1. Надежность, свойства надежности.

Надёжность — свойство объекта сохранять работоспособное состояние в течение некоторого времени. Оттсутствие непредвиденных изменений качества на стадии эксплуатации.

1. **Безотказность** – свойство объекта **непрерывно сохранять работоспособность** в течение некоторого времени или наработки.
2. **Долговечность** – свойство изделия **сохранять работоспособность до наступления предельного состояния**, при котором его дальнейшее использование неэффективно, а восстановить его невозможно или слишком дорого.
3. **Завершенность** (бездефектность **готового** программного средства)
4. **Ремонтопригодность**
5. **Сохраняемость** – свойство изделия **сохранять безотказность, долговечность и ремонтопригодность** во время и после хранения и транспортировки.

*Наработка* **-** продолжительность или объем работы объекта, выражается либо через время, либо через количество циклов работы, либо через количество решенных задач.

*Отказ*- событие, заключающееся в нарушении работоспособного состояния объекта.

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

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

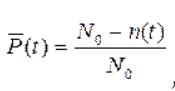
*Устойчивость*- свойство объекта сохранять работоспособность **в условиях неблагоприятных воздействий**, не предусмотренных нормальными условиями функционирования.

2. Показатели надежности невосстанавливаемых объектов

Невостанавливаемый объект работает до превого отказа.

1. вероятность безотказной работы P(t) - вероятность того, что в течении заданной наработки t невосстоналиваемый объект не даст отказа. (T — время исправной работы объекта от начала работы до первого отказа, а t — время, за которое необходимо определить свойства величины)
   1. P(t) = P{T >= t})
   2. P(0) = 1
   3. limP(t) = 0 - объекта не может сохранять работоспособность неограниченно долго

Вероятность безотказной работы по статистическим данным об отказах оценивается выражением



N0 – число объектов в начале испытания;

n(t) – число отказавших объектов за время t

2) Вероятностью отказа Q(t) - заданном интервале времени возникнет хотя бы один отказ.

Q(t) = 1 – P(t) = n(t) / N0

3) плотность распределения наработки на отказ

4**) Средняя наработка на отказ**

5) Интенсивность отказаов

3. Показатели надежности восстанавливаемых объектов.

1. **Поток отказов** - отношение числа отказавших объектов в единицу времени к числу испытываемых объектов при условии, что все вышедшие из строя изделия заменяются исправными
2. **Наработкой на отказ** называется среднее значение времени между соседними отказами.
3. **Коэффициентом простоя** - отношение времени восстановления к сумме времен наработки на отказ и времени восстановления. Кп= tв/( tср.+ tв),
4. **Коэффициент готовности** Кг используется в качестве показателя надежности, если кроме факта отказа необходимо учитывать время восстановления. Кг= tср./( tср.+ tв),
5. ***среднее время восстановления***

4. Основные причины появления ошибок в ПО.

1. Человеческий фактор. Бизнес-требования могут быть не донесены до всех участников процесса разработки или донесены в искаженном, неполном или противоречивом виде
2. временные ограничения
3. финансовые ограничения
4. баги в инструментах для разработки
5. также невозможность проведения исчерпывающего тестирования.

5. Способы обеспечения надежности ПО.

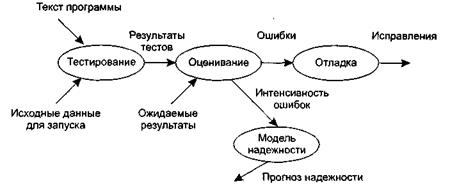
1. создавать программные **модули высокого качества**;
2. **предотвращать дефекты проектирования за счет эффективных технологий** и **средств автоматизации** обеспечения всего жиз­ненного цикла
3. обнаруживать и устранять ошибки **путем систематического тестирования** **на всех этапах жизненного цикла** ПС;
4. оперативно **выявлять последствия дефектов** программ и данных **и восстанавливать нормальное, функционирование**

6. Информационные потоки тестирования.

Тестирование ПО – **процесс поиска ошибок** в программном продукте.

На входе в процесс имеются потоки:

1. текст программы (выполняемый файл или запрос для web);
2. исходные данные для текста;
3. Ожидаемые результаты.



7. Тестовый план, тестовая стратегия.

План тестирования - документ, который определяет **объем, цель, подход, приоритизации задач, учет программного обеспечения и человеческих ресурсов**. Определяется на уровне проекта. Ведущий тестировщик утверждает план.

* Что **надо** тестировать?
* Что **будете** тестировать?
* Как будете тестировать?стратегия тестирования, а именно: виды тестирования
* Когда будете тестировать?
* Критерии начала тестирования:

1. готовность тестовой платформы
2. законченность разработки
3. наличие всей необходимой документации

* Критерии окончания тестирования:

1. результаты тестирования удовлетворяют критериям качества продукта:

требования к количеству открытых багов выполнены

Стратегия тестирования — это набор руководящих принципов, которые объясняют дизайн теста. Он устанавливается на уровне организации и может использоваться несколькими проектами.  
План тестирования может быть изменен, тогда как стратегия тестирования не может быть изменена.

