, независимо от специфики продукта.
* Они **не навязывают решение** - больше шансов что, заказчик случайно упомянет то, что для него очевидно (и поэтому он нам об этом раньше не говорил).
* Они **не загонят заказчика в ситуацию, когда ему приходится выбирать из имеющихся вариантов** (а на самом деле все будет по тому варианту, который вы забыли упомянуть).

Хорошая аналогия = вопросы репортера (или маленького ребёнка): «А что?», «А зачем?», «А почему?».

**Ещё одной проблемой чрезмерной общности утверждений является т.н. ТВО** («to be defined», «будет определено»).

Мы можем успокоиться (на время) ‚ только если знаем, кто и когда должен определить эти ТВ. И почему они не определены сейчас.

Также бывает, что стороны, ответственные за закрытие ТВО, просто забывают об этом. Вывод? Нужно напоминать.

Бывают противоречия как внутри одного требования, так и между двумя и более требованиями. Бывают противоречия между таблицами и текстом, картинкой и текстом. Противоречия между требованием и прототипом.

Для устранения противоречий в требованиях, **надо хотя бы один раз прочитать ВСЕ требования** (звучит страшно, но такова жизнь) — не только ту часть, за которую отвечаете именно вы, — чтобы как минимум знать о чём там говорится и представлять, где ещё есть вещи которые могут вас касаться.

Во-вторых, надо **убедиться, что одно и тоже называется во всех частях документации одним и тем же словом** -— тут помогает глоссарий.

В-третьих, изучая «требования про А», надо проверить, не противоречат ли они:

* другим «требованиям про А»
* требованиям, более общим, покрывающим «требования про А»
* требованиям, связанным с «требованиями про А»
* требованиям про что-то другое, но аналогичным «требованиям про А»
* как, следуя «требованиям про А», можно нарушить другие требования

**Проблемы двусмысленности**

Если что-то можно понять несколькими способами, можно быть уверенным, что разные люди поймут это по-разному.

Например, В требовании сказано, что «функциональность Х» является опциональной.

Что думает отдел разработки? «Чудесно. Опционально — значит необязательно. Можно не реализовывать.»

Что думает отдел маркетинга? «Ага. Мы выпустим две версии продукта. В более дорогой будет эта функциональность.»

Что думает заказчик? «Я за те же деньги получу ещё и вот эту функциональность».

Вывод? Следует писать требования так, чтобы исключить ВОЗМОЖНОСТЬ их двоякого понимания.

**Проблемы непроверяемости**

Для начала следует отметить, что все вышеперечисленные проблемы ведут в том числе к тому, что мы не можем проверить, удовлетворяет ли продукт требованию.

Как понять, что требование непроверяемо? Попробовать придумать несколько тестов для его проверки. Если тесты не придумываются - вот она проблема.

А бывает ли «просто непроверямое требование»? Да.

Например: «В приложении должно быть ноль ошибок», «Приложение должно поддерживать все версии всех операционных систем».

**Еще раз самое важное**

**Если нельзя проверить требование - это проблема.**

В конечном итоге именно тестировщики отвечают за то, чтобы требование было проверено. Если нельзя проверить требование по объективным причинам, то его необходимо уточнять до тех пор, пока оно не станет проверяемым. Для этого нужно расспрашивать заказчика, разработчиков, своих более опытных коллег и т.д.

**В случае очень важных и нерешаемых простыми способами проблем, решение будет приниматься менеджером проекта и заказчиком.**

**Набор немодифицируем**

Это значит, что после каждого обновления требований понадобится тратить недели чтобы выловить все появившиеся противоречия.

**Набор непроранжирован**

Можно потратить кучу усилий на неважную функциональность а самое основное не успеть. Или можно потратить много времени на то, что полностью изменится в новой версии требований, и работу придется повторять заново. Можно обмануть ожидания заказчика, надеявшегося получить тот или иной набор функций к такому-то времени.

**Набор непрослеживаем**

Если требования не пронумерованы, не имеют четкого оглавления, не имеют работающих перекрестными ссылок - это хаос. А в хаосе ошибки плодятся с удивительной скоростью.

***Работа с требованиями***

Одна из наиболее распространенных техник работы с требованиями -— взаимный перепросмотр.

Суть взаимного перепросмотра требований проста: **после того, как один человек создал требование, другой человек это требование проверяет.**

Обычно, выделяют три уровня перепросмотра:

**1) Неформальный** **перепросмотр.** Двое коллег просто обмениваются листиками (файликами) и правят найденные ошибки, которые потом обсуждаются за чашкой чая или в любое другое относительно свободное время.

**2) Технически перепросмотр.** Это немного более формализованный процесс, требующий подготовки, выделенного времени, участия некоторой группы специалистов (желательно, из различных областей).

**3) Формальная инспекция.** Проводится редко и в случае очень больших проектов и крайней необходимости. Описывается специальными стандартами, требует соблюдения широкого спектра правил и протоколирования результатов.

**Тест-кейсы, чек-листы. Признаки и классы эквивалентности**

**Тест-кейс** (test case) - набор **входных данных, условий выполнения и ожидаемых результатов,** разработанный с целью проверки того или иного свойства или поведения программного средства.

**Чек-лист** (check-list) - список **идей тестов.**

**Какие бывают тесты**

Основные **виды тестов:**

* позитивные;
* негативные.

**Направления** тестирования:

* статическое;
* динамическое.

**Методы** тестирования:

* чёрный ящик;
* белый ящик;
* серый ящик.

**Виды тестирования:**

* инсталляционное;
* регрессионное;
* нового функционала;
* конфигурационное;
* совместимости;
* удобство использования;

**Классы эквивалентности: подсказка**

* Размер файла (особенно кратный чему-либо).
* Объём памяти (особенно кратный чему-либо).
* Размер экрана, разрешение экрана.
* Размер окна.
* Количество цветов. Цветовую гамму.
* Версии операционной системы.
* Версии библиотек.
* Время (в т.ч. между событиями).
* Скорость передачи данных.
* Объём передаваемых данных.
* Интенсивность передачи данных.
* Переключение между разными алгоритмами (количество, время, скорость).

