Тестирование программных продуктов

Системное и прикладное программное обеспечение

Малышенко Владислав Викторович

Тестирование

Тестирование - очень важный и трудоемкий этап процесса разработки программного обеспечения, так как правильное тестирование позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ.

Процесс разработки программного обеспечения предполагает **три стадии тестирования**: **автономное, комплексное и системное**, каждая из которых соответствует завершению соответствующей части Системы.

Различают два подхода к формированию тестов: **структурный** и **функциональный**.

Содержание

- Виды контроля качества разрабатываемого программного обеспечения
- Ручной контроль программного обеспечения
- Структурное тестирование
- Функциональное тестирование
- Тестирования модулей и комплексное тестирование
- Оценочное тестирование

Зависимость вероятности правильного исправления и его стоимости

Современные технологии тестирования

Современные технологии разработки программного обеспечения предусматривают раннее обнаружение ошибок за счет выполнения контроля результатов всех этапов и стадий разработки.

На начальных этапах такой контроль осуществляют в основном вручную или с использованием CASE-средств, на последних - он принимает форму тестирования.

Тестирование

Тестирование - это процесс выполнения программы, целью которого является выявление ошибок.

Никакое тестирование не может доказать отсутствие ошибок в хоть сколько-нибудь сложном программном обеспечении.

Соблюдение основных правил тестирования и научно обоснованный подбор тестов может уменьшить их количество.

Тестирование

Три стадии тестирования:

- **автономное тестирование компонентов** программного обеспечения;
- **комплексное тестирование** разрабатываемого программного обеспечения;
- **системное или оценочное** тестирование на соответствие основным критериям качества.

Основные принципы тестирования

- предполагаемые результаты должны быть известны до тестирования;
- следует избегать тестирования программы автором;
- необходимо досконально изучать результаты каждого теста;
- необходимо проверять действия программы на неверных данных;
- необходимо проверять программу на неожиданные побочные эффекты на неверных данных.

Формирование тестовых наборов

Удачным следует считать тест, который обнаруживает хотя бы одну ошибку.

С этой точки зрения хотелось бы использовать такие наборы тестов, каждый из которых с максимальной вероятностью может обнаружить ошибку.

Формирование набора тестов имеет большое значение, поскольку тестирование является одним из наиболее трудоемких этапов (от 30 до 60 % общей трудоемкости) создания программного продукта.

Причем доля стоимости тестирования в общей стоимости разработки имеет тенденцию возрастать при увеличении сложности программного обеспечения и повышении требований к их качеству.

Два различных подхода к формированию тестовых наборов

- Структурный;
- функциональный.

Структурный подход

Структурный подход базируется на том, что **известна структура** тестируемого программного обеспечения, в том числе его алгоритмы («стеклянный ящик»).

Тесты строят так, чтобы проверить правильность реализации заданной логики в коде программы.

Функциональный подход

Функциональный подход основывается на том, что структура программного обеспечения не известна («черный ящик»).

Тесты строят, опираясь на функциональные спецификации. Этот подход называют также подходом, управляемым данными, так как при его использовании тесты строят на базе различных способов декомпозиции множества данных.

Ручной контроль программного обеспечения

Различают:

- **статический подход.**

  Анализируют структуру, управляющие и информационные связи программы, ее входные и выходные данные.

- **динамический подход.**

  Выполняют ручное тестирование, т. е. вручную моделируют процесс выполнения программы на заданных исходных данных.

Ручной контроль программного обеспечения

Исходными данными для таких проверок являются: техническое задание, спецификации, структурная и функциональная схемы программного продукта, схемы отдельных компонентов, алгоритмы, тексты, тестовые наборы.

Доказано, что ручной контроль способствует существенному увеличению производительности и повышению надежности программ и с его помощью можно находить от 30 до 70 % ошибок логического проектирования и кодирования.

Методы ручного контроля обязательно должны использоваться в каждом программном проекте.