8. Процесс разработки тестовых случаев.

Test-case —документ с **описанием шагов**, для проверки **одной части функционала**, с заданными **критериями прохождения**. Action > Expected Result > Test Result.

1. **Ознакомление с проектной документацией**: требованиями, тестовым планом и **непроектной** информацией (литературой по предметной области).

2. **Проектирование тестовых случаев**. Разбиение на модули. Один тестовый случай должен проверять только одну функциональность и иметь определенный ожидаемый результат. Формирование тестовых наборов связано с градацией тестов, из которой наиболее распространенной является разбивка на три блока – smoke, positive, negative.

3. **Написание тестовых случаев.**

4. **Проверка тестовых случаев**

– соответствует ли тестовый случай функциональности, заявленной в требованиях;

– покрывают ли тестовые случаи все требования к программному продукту

– имеются ли среди тестов избыточные, можно ли устранить эту

избыточность;

– снабжен ли каждый тестовый случай ожидаемыми результатами;

– достаточно ли подробно разработана методика тестирования, чтобы стала возможной ее автоматизация.

1. PreConditions - Список действий, которые приводят систему к состоянию пригодному для проведения основной проверки.
2. Test Case Description - Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям.
3. PostConditions – почистить базу на пример. (Не обязательно)

9. Свойства тестовых случаев.

a) высокая **вероятность обнаружения дефекта**

b) **воспроизводимый**;

c) четко определенные ожидаемыме результаты и критерии успешного выполнения теста;

d) **набор тестовых случаев не должен быть избыточным**.

10. Эквивалентирование и анализ граничных значений.

Класс эквивалентности– набор данных, обработка которых приводит к одному и тому же результату.

**Весь диапазон возможных входных значений**

Как минимум существует два класса эквивалентности: допустимые и запрещенные значения.  
Пример: снять деньги в банкомате – 100 –ок. 1000-смс, 100000 – отказ.

Очень важно проверять именно граничные значения, потому что довольно часто возникают ошибки именно на границах классов эквивалентности.

На каждой границе диапазона нужно проверить 3 значения: граничное значение, значения до и после границы.

11. Градация тестов.

**По выполнению кода**

* Статическое
* Динамическое

**По уровню тестирования**

* Модельное
* Интергационное
* Системное

**По исполнителям**

* Альфа
* Бетта

**По знаню систиемы**

* Белый ящик
* Серый
* Черный

**По позитивности**

* Позитивное
* Негативное
* Smoke test

**По степени автоматизации**

* Ручное
* Полуафтоматизированное
* Автоматизированное

По целям

* Функциональное
* Не функциональное(тестирование интерфейса, удобства, защищенности, надежности и восстановления после сбоев, производительности (назрузочное, масштабируемости...))

12. Ошибка. Свойства ошибки.

Ошибка - это расхождение между программой и ее спецификацией.

**1 - Важность**

* Критическая - дальнейшая работа с программой невозможна;
* серьезная - не работает основная функциональность (добавление записи в базу данных)
* средняя - не работает второстепенная функциональность либо имеются недочеты в работе основной функциональности (не работает сортировка товара, или не появляется сообщение для подтверждения удаления товара из базы данных)
* низкая - мелкая ошибка (ошибка интерфейса либо орфографическая ошибка)

**2 - Воспроизводимость**

* всегда - воспроизводится на всех устройствах и не зависит ни от каких условий
* иногда - при определенных условиях

**3 - Приоритет** (скорость исправления ошибки)

* очень высокий - ошибка должна быть исправлена немедленно
* высокий, когда ошибка должна быть исправлена как можно скорее
* средний, когда ошибка может быть исправлена, когда появится свободное время
* низкий, когда ошибка может быть исправлена в последнюю очередь

**4 - Симптом** (пункт 13)

13. Симптомы ошибок.

1. неожиданное поведение - вместо сообщения о сохранении данных, появляется сообщение об их успешном удалении
2. недружественное поведение - программа не сопровождает свои действия сообщениями
3. неверное действие - программа не выполняет требуемое действие или выполняет неверно ( не удаляет запись из бд)
4. отказ системы - дальнейшая работа с программой невозможна
5. потеря данных - добавленные данные исчезли
6. искажение данных - произошла замена данных программы искаженными, или данные выводятся на экран в другой кодировке
7. низкая производительность
8. локализационная ошибка
9. инсталляционная ошибка, которая проявляется во время разных способов инсталляции ПО
10. косметическая ошибка
11. ошибка документации связана с недочетами в документации
12. различия со спецификацией - названия элементов интерфейса, их количество, цвет и т. д. не соответствуют заявленным в требованиях
13. отсутствующая функциональность
14. **запрос на улучшение является симптомом**, который может быть выставлен, когда тестировщик предлагает улучшить некоторую функциональность программы либо ее внешний вид. Таким образом, ошибка с этим симптомом может не противоречить требованиям, и, соответственно, может не быть исправлена.