**Выводы:**

* Классы эквивалентности не всегда очевидны
* Как правило, негативных тестов получается больше, чем позитивных
* Принадлежность теста к позитивным или негативным зависит от требований

**Рекомендации по разработке тестов**

* Начинать необходимо с простых очевидных тестов. Используйте простые и очевидные значения для передачи в программу. Если она завалится даже на таких значениях, это будет очевидным показателем того, что существуют проблемы.
* Затем следует переходить к более сложным тестам. Если программа хорошо справляется с очевидными задачами, поставьте перед ней неочевидную.
* Используйте граничные условия. На граничных значениях можно построить много хороших тестов.
* Если остаётся время, занимайтесь исследовательским тестированием.

**Хороший тест-кейс**

Хороший тест-кейс удовлетворяет следующим критериям:

* Обладает высокой вероятностью обнаружения ошибки.
* Исследует соответствующую (“ту, которую надо”) область приложения.
* Выполняет какие-то интересные действия.
* Не выполняет ненужных действий.
* Является не слишком простым, но и не слишком сложным.
* Не является избыточным по отношению к другим тестам.
* Делает обнаруженную ошибку очевидной.
* Позволяет легко диагностировать ошибку.

**Документирование тестов. Тестовый сценарий**

Тесты документируются в виде «тест-кейсов», или «тестовых случаев» («test-cases»)

Под тест-кейсом понимается набор тестовых входных данных, условий выполнения и 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.

В общем случае отдельные поля шаблона тест-кейса могут опускаться.

При написании тест-кейсов следует:

1)  использовать простой технический стиль;

2) в русском языке использовать безличную форму: «открыть», «закрыть» и т.п. (вместо «откройте»);

3) описывать поведение систем следующим образом: «появляется окно», «приложение закрывается» и т.п.;

4) обязательно указывать точные названия всех элементов приложения;

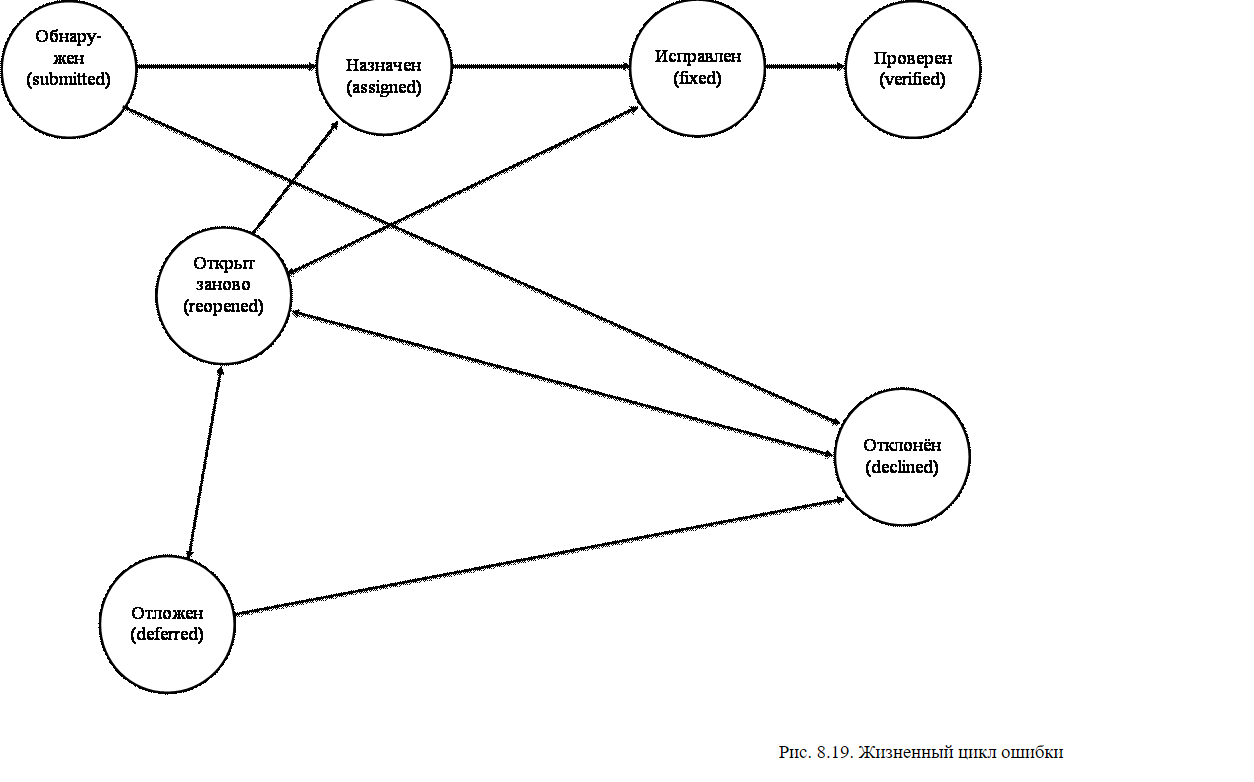
5) не объяснять базовые понятия работы с операционной системой.

Обычно тестовый набор связан с тестовым сценарием, хотя его можно выполнить и не определяя такую связь.

Тестовый сценарий - это неавтоматизированный или автоматизированный сценарий, содержащий инструкции по реализации тестового набора. Тестовый сценарий может быть написан вручную (для выполнения человеком) либо полностью или частично автоматизирован. Кроме того, автоматизированные тестовые сценарии для проверки функциональности, производительности или защиты можно связать с тестовым набором.

**Дефекты, жизненный цикл дефектов, отчеты о дефектах**

Дефект – брак или баг



**Отчёт о дефекте**- это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Цели написания отчёта о дефекте

Предоставить информацию о проблеме — уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы.