Основные методы ручного контроля

Основными методами ручного контроля являются:

- инспекции исходного текста,
- сквозные просмотры,
- проверка за столом,
- оценки программ.

Инспекции исходного текста

Инспекции исходного текста представляют собой набор процедур и приемов обнаружения ошибок при изучении текста группой специалистов.

В эту группу входят: автор программы, проектировщик, специалист по тестированию и координатор.

Общая процедура инспекции предполагает следующие операции:

- участникам группы заранее выдается листинг программы и спецификация на нее;
- программист рассказывает о логике работы программы и отвечает на вопросы инспекторов;
- программа анализируется по списку вопросов для выявления исторически сложившихся общих ошибок программирования.

1. Контроль обращений к данным

- Все ли переменные инициализированы?
- Не превышены ли максимальные (или реальные) размеры массивов и строк?
- Не перепутаны ли строки со столбцами при работе *с* матрицами?
- Присутствуют ли переменные со сходными именами?
- Используются ли файлы? Если да, то при вводе из файла проверяется ли завершение файла?
- Соответствуют ли типы записываемых и читаемых значений?
- Использованы ли нетипизированные переменные, открытые массивы, динамическая память? Если да, то соответствуют ли типы переменных при «наложении» формата? Не выходят ли индексы за границы массивов?

2. Контроль вычислений

- Правильно ли записаны выражения (порядок следования операторов)?
- Корректно ли выполнены вычисления над неарифметическими переменными?
- Корректно ли выполнены вычисления с переменными различных типов (в том числе с использованием целочисленной арифметики)?
- Возможно ли переполнение разрядной сетки или ситуация машинного нуля?
- Соответствуют ли вычисления заданным требованиям точности?
- Присутствуют ли сравнения переменных различных типов?

3. Контроль передачи управления

- Будут ли корректно завершены циклы?
- Будет ли завершена программа?
- Существуют ли циклы, которые не будут выполняться из-за нарушения условия входа? Корректно ли продолжатся вычисления?
- Существуют ли поисковые циклы? Корректно ли отрабатываются ситуации «элемент найден» и «элемент не найден»?

4. Контроль межмодульных интерфейсов

- Соответствуют ли списки параметров и аргументов по порядку, типу, единицам измерения?
- Не изменяет ли подпрограмма аргументов, которые не должны изменяться?
- Не происходит ли нарушения области действия глобальных и локальных переменных с одинаковыми именами?

Сквозные просмотры

Сквозной просмотр, как и инспекция, представляет собой набор способов обнаружения ошибок, осуществляемых группой лиц, просматривающих текст программы.

Такой просмотр имеет много общего с процессом инспектирования, но отличается процедурой и методами обнаружения ошибок.

Группа по выполнению сквозного контроля состоит из трех-пяти человек: председатель или координатор, секретарь, специалист по тестированию, программист и независимый эксперт.

Сквозные просмотры

Сквозной просмотр предполагает выполнение следующих процедур:

В большинстве сквозных просмотров при выполнении самих тестов находят меньше ошибок, чем при опросе программиста.

- каждый тест в соответствии с логикой программы, при этом состояние программы отслеживается на бумаге или доске;
- при необходимости программисту задают вопросы о логике проектирования и принятых допущениях.

Проверка за столом

Проверка исходного текста, выполняемая одним человеком, который читает текст программы, проверяет его на наличие возможных ошибок по специальному списку часто встречающихся ошибок и «пропускает» через программу тестовые данные.

Проверку за столом должен проводить человек, не являющийся автором программы.

Метод наименее результативен, так как проверка представляет собой полностью неупорядоченный процесс, при ней отсутствует обмен мнениями и здоровая конкуренция.

Оценка программ

Этот метод непосредственно не связан с тестированием, но его использование также улучшает качество программирования.

Его используют для анонимной оценки программы в терминах ее общего качества, простоты эксплуатации и ясности.

Цель метода - обеспечить сравнительно объективную оценку и самооценку программистов.