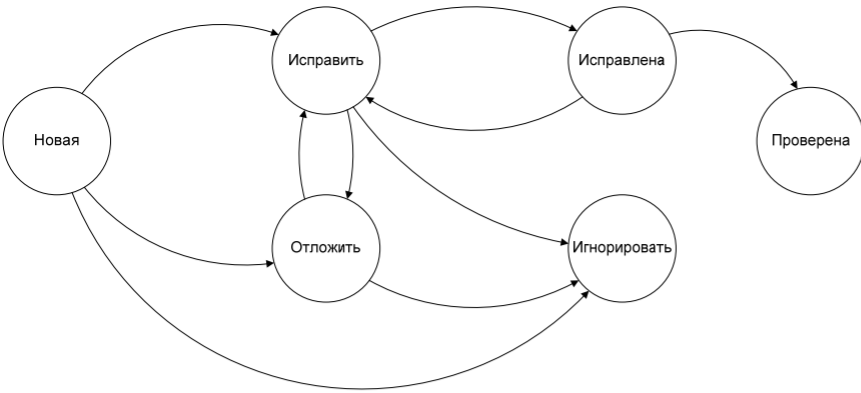
14. Правила составления отчетов об ошибках.

1. сразу
2. Не на бумаге
3. описание
4. шаги вопроизведения
5. название
6. Указать важность, симптом, воспроизводимость, приоритет.
7. простым языком
8. Ни кого не обвинять

* id ошибки
* статус (новая)
* Кем обнаружен
* Дата
* Краткое описание
* Развернутое описание
* Как воспроизвести

15. Жизненный цикл ошибки.

Жизненный цикл ошибки – это путь, который проходит ошибка с момента ее обнаружения до закрытия либо терминального статуса.



Возможны и другие статусы ошибок - дубликат...

16. Системы документирования ошибок.

Bug Tracking Systems, BTS не ограничиваются работой с ошибками, а представляют собой СУП. Назначение:

1) повышать взаимодействие между сотрудниками;

2) ни одна ошибка не должна быть не исправлена, потому что так решил разработчик;

3) как можно меньше ошибок должно остаться из-за проблем взаимодействия сотрудников.

BTS классы:

– свободно распространяемые (Bugzilla, Mantis, Redmine, Track);

– платные системы (Jira, ClearQuest, TestTrackPro);

– собственная разработка

критерии выбора BTS:

* стоимость и схема лицензирования;
* гибкость настройки системы;

17. Приемочный тест, критерии его непрохождения.

Smoke test – быстрая проверка основной функциональности. (название от печников, и радио инженеров)

После приемочного теста делается вывод о пригодности программы для дальнейшего тестирования. Каждая версия разрабатываемого ПО подвергается проверке «на дым», поскольку нет смысла тратить усилия тестировщиков на версию, в которой не работает даже основная функциональность.

Критерии непрохождения должны быть **в тестовом плане**:

1) отсутствие каких-либо файлов или компонент, без которых любое тестирование ПО невозможно;

2) сбой программного обеспечения или системы в начале работы

3) ошибка в программе, приводящая к сбою в середине работы и делающая дальнейшее тестирование невозможным;

4) достижение определенного процента ошибок, недопустимого для приемочного тестирования, например, 20–25 % тестовых случаев не проходит

18. Критическое тестирование на примере.

Тест критического пути — основной тип тестов. Может быть позитивным и негативным. Пусть тест критического пути – это suite состоящий из нескольких тест кейсов:

1. логин, проверка
2. создание поста, проверка
3. удаление поста, проверка
4. выход, проверка

19. Углубленное тестирование на примере.

Негативный тест часто объединяют с **мелкими проверками на нестандартное использование** (разный формат **даты** или перемещение через **табуляцию**, границы переполнения **массивов** данных, ввод **специальных** символов ), тогда он получает название «углубленный тест» (или расширенный Extended). Такой тест проводят **на стабильно работающей програ**мме ближе к окончанию разработки, что связано с экономией времени, поскольку **нет смысла проверять нестандартную работу программы, если не работают стандартные ситуаци**и.

20. Использование контрольных перечней в углубленном тестировании.

Чек-листы – это мини-тестовые случаи, которые создаются для стандартных элементов и полей. Применяемых практически в любых проектах(числовых, полей даты, времени...). Поэтому нет смысла для каждого отдельного проекта разрабатывать свои тестовые случаи для таких полей , а достаточно использовать контрольные перечни, **выработанные опытным путем** в течение времени.