Приоритизировать проблему — определить степень опасности проблемы для проекта и желаемые сроки её устранения.

Содействовать устранению проблемы

**Атрибуты отчёта об ошибке**

В общем отчёт об ошибке представляет собой технический документ, написанный с целью:

- 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 системы.

Часто данные системы интегрируются с системами управления проектами, так как ошибка в разрабатываемом приложении является угрозой для качественного выполнения проекта в целом. Типовая стандартная форма отчёта об ошибках, используемая в ведущих IT-компаниях, содержит следующие основные поля, которые заполняются как тестировщиком, так и менеджером проекта [25]:

1) Краткое описание(Summary).

2) Версия приложения, в котором была обнаружена ошибка(Build found) .

3) Симптом (Symptom) ошибки.

- некорректная операция (incorrect operation).

- повреждение/потеря данных (data corruption/loss).

- крах системы (system crash).

- косметический дефект (cosmetic flaw).

- проблема в документации (documentation issue)..

- нереализованная функциональность (missing feature).

4) Серьёзность (Severity) ошибки. Это поле показывает, насколько серьёзна найденная ошибка. Обычно выделяют следующие уровни серьёзности:

- критическая (critical) ошибка;

- высокая (major) ошибка.;

- средняя (medium) ошибка;

 -низкая (minor) ошибка.

5) Приоритет (Priority) ошибки. Это поле показывает, как быстро необходимо исправить ошибку. Обычно выделяют следующие уровни приоритета:

- наивысший (ASAP, as soon as possible);

- высокий (high);

- обычный (normal);

- низкий (low).

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

6) 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.

7) Шаги для воспроизведения ошибки (Steps to reproduce). Данное поле приводится для того, чтобы воспроизвести ошибку в кратчайшие сроки. Это поле следует заполнять максимально подробно.

8) 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.

**Особенности тестирования веб-ориентированных приложений. Технологии. Структура. Режимы работы. Запуск и остановка. Формирование интерфейса.**

1. ***Технологические отличия*.**

Классическое приложение работает с использованием одной или семейства родственных технологий.

Web-приложение работает с использованием принципиально различных технологий.

1. ***Структурные отличия*.**

Классическое приложение “монолитное”. Состоит из одного или небольшого количества модулей. Не использует серверы БД, web-серверы и т.д.

Web-приложение — “многокомпонентное”. Состоит из большого числа модулей. Обязательно использует серверы БД, web-серверы, серверы приложений.

1. ***Отличия режимов работы.***

Классическое приложение работает *в режиме реального времени*, т.е. известно о действиях пользователя сразу же, как только оно выполнено.

Web-приложение работает в режиме “запрос-ответ”, т.е. известно о некотором наборе действий только после запроса на сервер.

Особенности тестирования web-приложений, режим работы

1. ***Отличия формирования интерфейса*.**

Классическое приложение использует для формирования интерфейса пользователя относительно устоявшиеся и стандартизированные технологии.

Web-приложение использует для формирования пользовательского интерфейса стремительно развивающиеся технологии, множество которых конкурирует между собой.

1. ***Отличия запуска и остановки*.**

Классическое приложение запускается и останавливается редко.

Web-приложение запускается и останавливается по факту поступления каждого запроса, т.е. очень часто.

верной, и клиентской части.

**Особенности тестирования веб-ориентированных приложений. Сеть. Количество пользователей. Сбои и отказы. Инсталляция. Удаление. Среда функционирования. Компоненты. Архитектура.**

1. ***Отличия работы с сетью*.**

Классическое приложение практически не использует сетевые каналы передачи данных.

Web-приложение активно использует сетевые каналы передачи данных.

1. ***Разница в количестве пользователей*.**

Классическое приложение: количество пользователей, одновременно использующих приложение, подвержено контролю, ограничено и легко прогнозируемо.

Web-приложение: количество пользователей, одновременно использующих приложение, сложнопрогнозируемо и может скачкообразно меняться в широких диапазонах.

1. ***Особенности сбоев и отказов*.**

Классическое приложение: выход из строя тех или иных компонентов сразу становится очевидным.

Web-приложение: выход из строя некоторых компонентов оказывает непредсказуемое влияние на работоспособность приложения в целом.

1. ***Отличия в инсталляции*.**

Классическое приложение — процесс инсталляции стандартизирован и максимально ориентирован на широкую аудиторию пользователей. Не требует специфических знаний. Добавление компонентов приложения выполняется стандартным способом с использованием одного и того же инсталлятора.

Web-приложение — процесс инсталляции часто недоступен конечному пользователю. Инсталляция требует специфических знаний. Процесс изменения компонент приложения не предусматривается или требует квалификации пользователей. инсталлятор отсутствует.

1. ***Отличия в деинсталляции.***

Классическое приложение: процесс деинсталляции стандартизирован и выполняется автоматически или полуавтоматически.

Web-приложение: процесс деинсталляции требует специфических знаний для вмешательства администратора и часто сопряжен с изменением кода среды функционирования приложения, БД, настройки системного ОС.

1. ***Особенности среды функционирования*.**

Классическое приложение: среда функционирования стандартизирована и не сильно влияет на функционирование приложения.

Web-приложение: среда функционирования очень разнообразна и может оказать серьезное влияние на работоспособность и серверной, и клиентской части.

**Тестирование по методу белого ящика. Инструментальные средства тестирования веб-ориентированных приложений по методу белого ящика**

Тестирование методом белого ящика (также: прозрачного, открытого, стеклянного ящика; основанное на коде или структурное тестирование) – метод тестирования программного обеспечения, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Тестирование белого ящика – углубление во внутренне устройство системы, за пределы ее внешних интерфейсов.

**тестирование белого ящика – это:**

– тестирование, основанное на анализе внутренней структуры компонента или системы.