Структурное тестирование

Структурное тестирование. Тестовые наборы формируют путем анализа маршрутов, предусмотренных алгоритмом.

Под **маршрутами** при этом понимают последовательности операторов программы, которые выполняются при конкретном варианте исходных данных.

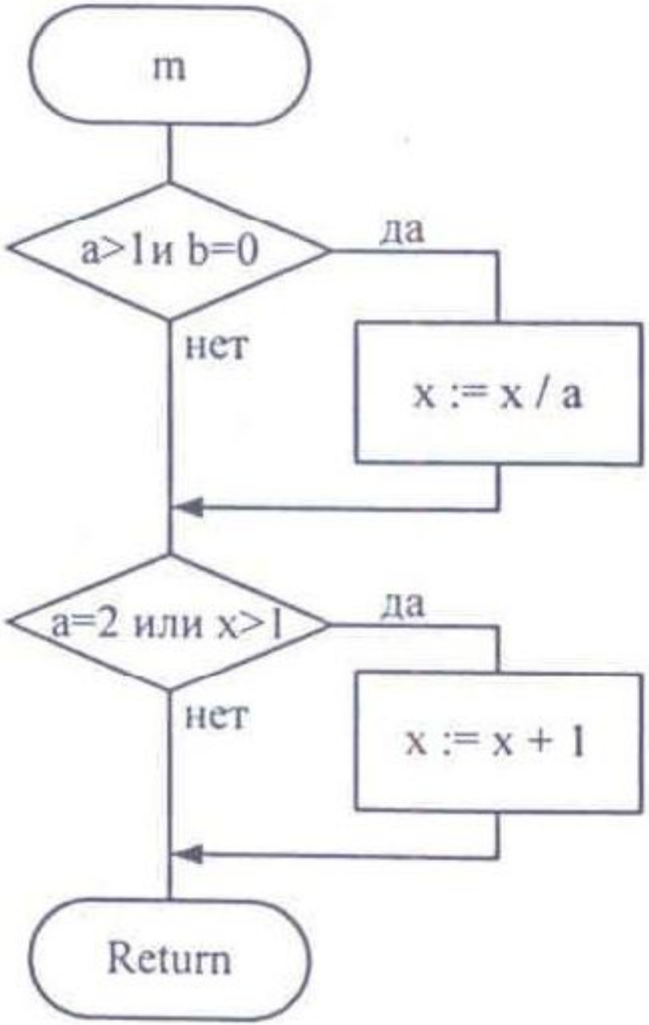
В основе структурного тестирования лежит концепция максимально полного тестирования всех маршрутов программы.