Проверка числовых полей:

– проверить на минимально и максимально допустимое число;

– проверить на отрицательные значения;

– проверить на буквенные символы;

– проверить на 0, 1, –1.

21. Специфика тестирования веб-приложений.

Все приложения принято делить на две группы: веб и standalone.

* время **отклика** страниц
* зависимость от браузера, устойчивость к инъекциям, модификации get и post запросов, изменение информации в URL, прямые запросы в базу, проверка на уязвимость, проверка разграничений прав доступа
* тестирование **для мобильных устройств**. Должны быть специальные версии для этих устройств или проверка на разрешение экрана
* **поиск поврежденных ссылок и отсутствующих страниц.**

22. Методология автоматизации функционального тестирования.

Автоматизация тестирования – это процесс замены ручного тестирования некоторым **инструментальным** средством.

Этапы автоматизации

1) **принятие решения об автоматизации** тестирования;

2) **выбор инструментальных средств** тестирования;

3) **планирование и проектирование** тестирования;

4) **выполнение и управление** тестированием;

5) **оценка результатов** тестирования.

Популярность тестирования – GUI объясняется двумя факторами: во-первых, приложение тестируется тем же способом, которым его будет использовать человек, во-вторых, можно тестировать приложение, не имея при этом доступа к коду. Общим свойством всех инструментов является получение тестового автоматизированного скрипта. Следует применять только на стабильно работающем программном продукте

23. Достоинства и недостатки автоматизации функционального тестирования.

Достоинства

1. Автоматизированные тесты могут прогоняться **множество раз**
2. **скорость**.
3. Протекают каждый раз **одинаково**.
4. Возможна автоматизация однообразных и однотипных задач.
5. **ночью и в выходные дни**.
6. Автоматизированные испытания **генерируют отчеты**
7. Уменьшаются временные затраты на формирование статистических данных о проекте.
8. **Исключение** ошибок, связанных с человеческим фактором.
9. Взаимодействие с бд напрямую.
10. **Освобождение человека от рутинной** работы с возможностью заняться более интересными и креативными задачами.

Недостатки

1. долго (15)
2. Дорого
3. Стоимость инструмента автоматизации
4. Человек в отличии о автотеста может заметить побочные ошибки

24. Необоснованные ожидания от автоматизированного тестирования.

1. Автоматизировать можно **все что угодно** (удобство использывания нельзя)
2. Можно обнаружить **бόльшее количество ошибок** (автотест идет только по сценарию, а человек видит побочные ошибки)
3. Можно исключить или **значительно сократить ручное** тестирование (1)
4. Возможно **100 % покрытие** функциональности (1)
5. Все необходимое тестирование может выполнять **одно инструментальное средство**
6. **Быстро** (разработать в 15 раз дольше).
7. **Автоматизация недорога**
8. **Средства автоматизации просты** в использовании (нет)

25. Выбор тестов для автоматизации.

Наиболее важный фактор при выборе тест кейсов для автоматизации – ROI = прибыль / затраты. Если > 1 мы в плюсе

ROIавтоматизации = (X – Y) / Y

1. Определить частоту выполнеия тест кейса
2. Убедится что анализ результатов автотеста не будет превышать время, затрачиваеое при ручном тестировании.
3. Основные критические пути
4. Тесты, которые должны прогоняться при каждом билде
5. Data driven
6. Тесты которые работают долго(ночью)
7. Производительность

Исключить

1. Тетсы юзабилити
2. Те, которые не нужно запускать повторно

26. Требования, предъявляемые к автоматизированным тестам.

1) Тест должен не только эмулировать действия пользователя, но **и проводить проверки.**

2) Тест должен выдавать **результат в таком виде**, чтобы сразу можно было **понять, где ошибка**.

3) **Один тест** должен проверять только **одну функциональность**

4) Каждый тест должен быть **независимым**

5) Тест должен запускаться **на разных платформах и в разных браузерах.**

6) Наличие **документированных требований** к запуску теста.

7) Тест должен **одинаково хорошо работать** на быстрой машине и медленной.

27. Метод «Play&Record» в автоматизированном тестировании.

Записанные действия преобразуются в программный код на языке программирования, который встроен в инструмент автоматизации.

1. Из инструмента автоматизации происходит захват тестируемого приложения.

2. Включается запись действий.

3. Выполняются требуемые действия над программой.

4. Останавливается запись.

5. Включается проигрывание записанных действий.

Недостатки

* Скрипты, получаемые этим методом, содержат фиксированные значения, которые должны быть изменены каждый раз, когда в приложении что-то меняется.
* Приходится часто переписывать = дорго
* Полученные скрипты не надежны и часто не работают, даже если тестируемое приложение не изменялось. (Всплывающие окна)
* Если тестировщик совершает ошибку, например, во время ввода данных, тест должен быть записан заново.

28. Метод «Data-driven» в автоматизированном тестировании.