– тест-дизайн, основанный на технике белого ящика – процедура написания или выбора тест-кейсов на основе анализа внутреннего устройства системы или компонента.

**Тестирование по методу белого ящика. Валидаторы кода. Ссылки**

Валидатор — это позволяет пользователям Интернета проверять документы HTML и XHTML до HTML5 на предмет правильной разметки по определению типа документа.

Так же, как и с проверкой грамотности языка, HTML-код можно проверять вручную — своими глазами и мозгами, а можно пользоваться и автоматическими помощниками. Это может быть отдельный целостный сервис, а может быть дополнение к браузеру. Первое лучше. Инструменты валидации HTML-кода онлайн облегчают жизнь разработчика, которому не нужно самому вычислять, например, парность скобок.

Firebug – это полнофункциональный отладчик и редактор, который позволяет вам работать с HTML, JavaScript, CSS, DOM и многими другими страницами. Вы также можете использовать расширение для мониторинга JavaScript, CSS и XML в режиме реального времени, искать ошибки, которые могут быть в них, и узнавать, что вам нужно сделать, чтобы их исправить. Являясь важным инструментом практически в каждом арсенале инструментов дизайнера, Firebug стал настолько обычным явлением, что даже начал получать свои собственные расширения

**Виды, направления и цели тестирования приложений под нагрузкой**

Нагрузочное тестирование или тестирование производительности — это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

Тестирование производительности (Performance testing)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций

определение количества пользователей, одновременно работающих с приложением

определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)

исследование производительности на высоких, предельных, стрессовых нагрузках

Стрессовое тестирование (Stress Testing)

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

Объемное тестирование (Volume Testing)

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций

может производиться определение количества пользователей, одновременно работающих с приложением

Тестирование стабильности или надежности (Stability / Reliability Testing)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.

**Основные задачи ramp-up теста**

Скорость наращивания скорости во время тестирования нагрузки – это скорость, с которой новые одновременные пользователи пытаются получить доступ к приложению. Обычно в режиме реального времени каждое приложение имеет свои часы пик, где одновременные пользователи, как правило, медленно увеличиваются до начала пикового времени. Скорость, с которой пик трафика коснулся серии времени известен как Ramp скорость загрузки испытаний. Наша цель всегда должна быть, чтобы соответствовать нарастить скорость с как можно ближе к пользовательской модели

**Основные задачи rush hour и longevity тестов**

В рамках нагрузочного тестирования и тестирования производительности, как правило, выполняются следующие основные тесты.

Тест на выживаемость (longevity test) показывает способность системы работать длительное время под высокой нагрузкой. Одной из наиболее опасных проблем, выявляемых данным тестом, является утечка памяти и иное снижение эффективности использования аппаратных ресурсов из-за накапливающихся со временем ошибок в работе приложения.

Тест "часа пик" (rush hour test) позволяет оценить реакцию системы на резкое изменение нагрузки. Во время теста проверяется способность системы выдержать скачкообразное увеличение нагрузки во время "часа пик", а также способность системы вернуться к изначальным показателям производительности после завершения "часа пик" (восстановления исходных показателей нагрузки на систему). Такой тест позволяет выявить проблемы с синхронизацией выполнения отдельных участков кода, а также проблемы с управлением всеми видами межкомпонентного взаимодействия (в т.ч. сетевых и локальных соединений) на всех уровнях системы.

**Этапы проведения и отчетность тестирования производительности**

Тестирование производительности проводится для обеспечения бесперебойной работы и снижения стоимости владения ПО на всех этапах жизненного цикла решения.

* анализ системы и подбор требований;
* подготовка стратегии;
* настройка генератора нагрузки;
* проведение мониторинга серверов и генератора нагрузки;
* подготовка тестовых данных;
* разработка нагрузочных скриптов;
* предварительные запуски тестов;
* проведение тестирования;
* анализ результатов и подготовка отчёта.

**Инструментальные средства тестирования производительности**

SilkTest

SilkTest – инструмент автоматизированного тестирования прикладных программ через графический интерфейс пользователя. Программа предназначена для регрессионного, кросс-платформенного и локализационного тестирования для широкого диапазона технологий разработки приложений, в том числе Web, Java, .NET и клиент-серверных технологий. Затрачивая меньше времени на создание и обслуживание тест комплектов регрессионного тестирования, специалисты по обеспечению качества (QA) смогут расширить тестовое покрытие и повысить качество приложения.

Основные возможности:

- базовая последовательность выполнения Basic Workflow для мгновенного достижения производительности;

- управляемая данными последовательность выполнения Data Driven Workflow для тестирования функций бизнес-логики с помощью данных, хранящихся во внешних источниках;

- функция Code Completion для быстрой настройки тестов и разработки инфраструктуры автоматизации интеграция с коллективными коммуникационными средствами Rational;

- уровень абстракции GUI для простого обслуживания многократно используемых тестов и сценариев;

- SilkTest предоставляет уровень абстракции GUI, так называемые карты отображения GUI (GUI maps), сопоставляющие графические элементы тестируемого приложения с объектами более низкого уровня, генерируемого SilkTest;

- выполнение проверок баз данных с помощью стандартного доступа через ODBC, что поможет гарантировать точность выполнения сложных транзакций.

HP QuickTest Professional

Поставщик: Hewlett Packard (HP)

Quicktest Professional (QuickTestPro или QTP) – основной инcтрумент автоматизации функционального тестирования. QTP позволяет автоматизировать функциональные и регрессионные тесты путем записи действий пользователя при работе с тестируемым приложением и дальнейшего исполнения записанных действий с целью проверки работоспособности ПО. Записанные действия сохраняются в виде скриптов. Скрипты отображаются как VBScript (Expert View) или как визуальные последовательные шаги с действиями (Keyword View). Каждый шаг скрипта может быть отредактирован и дополнен точками проверки (CheckPoint) для сравнения ожидаемого результата с фактическим.