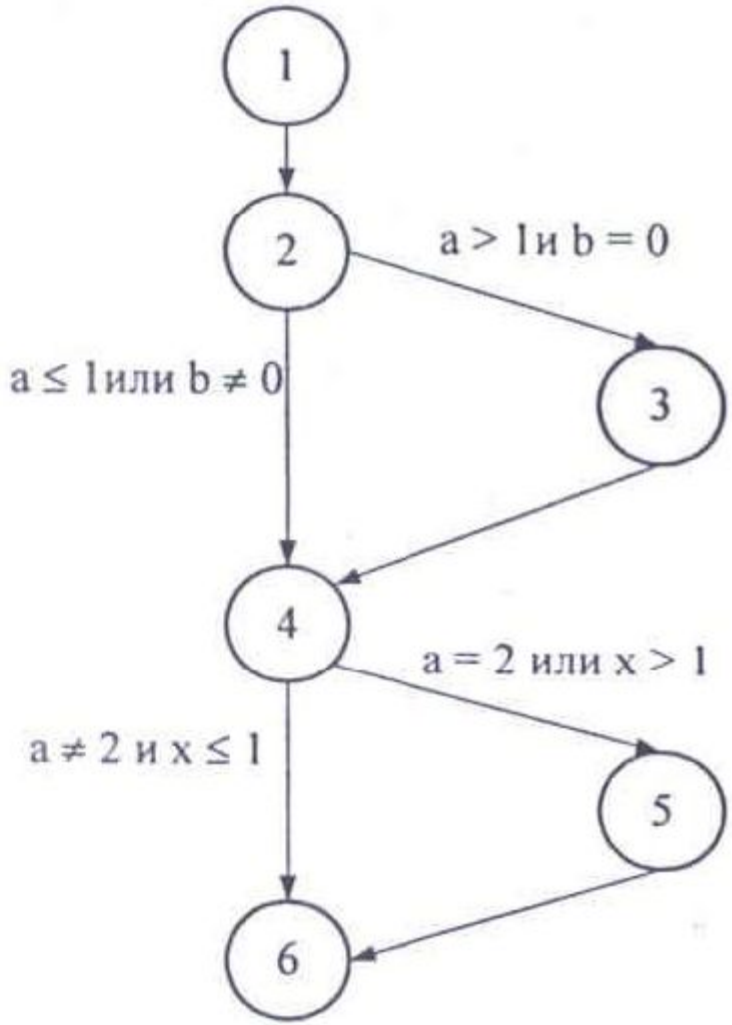
Структурный подход. Недостатки

Тестовые наборы, построенные по данной стратегии:

- не обнаруживают пропущенных маршрутов;
- не обнаруживают ошибок, зависящих от обрабатываемых данных, например, в операторе *if* (*a* - *b*) < *eps* - пропуск функции абсолютного значения *abs* проявится только, если *a* < *b*;
- не дают гарантии, что программа правильна, например, если вместо сортировки по убыванию реализована сортировка по возрастанию

```
Procedure m (a, b: real; var x: real);
  begin
    if (a1) and (b=0) then x: =x/a;
    if (a=2) or (x>1) then x: =x+1;
  end;
```





Формирование тестовых наборов

Формирование тестовых наборов для тестирования маршрутов может осуществляться по нескольким критериям:

- покрытие операторов;
- покрытие решений (переходов);
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

Покрытие операторов

Критерий покрытия операторов подразумевает такой подбор тестов, чтобы каждый оператор программы выполнялся, по крайней мере, один раз.

Это необходимое, но недостаточное условие для приемлемого тестирования.

Покрытие решений (переходов)

Для реализации этого критерия необходимо такое количество и состав тестов, чтобы результат проверки каждого условия принимал значения «истина» или «ложь», по крайней мере, один раз.

Критерий покрытия решений удовлетворяет критерию покрытия операторов, но является более «сильным».

Покрытие условий

В этом случае формируют некоторое количество тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении были выполнены, по крайней мере, один раз.

Однако, как и в случае покрытия решений, этот критерий не всегда приводит к выполнению каждого оператора, по крайней мере, один раз.

К критерию требуется дополнение, заключающееся в том, что каждой точке входа управление должно быть передано, по крайней мере, один раз.

Покрытие решений/условий

Согласно этому методу тесты должны составляться так, чтобы, по крайней мере, один раз выполнились все возможные результаты каждого условия и все результаты каждого решения, и каждому оператору управление передавалось, по крайней мере, один раз.

Комбинаторное покрытие условий

Этот критерий требует создания такого множества тестов, чтобы все возможные комбинации результатов условий в каждом решении и все операторы выполнялись, по крайней мере, один раз.

Функциональное тестирование

Одним из способов проверки программ является тестирование с управлением по данным или по принципу «черного ящика».

В этом случае программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

Функциональное тестирование

Для обнаружения всех ошибок в программе, используя управление по данным, необходимо выполнить **исчерпывающее тестирование**.

Для программ, где исполнение команды зависит от предшествующих ей событий, необходимо проверить и все возможные последовательности.

Проведение исчерпывающего тестирования невозможно. Поэтому обычно выполняют «разумное» или «приемлемое» тестирование. Этот вариант не дает гарантии отсутствия отклонений от спецификаций.

Функциональное тестирование

Правильно выбранный тест должен уменьшать, причем более чем на единицу, число других тестов, которые должны быть разработаны для обеспечения требуемого качества программного обеспечения.