DDT - данные для тестов выносятся за пределы кода скрипта - в Exсelтаблицу. Например, для проверки авторизации может быть создана таблица, которая в каждой строке хранит имя пользователя, пароль и ожидаемый результат.

29. Метод «Keyword-driven» в автоматизированном тестировании.

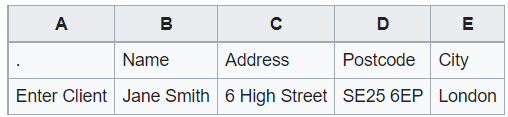
Тестовый сценарий представлен в виде таблицы, содержащей в одной из колонок ключевые слова, а в остальных колонках могут быть тестовые данные, ожидаемые результаты. Ключевое слово является, как правило, функцией некоторой библиотеки и реализует предписанную последовательность действий. Управляющий скрипт обращается к таблице тестового сценария, находит ключевое слово в нем, считывает из той же строки тестовые данные и выполняет действие, описанное этим ключевым словом. Полученный результат далее сравнивается с ожидаемым, и генерируется отчет о прохождении теста.

Таким образом, реализуется трехслойная модель автоматизации:

– 1-й слой – библиотека ключевых слов;

– 2-й слой – тестовый сценарий в виде электронной таблицы;

– 3-й слой – управляющий скрипт.



30. Возможности Selenium IDE и WebDriver.

Selenium IDE – рекордер

Selenium WebDriver - библиотека, позволяющая разрабатывать программы, управляющие поведением браузера. Для группировки и запуска тестов, а также для генерации отчетов о тестировании используется JUint, TestNG.

Программный каркас для автоматизации тестирования с использованием Selenium WebDriver разделяется на несколько слоев

● Слой тестов

● Бизнес-слой.

● Слой для работы с элементами UI

● Слой вспомогательных библиотек.

31. Возможности системы TestComplete.

Позволяет автоматизировать тестирование Windows приложений, а так же веб-страниц. Используя TestComplete, можно выполнять **функциональное, unit, регрессионное, распределенное** и другие виды тестирования. Тесты можно не только **записывать, но и редактировать или создавать с нуля**. Для этого TestComplete включает **редактор keyword тестов** и встроенный **редактор скриптов**. Скриптовый редактор обладает всеми редакторскими "примочками": code completion, code templates, распахивание/скрытие кусков кода, закладки и т.д. Оба редактора поддерживают **отладку** тестовых процедур. TestComplete **способен определять тип элемента управления, с которым вы работаете (кнопка, выпадающий список, меню и т.п.).** Во время выполнения тестов, TestComplete формирует **отчет** о всех выполненных действиях над приложением, прохождении контрольных точек, обнаруженных ошибках а так же скриншоты.

32. Возможности тестового инструмента CodedUI.

Coded UI – это фреймворк для автоматизации (UI) Web и Windows Forms приложений в среде Visual Studio. Record&PlayBack. Гененрит скрипт. Создание ассертов во время записи. Можно писать скрипт самому. Делает скриншоты. Автоматизация построения "карты интерфейса" (свойства элементов управления, параметры их поиска и доступа к ним)

33. Математические модели оценки надежности ПО.

Аналитические

Динамические

Дискретные

Не прерывные

Статические

Эмперические

Математические модели надежности позволяют определить оценки показателей надежности ПО

* вероятность безотказной работы P(t),
* интенсивность отказов
* математическое ожидание времени наработки на отказ m
* число ошибок в ПО **до начала тестирования Е0**.

34. Модель Джелинского-Моранды.

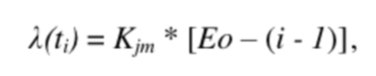
1. **В модели учитываются промежутки времени между отдельными отказами**
2. **Время до очередного отказа распределено по экспоненциальному закону.**
3. **На каждом этапе тестирования обнаруживается одна ошибка, которая устраняется до следующего этапа.**
4. **Частота появления ошибок пропорциональна числу не выявленных ошибок**

*Допущения модели:*

* Время до очередного отказа распределено по экспоненциальному

закону.

* Все ошибки равновероятны и их появление не зависит друг от друга.



где Ео — число ошибок в ПО до начала тестирования;

Кjm - коэффициент Джелинского-Моранды;

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

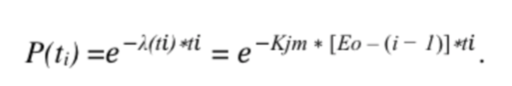
i — число ошибок, обнаруженных к моменту времени ti.

*  на интервале между 2-мя смежными моментами появления
* Каждая обнаруженная ошибка в ПО немедленно устраняется и число

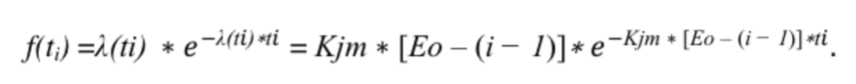
ошибок уменьшается на 1.