Основные возможности:

- проверка функциональности через точки проверки (CheckPoints);

- обработка исключительных ситуаций (Exception Handling);

- формирование данных в таблицы с последующим использованием подхода Data-Driven Testе;

- работа со сложными UI-объектами;

- расширяемость за счет дополнительных модулей;

- формирование отчетов с результатами выполнения тестирования.

Rational Functional Tester

Поставщик: IBM Rational Software

Rational Functional Tester – инструмент автоматизированного тестирования, позволяющий выполнять функциональное тестирование, регрессионное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными (data-driven тестирование).

Основные возможности:

- использование технологии ScriptAssure и проверка изменения прикладных данных;

- интеграция с коллективными коммуникационными средствами Rational;

- поддерживает тестирование приложений 3270 (zSeries) и 5250 (iSeries) с использованием расширения Functional Tester Extension для приложений на основе терминалов;

- интеграция в среду Eclipse, WebSphere Studio и Rational XDE Developer.

- позволяет тестировщикам автоматизировать тестирование, устойчивое к частым изменениям пользовательского интерфейса приложений, благодаря технологии ScriptAssure™.

- выполняет проверку динамических данных с использованием различных мастеров, точек проверки и шаблонов регулярных выражений;

- использование автоматизированного мастера для создания тестов, управляемых данными, позволяет повысить полноту тестирования за счет многократного использования тестов с различными наборами данных.

- допускает использование ключевых слов для частичной автоматизации ручного тестирования.

- позволяет тестировщикам выбрать язык сценариев для разработки и настройки тестов: Java в среде Eclipse или Microsoft Visual Basic .NET в среде Visual Studio .NET.

- поддерживает пользовательские элементы управления благодаря прокси-объекту SDK (Java/.Net).

- поддерживает функциональное тестирование сред приложений Oracle ERP посредством поставляемых расширений.

TestComplete

Поставщик: SmartBear Software

TestComplete – полнофункциональная система для автоматизации тестирования приложений. С помощью TestComplete можно выполнять функциональное, узловое, регрессионное, распределенное тестирование, а также тестирование работоспособности HTTP на проектном уровне.

Основные возможности:

- встроенный редактор кода, помогающий тестировщикам писать скрипты вручную;

- режим Record/PlayBack с записью только ключевых действий;

- отладчик скриптов для построчного выполнения теста;

- распознавание объектов приложений, написанных на Delphi, C++Builder, .Net, WPF, Java и Visual Basic;

- TestComplete включает в себя шаблоны основных баг-трекенговых систем: Microsoft Visual Studio 2005, 2008, 2010 Team System, BugZilla и AutomatedQA AQdevTeam;

- движок TestComplete основан на открытом API, COM интерфейсе. Он независим от языка программирования и может быть использован через TestComplete Debug Info Agent по ходу выполнения;

- TestComplete позволяет автоматически делать снимки экрана во время записи и выполнения тестов. Это позволяет проводить быстрое сравнение ожидаемого и фактического результатов.

Selenium

Selenium – это инструмент для тестирования Web-приложений. Selenium – объектно-ориентированное JavaScript приложение для анализа файлов определенной структуры находить в них команды для манипуляции браузером и команды для выполнения определенных действий и проверок. Selenium поддерживается Microsoft Internet Explorer, Google Chrome, Mozilla Suite и Mozilla Firefox для Microsoft Windows, Linux, и Apple Macintosh.

Selenium IDE предоставляет утилиту для записи и последующего воспроизведения автотестов. Тестовые скрипты могут быть автоматически записаны и изменены вручную. Selenium Remote Control (RC) – сервер, написанный на Java, который воспринимает команды для браузера через HTTP. RC делает возможным написание тестов для Web-приложений на любом удобном языке программирования, что позволяет интегрировать Selenium в качестве фреймворка для Unit-тестирования.**Средства сбора информации о деятельности приложения**

Несмотря на огромное количество разнообразных исследовательских методик и техник, общая схема мероприятий, реализуемых в рамках рыночных исследований, достаточно проста и понятна. Основными источниками получения маркетинговой информации являются:

· Интервью и опросы;

· Регистрация (наблюдение);

· Эксперимент;

· Панель;

· Экспертная оценка.

Интервью (опрос) - выяснение позиции людей или получение от них справки по какому-либо вопросу. Опрос - это наиболее распространенная и важнейшая форма сбора данных в маркетинге. Приблизительно 90% исследований используют этот метод. Опрос может быть устным (личным) или письменным.

Наблюдение (регистрация) представляет собой форму маркетинговых исследований, с помощью которых осуществляется систематическое, планомерное изучение поведения того или иного объекта или субъекта. Наблюдение, в отличие от опроса не зависит от готовности наблюдаемого объекта сообщать информацию. Наблюдение - это процесс открытого или скрытого от наблюдаемого сбора и регистрации событий или особых моментов, связанных с поведением изучаемого объекта. Предметом наблюдений могут быть свойства и поведение индивидуумов; перемещение вещей, товаров и т.п. Недостатком наблюдений является невозможность выявления мнений, представлений, знаний людей. Поэтому на практике наблюдения обычно используются совместно с другими методами исследований.

Эксперимент - это исследование влияния одного фактора на другой при одновременном контроле посторонних факторов. Эксперименты подразделяются на лабораторные, проходящие в искусственной обстановке (тест продукта), и полевые, протекающие в реальных условиях (тест рынка). Основными недостатками, данного метода являются значительная стоимость и длительность проведения, что существенно ограничивает применение этого метода в практических исследованиях.

Панель - это повторяющийся сбор данных у одной группы опрашиваемых через равные промежутки времени. Таким образом, панель - это вид непрерывной выборки. Она позволяет зафиксировать изменения наблюдаемых величин, характеристик. Панельный опрос используют при изучении мнений потребителей определенной группы за какой-либо промежуток времени, когда определяются их потребности, привычки, вкусы, рекламации. Недостатками использования панелей являются: "смертность" панели, проявляющаяся в постепенном отказе участников от сотрудничества или переходе в другую потребительскую категорию, и "эффект панели", заключающийся в сознательном или бессознательном изменении образа поведения участников, находящихся под длительным контролем.