Функциональное тестирование

При функциональном тестировании различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

Эквивалентное разбиение

Область всех возможных наборов входных данных программы по каждому параметру разбивают на конечное число групп - **классов эквивалентности**.

Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок:

- если набор какого-либо класса обнаруживает некоторую ошибку, то предполагается, что все другие тесты этого класса эквивалентности тоже обнаружат эту ошибку и наоборот.

Классы эквивалентности

Разработку тестов методом эквивалентного разбиения осуществляют в два этапа:

1. выделяют классы эквивалентности
2. формируют тесты.

Выделение классов эквивалентности является эвристическим процессом, однако целесообразным считают выделять в отдельные классы эквивалентности наборы, содержащие допустимые и недопустимые значения некоторого параметра.

Классы эквивалентности. Правила

- если некоторый параметр x может принимать значения в интервале [1, 999], то выделяют один правильный класс $1 \le x \le 999$ и два неправильных: $x < 1$ и $x > 999$;
- если входное условие определяет диапазон значений порядкового типа, например, «в автомобиле могут ехать от одного до шести человек», то определяется один правильный класс эквивалентности и два неправильных: ни одного и более шести человек;
- если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо, например, «типы графических файлов: bmp, jpeg, vsd», то определяют правильный класс эквивалентности для каждого значения и один неправильный класс, например, txt;

Классы эквивалентности. Правила (2)

- если входное условие описывает ситуацию «должно быть», например, «первым символом идентификатора должна быть буква», то определяется один правильный класс эквивалентности (первый символ - буква) и один неправильный (первый символ - не буква);
- если есть основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

Классы эквивалентности. Формирование тестов

При построении тестов правильных классов учитывают, что каждый тест должен проверять по возможности максимальное количество различных входных условий. Такой подход позволяет минимизировать общее число необходимых тестов.

Для каждого неправильного класса эквивалентности формируют свой тест. Последнее обусловлено тем, что определенные проверки с ошибочными входами скрывают или заменяют другие проверки с ошибочными входами.

Анализ граничных значений

Граничные значения - это значения на границах классов эквивалентности входных значений или около них.

Анализ показывает, что в этих местах резко увеличивается возможность обнаружения ошибок.

Например, если в программе анализа вида треугольника было записано A + B ≥ C вместо A + B > C, то задание граничных значений приведет к ошибке: линия будет отнесена к одному из видов треугольника.

Анализ причинно-следственных связей

Анализ причинно-следственных связей позволяет системно выбирать высоко результативные тесты. Метод использует алгебру логики и оперирует понятиями «причина» и «следствие».

Причиной – называют отдельное входное условие или класс эквивалентности.

Следствием – выходное условие или преобразование системы.

Идея метода заключается в отнесении всех следствий к причинам. Данный метод дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

Предположение об ошибке

Часто программист с большим опытом находит ошибки, «не применяя никаких методов». На самом деле он подсознательно использует метод «предположение об ошибке».

Процедура метода предположения об ошибке в значительной степени основана на интуиции.

Основная его идея заключается в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты.

Другими словами, требуется перечислить те особые случаи, которые могут быть не учтены при проектировании.