* Ошибки корректируются без внесения новых ошибок.

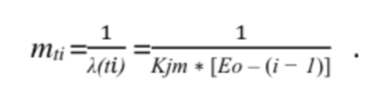
*Вероятность безотказной работы:*



*Плотность вероятности отказа:*



*Среднее время безотказной работы:*



**Таким образом, исходными данными являются интервалы времени между отказами. Характеристиками модели (что требуется определить) являются:**

1. **Р(ti) — вероятность безотказной работы;**
2. **mti — среднее время наработки на отказ (или математическое ожидание);**
3. **Kjm — коэффициент Джелинского-Моранды;**
4. **E0 — первоначальное количество ошибок в программе.**

Определить характеристики модели можно по методу максимального

правдоподобия.

35. Модель Шика-Волвертона.

**Модель Шика-Волвертона является модификацией модели Джелинского-Моранды для случая возникновения более одной оштбки на каждом промежутке времени.**

***Допущения модели:***

* **все допущения Джелинского-Моранды**
* **в основе модели лежит предположение, согласно которому частота появления ошибок пропорциональна не только количеству ошибок в программах, но и времени тестирования программы, т.е. вероятность обнаружения ошибок с течением времени должна возрастать**



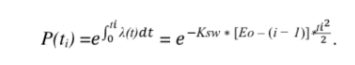
где Eo - число ошибок в ПО до начала тестирования и отладки

Ksw - коэффициент Шика-Волвертона

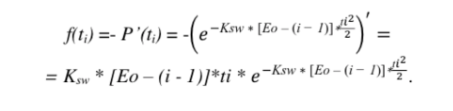
ti - интервал времени между обнаружением i-1 и i-й ошибкой

i - число ошибок, обнаруженных к моменту отладки

*Вероятность безотказной работы:*



*Плотность вероятность отказов:*

****

36. Геометрическая модель.

Является модификацией модели Джелинского-Моранды.

*Допущение модели:*

* Общее число ошибок в программе неограниченно
* Обнаружение ошибок не равновероятно
* что чем дольше длится процесс тестирования, тем труднее становится обнаруживать в нем ошибки
* таким образом, ПО никогда полностью не освобождается от ошибок.



D - исходное значение интенсивность отказа,

К - константа пропорциональности, 0<K<1

ti - время между появлениями i-1 и i-й ошибками

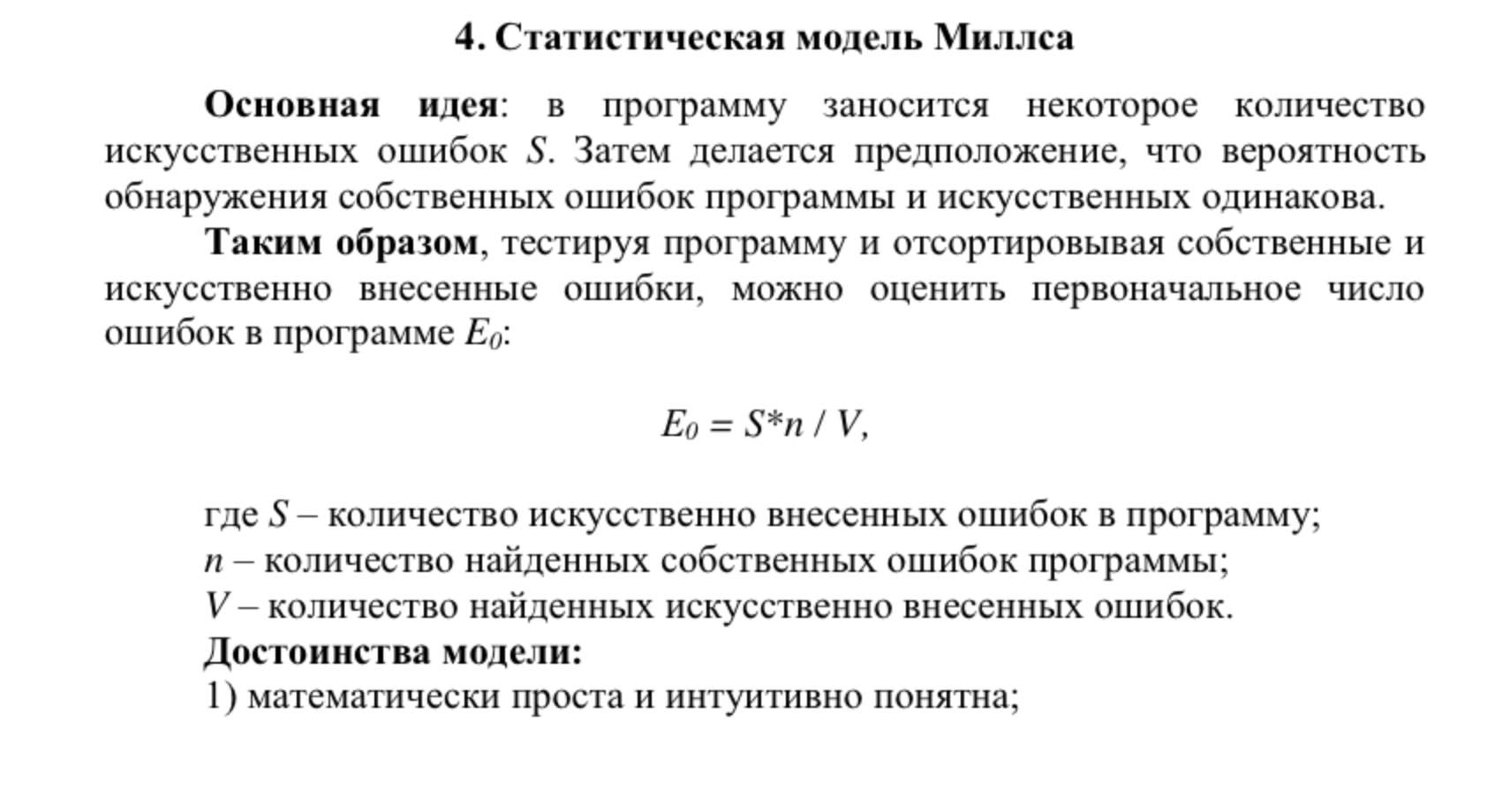
Функция риска убывает со скоростью геометрической прогресии