Экспертная оценка - это оценка исследуемых процессов квалифицированными специалистами - экспертами. Подобная оценка особенно необходима, когда невозможно получит неопосредованную информацию о каком-либо процессе или явлении. На практике для проведения экспертных оценок чаще всего применяют дельфи-метод, метод мозговой атаки и метод синектик.

**Виды и источники угроз безопасности веб-ориентированных приложений**

Основные типы угроз информационной безопасности веб-приложения:

Угрозы конфиденциальности – несанкционированный доступ к данным.

Угрозы целостности – несанкционированное искажение или уничтожение данных.

Угрозы доступности – ограничение или блокирование доступа к данным.

Угрозы безопасности связаны с несколькими факторами: в первую очередь это уязвимости веб-приложений или их компонентов. Во вторую — с используемыми механизмами проверки идентификации. В третью очередь угрозы безопасности относятся к атакам на самих пользователей, клиент-сайд атаки. Четвертый вид угроз — утечка или разглашение критичной информации. Пятый вид угроз — логические атаки.

Виды атак на веб-приложения Целевые атаки — это атаки, специально нацеленные на один сайт или их группу, объединенную одним признаком (сайты одной компании, либо сайты, относящиеся к определённой сфере деятельности, либо объединенные рядом признаков). Опасность таких атак заключается именно в «заказном» характере. Исполнителями таких атак становятся, как правило, злоумышленники, обладающие высокой квалификацией в области безопасности веб-приложений. Целью таких атак обычно является получение конфиденциальной информации, которая может быть использована недобросовестными конкурентами или преступниками для получения прибыли. Нецелевые атаки — это атаки, которые проводится фактически “на удачу”, а ее жертвами становятся случайные веб-сайты независимо от популярности, размера бизнеса, географии или отрасли. Нецелевая атака на сайт – это попытка получения несанкционированного доступа к веб-ресурсу, при которой злоумышленник не ставит целью взломать конкретный сайт, а атакует сразу сотни или тысячи ресурсов, отобранных по какому-то критерию. Например, сайты, работающие на определенной версии системы управления сайтом. Такого рода атаки бьют по «площадям», стараясь охватить максимальное количество сайтов при минимуме затрат. При удачной попытке атаке злоумышленник старается извлечь из этого пользу: закрепиться на сайте, загрузив хакерский скрипт (бэкдор, веб-шелл), добавить еще одного администратора, внедрить вредоносный код или получить необходимую информацию из базы данных. Целевые атаки — проводятся скрытно, как правило достигают своей цели. Нецелевые атаки довольно «шумные» и зачастую не достигают поставленных целей, но, тем не менее, могут доставить множество проблем владельцу веб-ресурса.

**Универсальные подходы к обеспечению и тестированию безопасности веб-ориентированных приложений**

Хеширование паролей, защита от sql инъекций и скриптов.

**Планирование тестов безопасности**

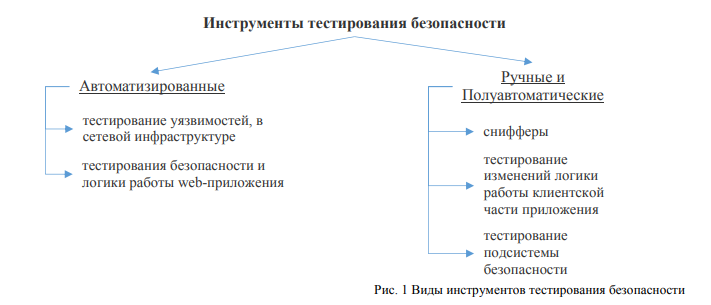
Планирование тестирования включает действия, направленные на определение основных целей тестирования и задач, выполнение которых необходимо для достижения этих целей.

**Тест-политика** – высокоуровневый документ, описывающий принципы, подходы и основные цели компании в сфере тестирования.

**Тест-стратегия** – высокоуровневый документ, содержащий описание уровней тестирования и подходов к тестированию в пределах этих уровней. Действует на уровне компании или программы (одного или больше проектов).

**Тест-план**– документ, описывающий средства, подходы, график работ и ресурсы, необходимые для проведения тестирования. Помимо прочего, определяет инструменты тестирования, функциональность, которую требуется протестировать, распределение ролей в команде, тестовое окружение, используемые техники тест-дизайна, критерии начала и окончания тестирования и риски. То есть, это подробное описание всего процесса тестирования.  
**Использование готовых инструментальных средств тестирования безопасности**

Тестирование безопасности — это процесс, который осуществляется с целью выявления недостатков в механизмах безопасности и нахождения уязвимостей в программных приложениях. Главная цель тестирования безопасности – определить уязвимости ИС и уровень защиты ресурсов от потенциальных злоумышленников.



**Противостояние социальной инженерии**

**Принципы юзабилити веб-ориентированных приложений**

Научно юзабилити (удобство использования) трактуется как

* мера пользовательской реакции
* мера пользовательского опыта

Говоря простым языком, это:

* насколько пользователю нравится приложение
* насколько быстро пользователь обучается работе с приложением

Применительно к веб-сайтам юзабилити - это удобство пользования сайтом, достигаемое применением концепции построения веб-интерфейсов сайта, направленной на достижение основной цели (или целей), поставленной перед сайтом

Примеры целей

* звонок менеджеру
* заказ
* покупка товара
* сообщение о сайте знакомым

Правило 7+/-2

Психологи утверждают, что большинство людей может одновременно удерживать во внимании примерно семь (плюс-минус два) объектов

Таким образом перегруженные меню или каталоги с сотнями позиций являются НЕудобными

Старайтесь акцентировать внимание на нескольких первостепенных объектах на странице, в меню и т. п.