Тестирования модулей и комплексное тестирование

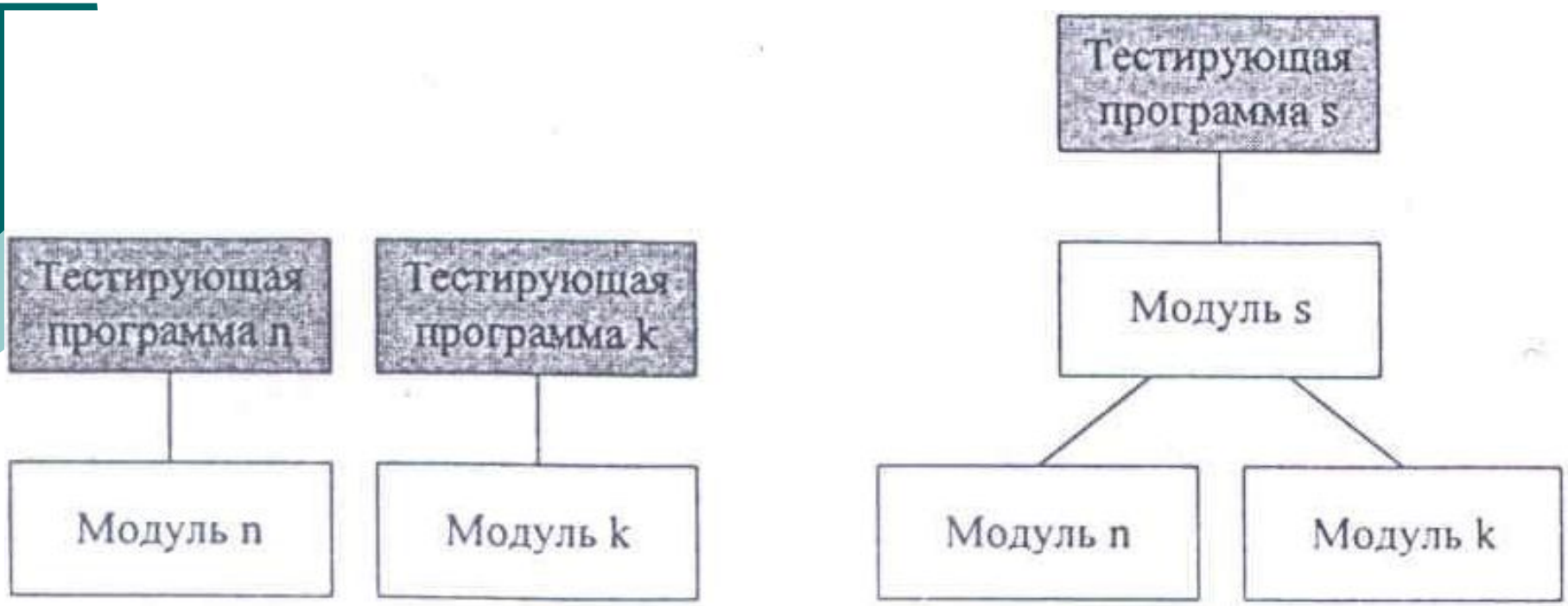
Восходящее тестирование.

Восходящий подход предполагает, что каждый модуль тестируют отдельно на соответствие имеющимся спецификациям на него, затем собирают оттестированные модули в модули более высокой степени интеграции и тестируют их.

Проверяются межмодульные интерфейсы, используемые для подключения модулей более низкого уровня иерархии.

И так пока не будет собран весь программный продукт.

Восходящее тестирование



Восходящее тестирование. Достоинства и недостатки

Достоинства: обеспечивается полностью автономное тестирование, для которого просто генерировать тестовые последовательности, которые передаются в модуль напрямую.

Недостатки:

1. При восходящем тестировании серьезные ошибки в спецификациях, алгоритмах и интерфейсе могут быть обнаружены только на завершающей стадии работы над проектом.
2. Для того, чтобы тестировать модули нижних уровней, необходимо разработать специальные тестирующие программы, которые обеспечивают вызов модулей с необходимыми параметрами.
3. Причем эти тестирующие программы также могут содержать ошибки.

Нисходящее тестирование

Нисходящее тестирование органически связано с нисходящим проектированием и разработкой; как только проектирование какого-либо модуля заканчивается, его кодируют и передают на тестирование.

Автономно тестируется только основной модуль. При его тестировании все вызываемые им модули заменяют модулями, которые в той или иной степени имитируют поведение вызываемых модулей. Такие модули принято называть «**заглушками**».

Нисходящее тестирование

Нисходящее тестирование .
Достоинства и недостатки

Основной **недостаток** –

отсутствие автономного тестирования модулей.