Как правило, интенсивность геометрической модели определяется методом максимального правдоподобия.

37. Статистическая модель Миллса.

Основная идея: в программу вносится некоторое количество искусственных ошибок S. Затем делается предположение, что вероятность обнаружения собственных ошибок программы и искусственных одинакова.

Таким образом, тестируя программу и отсортировав собственные и искусственно внесенные ошибки, можно оценить первоначальное число ошибок в программе Ео.

****

где S - количество искусственно внесенных ошибок в программу

n - количество найденных собственных ошибок программы

V - количество найденных искусственно внесенных ошибок

Достоинства

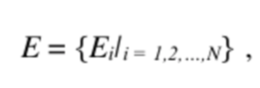
* математически проста и интуитивно понятна
* оказывает положительное психологическое влияние на тестировщиков, т.е. обнаружения не всех искусственно внесенных ошибок в программу мотивирует продолжение процесса тестирования

Недостаток

* процесс искусственного внесения ошибок, т.к. предполагается что собственные и внесенные ошибки появляются с равной вероятностью, т.е. искусственно вносимые ошибки должны быть типичны для этой программы

38. Модель Нельсона.

*Идея модели:* представление результатов работы программы в виде некоторой функции F(Еi), где Еi набор значений данных, необходимых для выполнения прогона программы:

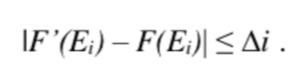


где Е — множество всех значений наборов входных данных;

N — количество возможных наборов входных данных Е.

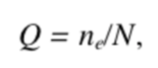
Выполнение (прогон) программы приводит к получению для каждого Ei определенного значения F(Ei). Наличие дефектов в программе приводит к тому, что получается некоторая величина F’, отличная от F.

Тогда для получения *приемлемого результата работы* программы:



Множество наборов, не удовлетворяющих данному условию, образуют подмножество Ee, ‚ т.е. отказы программы.

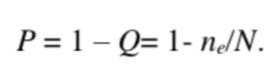
Тогда вероятность отказа:



где N — количество возможных наборов значений входных данных;

ne — количество различных наборов значений, содержащихся в Ee.

Тогда *вероятность удачного прогона* программы:



39. Сложность ПО.

Сложность программ - одна из главных причин их ненадежности.

Критерии оценки сложности ПО:

1) Длина программы, т.е. количество операторов

2) Цикломатическое число МакКейба

G = R - V + 2;

где R - количество ребер

V - количество вершин графа МакКейба

Если G>15 - программный модуль ненадежен, его необходимо разбить на части.

3) Мера сложности Холстеда

Сложность вызывается четырьмя основными причинами:

* сложностью реальной предметной области
* трудностью управления процессом разработки;
* необходимостью обеспечить достаточную гибкость программы;

40. Иерархическая модель оценки надежности ПО. (лаб5)

● 6 характеристик качества(функциональность, надежность, практичность, эффективность, сопровождаемость, мобильность) ФНЭС + МП

● Подхарактеристики (надежность – стабильность, восстанавливаемость)

● метрики качества

Метрика — это метод измерения и шкала измерения некоторого свойства программного средства.

1. **ГОСТ 28195-99.**
2. **СТБ ИСО/МЭК 9126-2003.**

41. ГОСТ 28195-99.