Правило одной секунды

Ещё 2-3 года назад это правило называлось “правилом двух секунд”, но сейчас время изменилось

Если пользователь ждёт реакции на свои действия более секунды - он это замечает и начинает раздражаться

Если пользователь ждёт больше двух секунд - он злится

Ожидание более пяти секунд большинством пользователей воспринимается как “подвисание” приложения

Правило трёх клиентов

Пользователь не хочет и не будет продираться сквозь непонятные бессмысленные многоуровневые меню, ведущие его в неизвестность

Пользователь хочет, чтобы навигация была простой, как кирпич

Число “три” в названии правило выбрано как “наиболее привлекательное”, и не является догмой, однако если для выполнения простой задачи пользователю приходится делать 5-7 и более действий - это плохо. Что-то нужно менять

Правило пирамиды

Суть этого правила состоит в том, чтобы помещать в самом начале информационного блока итоговый вывод, а затем расширять тему

Это позволяет пользователю максимально быстро получать информацию, оценивать её и либо погружаться в более подробное рассмотрение, либо переходить к следующему информационному блоку.

Принцип удовлетворённости

Пользователю далеко не всегда нужно самое лучшее, их больше интересует “то, что их устроит”

Иными словами, если пользователь быстро нашёл удовлетворяющее его решение - он останется доволен

Даже если при этом существует решение, которое лучше найденного, но требует более сложной схемы достижения результата

Синдром утёнка

Огромное значение для пользователя имеет его первый опыт работы с некоторым классом приложений

Впоследствии всё новое пользователь сравнивает с тем первым. Сравнение, как правило, оказывается в пользу первого опыта

Это правило следует учитывать при редизайне сайта (можно потерять старых клиентов) и при разработке некоего совершенно оригинального дизайна приложения для которого уже сложились традиции

Баннерная слепота

Опытные пользователи инстинктивно игнорируют всё, что так или иначе напоминает банеры или иную рекламу

Поэтому если вы решили разместить что-то полезное, но оформить это “в рекламном стиле”, оно многими останется незамеченным

Принципы восприятия форм

Закон близости утверждает, что когда мы видим набор объектов, объекты расположенные ближе друг к другу, мы распознаём как группу.

Закон сходства утверждает, что сходные объекты человек подсознательно группирует

Закон содержательности утверждает, что один и тот же объект может играть важную роль в одном визуальном поле и быть частью фона в другом

Закон симметрии утверждает, что мы склонны воспринимать симметричные объекты как один объект

Закон смыкания утверждает, что люди склонные объединять объекты, которые на самом деле едиными не являются

Правила построения интерфейса

1. Интерфейс должен быть логичным не вводящим в заблуждение
2. Для опытных пользователей должен существовать быстрый способ выполнения операций (горячие клавиши, макросы)
3. Обработка ошибок должна быть простой и информативной
4. Всегда должен существовать простой способ отмены действия
5. Пользователь должен чувствовать, что всё под его контролем
6. Как можно меньше загружайте кратковременную память пользователя

Поскольку многие пользователи сразу открывают несколько сайтов в результатах поиска и бегло просматривают страницы, лучше размещать полезную с точки зрения сайта информацию так, чтобы пользователю не приходилось прокручивать страницу вниз

Качество дизайна и актуальность информации являются показателями доверия

Пользователи концентрируют своё внимание на лицах людей. Поэтому дизайн, в котором присутствуют фотографии, выигрывает у дизайна, в котором фотографий нет.

При этом если “взгляд” людей на фотографии “направлен на некоторую часть страницы” пользователи сами склонны посмотреть туда. Так можно привлечь внимание к важным элементам страницы

Свободное пространство улучшает восприятие текста

Также следует помнить, что люди охотнее (и быстрее!) читают текст в несколько узких колонок, нежели со строками, длина которых почти равна ширине страницы.

**Способы тестирования юзабилити веб-ориентированных приложений**

**Карточная сортировка (card sorting)**

Это **классификационный метод,** при котором пользователи сортируют различные элементы по нескольким категориям

Для проведения карточной сортировки создаётся список параметров каждый из которых выписывается на отдельной карточке

**Карточки показывают пользователям и просят сгруппировать наиболее логичным образом**

Полученную в результате информацию используют для организации пользовательского интерфейса

Так, например, можно сгруппировать пункты подменю по пунктам меню, информационные блоки по блокам страницы, страницы по разделам и т.п.

**Оценочные листы (marklists)**

Оценочные листы помогают **удостовериться в том, что продукт выполнен с учётом принципов юзабилити**

При создании оценочного листа следует помнить, что это - **НЕ анкета.** Т.е. **вопросы в оценочном листе должны быть предельно короткими, ясными, сгруппированными по категориям и допускать только однозначные ответы вида “ДА” или “НЕТ”**

Иногда в оценочных листах допускаются “ответы с ранжированием”, но куда чаще это делается в классических анкетах

**Анкетирование (questioning)**

В то время как оценочные листы применяются на поздних стадиях для контроля качества, **анкетирование даёт наибольшую отдачу на стадии прототипирования или сбора требований**

Анкетирование позволяет **собрать много сведений** о предпочтениях пользователей

Главное требование к анкетам - их верное составление. **Вопросы должны быть понятны пользователям, а варианты ответов должны позволять автоматизированную обработку данных**

**Прототипирование (prototyping)**

Прототипирование в контексте юзабилити в основном сводится к **построению макетов (прототипов) дизайна интерфейса**

Прототип демонстрируется заказчику и/или целевой аудитории, после чего на него собираются отзывы. **Наибольший эффект даёт сбор отзывов в свободном виде дополненный оценочным листом.**

**Протоколы самоотчёта (self-reporting logs)**

Данный способ является **одним из наименее затратных,** т.к. участие специалистов требуется только на стадии обработки данных, однако качество собранной информации невелико, т.к. **фактически пользователям приходится проводить исследовательское тестирование юзабилити, после чего писать баг-репорты.**