Основное **достоинство** –

ранняя проверка основных решений и качественное многократное тестирование сопряжения модулей в контексте программного обеспечения.

Есть возможность согласования с заказчиком внешнего вида программного обеспечения.

Комбинированный подход

применяют следующим образом:

- **Модули верхних уровней** тестируют нисходящим способом;
- **Модули нижних уровней** - восходящим.

Этот способ позволяет:

- Проводить тестирование интерфейса;
- Обеспечить качественное автономное тестирование модулей низших уровней.

Тестирование программного обеспечения специалистами

Согласно основным принципам нежелательно тестирование программного обеспечения его автором.

Задачей специалиста по тестированию является обнаружение максимального количества несоответствий тестируемого модуля и спецификаций на него.

Для выполнения этой задачи специалист по тестированию формирует тесты, обеспечивая всестороннее тестирование.

Название компании *Конфиденциально* Отчет о проблеме
№___
Программа _____ Выпуск _____ Версия _____

Название компании *Конфиденциально* Отчет о проблеме
№___

Заполняется разработчиком

Функциональная область _____ Ответственный _____

Комментарии _____

Состояние (1-2)___ Приоритет (1-5)__

1 - Открыто 2 - Закрыто

Резолюция (1-9)___ Исправленная версия _____

1 - Рассматривается
2 - Исправлено
3 - Не воспроизводится
4 - Отложено
5 - Соответствует проекту
6 - Не может быть исправлено
7 - Отозвано составителем
8 - Нужна дополнительная информация
9 - Не согласен с предложением

Рассмотрено _____ Дата __/__/___

Проконтролировано _____ Дата __/__/___

Считать отложенным (Д/Н)

Комплексное тестирование

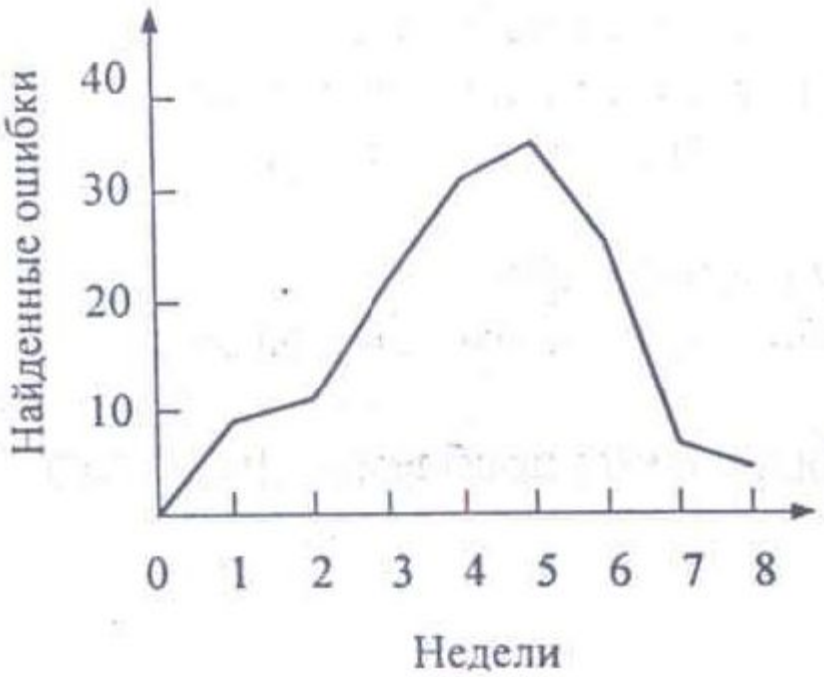
Особенностью комплексного тестирования является то, что структурное тестирование для него практически не применимо.

В основном на данной стадии используют тесты, построенные по методам эквивалентных классов, граничных условий и предположении об ошибках.

Критерии завершения тестирования и отладки

Одним из самых сложных является вопрос о том, **когда следует завершать тестирование**, поскольку невозможно гарантировать, что в разрабатываемом программном обеспечении не осталось ошибок.