Регламентирует выполнение оценки качества ПС на основе иерархической модели качества. ГОСТ 28195-99 определяет четырёхуровневую иерархическую модель оценки качества ПС.

1. факторы качества (соответствуют характеристикам качества)
2. критерии качества (подхарактеристики качества)
3. метрики,
4. оценочные элементы.

Одним из факторов качества является надежность ПО.

* критерии для фактора надежность ПО: работоспособность.
* метрики для фактора надежность: средства восстановления при ошибках на входе
* Примеры оценочных элементов для фактора надежность: возможность обработки ошибочных ситуаций.

42. СТБ ИСО/МЭК 9126-2003.

Данный стандарт регламентирует методы оценки качества программных средств на основании *трехуровневой модели качества:*

*Описание процесса оценки состоит из трех стадий*

* определение требований к надежности
* подготовка к оцениванию
  + выбор метрик надежности
  + определение уровней ранжирования
  + определение критериев оценки надежности
* процесс оценивания
  + измерение (применение метрик к ПО)
  + ранжирование (установка ранжирования от шкалы)
  + оценка (заключение о надежности)

43. ISO/IEC 9126-1-4 2001-2004. а метрики измеряются атрибутами

*Надежность включает 4 подхарактеристики:*

* *завершенность* — стабильность программного продукта избегать отказов вследствие ошибок приложения;
* *устойчивость к ошибке* — способность программного продукта поддерживать

заданный уровень качества функционирования в случае ошибок в программах или недопустимом задании интерфейса;

* *восстанавливаемость* — способность программного продукта восстанавливать заданный уровень функционирования и данные, поврежденные в результате отказа. Одним из показателей является длительность восстановления;
* *соответствие надежности* — свойство программного продукта соответствовать

стандартам, соглашениям и нормативным документам.

Выделяют 5 условную подхарактеристику надежности — *готовность*.

44. Внутренние метрики для оценки надежности ПО ISO/IEC 9126-1-4 2001-2004.

* Внутренние метрики —измерят собственные свойства ПС.
* Они измеряются в процессе разработки ПС на основе спецификации требований, результатов проектирования, исходного кода.
* Позволяют измерять качество промежуточных звеньев и таким образом прогнозировать качество конечной продукции.

Примеры метрик:

● Полнота тестирования, оценивает завершенность.

Внутреннее качество определяют как среднеарифметическое внутренних метрик, а внешнее – как среднеарифметическое внешних метрик.

45. Внешние метрики для оценки надежности ПО ISO/IEC 9126-1-4 2001-2004.

* Внешние метрики – измеряют качество ПС путем измерения поведения системы, частью которой является данный ПП.
* *Внешние метрики характеризуют* ***внешнее качество ПО (качество после тестирования)***
* Используются в процессе **эксплуатации** и на стадиях тестирования или испытаний в процессах разработки и **сопровождения** ПС, когда уже созданы исполнимые коды ПП.

К ним относятся:

* Плотность ошибок, оценивает завершенность.
* Предотвращение некорректных действий, оценивает устойчивость к ошибке.
* Способность к восстановлению, оценивает восстанавливаемость.
* Соответствие надежности, оценивает соответствие надежности.

46. Тестирование безопасности ПО.

Тестирование безопасности - это оценка уязвимости программного обеспечения к различного рода атакам.

В ходе тестирования безопасности тестировщик производит следующие действия:

* попытки узнать пароль с помощью внешних средств;
* атака системы с помощью специальных утилит, анализирующих защиту;
* подавление работы с другими пользователями
* просмотр несекретных данных в надежде найти ключ для входа в систему.

Виды уязвимостей

* Cross Site Scripting XSS – JS injection
* Cross Site Request Forgery (межсайтовая подделка запроса)
* Server-Side Includes Injection (инъекции на стороне сервера)
* Authorization Bypass (обход аутентификации)

Принципы безопасности программного обеспечения

* Конфиденциальность - сокрытие определенных ресурсов
* Целостность – доверие(Ожидается, что ресурс будет изменен определенной группой пользователей.)
* Доступность - ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству.

47. Инсталляционное тестирование ПО.

Тестирование установки позволяет удостовериться в том, что ПО корректно устанавливается и настраивается, накат новых версий происходит без ошибок, а также есть возможность деинсталлировать и удалить данное ПО. Тестирование инсталляции необходимо проводить при создании ПО, после появления новой версии, а также при изменении конфигурации стенда.

Данный вид тестирования позволяет избежать следующих проблем:

● Невозможность установить ПО

● Потеря данных после установки новой версии

● Невозможность откатиться до предыдущей версий

Проверки:

* Подсчитывается  ли при установке количество свободного места
* Восстановится ли процесс установки при внезапном его прерывании
* Установка приложения, его запуск, удаление приложения должны возвращать систему в исходное состояние
* Повторный запуск установки приложения при уже текущем должен выдавать корректное сообщение