Мало кто из конечных пользователей на такое способен

**Фиксация “мыслей вслух” (thinking aloud protocol)**

Это очень популярный способ тестирования юзабилити, при котором **пользователя просят проговаривать всё, что он делает**

Иногда для усложнения эксперимента задачу просят выполнить **двух пользователей совместно, при чём у одного есть доступ только к клавиатуре, а у другого к мыши**

Данный способ позволяет не только отследить эмоциональные реакции пользователей, но и **почерпнуть из их поведения ценные мысли**

**Фокусные группы (focus groups)**

Данный метод заключается в подробном опросе специально отобранной группы пользователей, представляющих собой фрагмент целевой аудитории приложения

Это довольно затратный способ, но он позволяет не только отследить реакцию отдельных пользователей, но и **изменение реакции в процессе обмена мнениями**

Не следует делать фокусные группы чрезмерно большими. **Достаточно 6-9 человек.**

**Основные понятия автоматизированного тестирования**

Автоматизированное тестирование программного обеспечения (Software Automation Testing) - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Специалист по автоматизированному тестированию программного обеспечения (Software Automation Tester) - это технический специалист, обеспечивающий создание, отладку и поддержку работоспособного состояния тест скриптов, тестовых наборов и инструментов для автоматизированного тестирования.

Инструмент для автоматизированного тестирования (Automation Test Tool) - это программное обеспечение, посредством которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тест скриптов.

Тест Скрипт (Test Script) - это набор инструкций, для автоматической проверки определенной части программного обеспечения.

Тестовый набор (Test Suite) - это комбинация тест скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

Тесты для запуска (Test Run) - это комбинация тест скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования).

**Семейство инструментальных средств Selenium**

Selenium – это инструмент для тестирования Web-приложений. Selenium – объектно-ориентированное JavaScript приложение для анализа файлов определенной структуры находить в них команды для манипуляции браузером и команды для выполнения определенных действий и проверок. Selenium поддерживается Microsoft Internet Explorer, Google Chrome, Mozilla Suite и Mozilla Firefox для Microsoft Windows, Linux, и Apple Macintosh.

Selenium IDE предоставляет утилиту для записи и последующего воспроизведения автотестов. Тестовые скрипты могут быть автоматически записаны и изменены вручную. Selenium Remote Control (RC) – сервер, написанный на Java, который воспринимает команды для браузера через HTTP. RC делает возможным написание тестов для Web-приложений на любом удобном языке программирования, что позволяет интегрировать Selenium в качестве фреймворка для Unit-тестирования.

**Модульное тестирование Junit**

JUnit – это Java фреймворк для тестирования, т. е. тестирования отдельных участков кода, например, методов или классов. Опыт, полученный при работе с JUnit, важен в разработке концепций тестирования программного обеспечения.

JUnit позволяет в любой момент быстро убедиться в работоспособности кода. Если программа не является совсем простой и включает множество классов и методов, то для её проверки может потребоваться значительное время. Естественно, что данный процесс лучше автоматизировать. Использование JUnit позволяет проверить код программы без значительных усилий и не занимает много времени.

**N–версионное программирование. Отказоустойчивость компьютерных систем**

1. технологии которые минимизируют кол-во ошибок ПО
2. Использование методов верификации
3. Использование метод тестирования и отладки

Отказоустойчивость ПО можно повысить за счёт введения различных форм избыточности.

**N-версионное тестирование** - введение избыточности, позволяющее повысить отказоустойчивость ПО.

**N-версионное программирование** - независимое создание 2 и более функционально независимых эквивалентных программ на основе общего исходного описания.

**3 требования к N-версионное программированию**

1. Для выполнения одной и той же задачи необходимо сделать N независимых программ. Желательно с разными алгоритмами, ЯП и средствами разработки.
2. Спецификации ПО предусматривается дополнительно
   1. разработать алгоритм сравнения по совпадению при n = 2, или по мажоритарному принципу сравнения при n > 2
   2. необходимо предусмотреть точки контроля
   3. для оценки правильности результатов в алгоритме сравнения устанавливается допустимый диапазон расхождения числовых результатов
3. Н независимых разработанных программ должны работать параллельно на н независимых компьютерах (результаты вычисления на каждом шаге сравниваются между собой)

Чем больше количество элементов, тем выше достоверность получаемых результатов.

Отказоустойчивость компьютерных систем

**Отказоустойчивость** - это свойство архитектуры компьютерных систем, позволяющих пользователю или программе продолжить работу и тогда, когда в аппаратных или программных средствах возникли отказы.

По способу реализации отказоустойчивость делится на:

* активная
* пассивная

Отказоустойчивость активная базируется на процессах:

1. обнаружение отказа
2. локализация отказа
3. реконфигурация систем

**Активная отказоустойчивость** локализована в многопроцессорных системах с общей памятью, общей шиной, кольцевой или иерархической структурой.

**Пассивная отказоустойчивость** заключается в свойстве системы не потерять функциональные свойства в случае отказа отдельных элементов системы.

Пассивная отказоустойчивость применяется:

1. на особо ответственных компьютерных системах
2. Системы, в работе которых недопустим перерыв в работе
3. Обеспечение отказоустойчивости важнейших подсистем компьютерной системы

**Отказоустойчивость системы может быть обеспечена**:

1. временной избыточностью
2. алгоритмической избыточностью
3. структурной избыточностью

**Временная избыточность** заключается в наличии дополнительного времени для решения задачи.

**Алгоритмическая избыточность** заключается в применении таких алгоритмов которые обеспечивают удовлетворительные результаты в случае наличия или возникновения ошибок в процессе вычислений. При возникновении ошибок увеличивается время необходимое для реализации алгоритма.

**Структурная избыточность** выражается в наличии дополнительных подсистем в структуре компьютерной системы, предназначенных для автоматической замены отказавших элементов.