Крите



Три группы

- основанн ектирования тестов ство тестов перестаю
- основанн количества ошибок шают при нахожден к;
- основанные на исследовании результатов тестирования - строят график зависимости количества обнаруженных ошибок от времени тестирования.

Критерии завершения

Часто тестирование завершают потому, что закончилось время.

Минимальное тестирование предполагает:

- тестирование граничных значений;
- тщательную проверку руководства;
- тестирование минимальных конфигураций технических средств;
- тестирование возможности редактирования команд и повторения их в любой последовательности;
- тестирование устойчивости к ошибкам пользователя.

Оценочное тестирование

Оценочное тестирование

Цель оценочного тестирования является тестирование программы на соответствие основным требованиям.

Эта стадия тестирования особенно важна для программных продуктов, предназначенных для продажи на рынке.

Оценочное тестирование. Виды тестирования

- тестирование удобства использования;
- тестирование на предельных объемах;
- тестирование на предельных нагрузках;
- тестирование удобства эксплуатации;
- тестирование защиты;
- тестирование производительности;
- тестирование требований к памяти;
- тестирование конфигурации оборудования;
- тестирование совместимости;
- тестирование удобства установки;
- тестирование надежности;
- тестирование восстановления;
- тестирование удобства обслуживания;
- тестирование документации;
- тестирование процедуры.

Оценочное тестирование. Виды тестирования

- тестирование удобства использования - последовательная проверка соответствия программного продукта и документации на него основным положениям технического задания;
- тестирование на предельных объемах - проверка работоспособности программы на максимально больших объемах данных, например, объемах текстов, таблиц, большом количестве файлов и т. п.;

Оценочное тестирование. Виды тестирования

- тестирование на предельных нагрузках - проверка выполнения программы на возможность обработки большого объема данных, поступивших в течение короткого времени;
- тестирование удобства эксплуатации - анализ психологических факторов, возникающих при работе с программным обеспечением; это тестирование позволяет определить, удобен ли интерфейс, не раздражает ли цветовое или звуковое сопровождение и т. п.;
- тестирование защиты - проверка защиты, например, от несанкционированного доступа к информации;

Оценочное тестирование. Виды тестирования

- тестирование производительности - определение пропускной способности при заданной конфигурации и нагрузке;
- тестирование требований к памяти - определение реальных потребностей в оперативной и внешней памяти;
- тестирование конфигурации оборудования - проверка работоспособности программного обеспечения на разном оборудовании;

Оценочное тестирование. Виды тестирования

- тестирование совместимости - проверка преемственности версий: в тех случаях, если очередная версия системы меняет форматы данных, она должна предусматривать специальные конвекторы, обеспечивающие возможность работы с файлами, созданными предыдущей версией системы;
- тестирование удобства установки - проверка удобства установки;
- тестирование надежности - проверка надежности с использованием соответствующих математических моделей;

Оценочное тестирование. Виды тестирования

- тестирование восстановления - проверка восстановления программного обеспечения, например системы, включающей базу данных, после сбоев оборудования и программы;
- тестирование удобства обслуживания - проверка средств обслуживания, включенных в программное обеспечение;
- тестирование документации - тщательная проверка документации, например, если документация содержит примеры, то их все необходимо попробовать;
- тестирование процедуры - проверка ручных процессов, предполагаемых в системе.

Оценочное тестирование

Целью всех проверок является поиск несоответствий техническому заданию.

Только после выполнения всех видов тестирования программный продукт может быть представлен пользователю или к реализации.

Однако на практике обычно выполняют не все виды оценочного тестирования, так как это очень дорого и трудоемко.

Для каждого типа программного обеспечения выполняют те виды тестирования, которые являются для него наиболее важными.

Еще не все ?

Содержание

- Виды контроля качества разрабатываемого программного обеспечения
- Ручной контроль программного обеспечения
- Структурное тестирование
- Функциональное тестирование
- Тестирования модулей и комплексное тестирование
- Оценочное тестирование