Оглавление

ТПО

Лекция 1(02.09.2020)

Формальное определение тестирования

Тестирование является одним из основных инструментов обеспечения качества продуктов. Оно также является одной из фаз разработки (~40%).

Программа – это аналог формулы в обычной математике.

Формула для функции f, полученной суперпозицией функций f1, f2 … fn – выражение, описывающее эту суперпозицию:

f=f1*f2*f3*…*fn

Если аналог f1, f2 … fn – операторы языка программирования, то их формула – программа.

Методы обоснования истинности формул:

1. Формальный подход – используя формальные процедуры и аксиом преобразовать левую и правую часть к одному и тому же виду.

Позволяет избежать работы с большими наборами значений переменных, но к тестированию программ этот подход не применим.

2. Интерпретационный – тождество проверяется на всех возможных наборах переменных.

Недостаток в том, что наборов может быть очень много.

Основные понятия тестирования

Отладка – процесс поиска, локализации и исправления ошибок программы. Если программа не содержит синтаксических ошибок, она может быть скомпилирована и выполнена, то она в любом случае выполняет какую-то функции (отображение множества входных данных на множество выходных).Судить о правильности или неправильностирезультатов выполнения программы можно только сравнивая спецификацию желаемой функции с её результатами.

Подходы к отладке:

1. Выполнение программы в уме

2. Добавление в программу операторов отладочной печати или протоколированная промежуточных результатов

3. Пошаговое выполнение программы

4. Выполнение программы с заранее заданными точками останова (Breakpoints)

5. Реверсивное выполнение программы

Определения «Тестирование»

Тестирование – это процесс выполнения программного обеспечения, системы или компонента в условии анализа или записи получаемых результатов с целью проверки или оценки некоторых свойств тестируемого объекта.

Тестирование – процесс анализа пункта требований к программному обеспечению с целью фиксацией различия между существующей программой и требуемой при экспериментальной проверке соответствующего пункта требований.

Тестирование (IEEE Std 829-1983) – это контролируемое выполнение программы на конечном множестве тестовых данных и анализ результатов этого выполнения для поиска ошибок.

Тестирование обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок).

Как правило, на фазе тестирования осуществляется и исправление идентифицированных ошибок, включающее локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы тестируемого приложения (Application Under Testing (AUT) или Implementation Under Testing(IUT)).

Оракул дает заключение о факте появления неправильной пары $(x,y_в)$ и ничего не говорит о том, каким образом она была вычислена или каков правильный алгоритм – он только сравнивает вычисленные и желаемые результаты. Оракулом может быть даже Заказчик или программист, производящий соответствующие вычисления в уме, поскольку Оракулу нужен какой-либо альтернативный способ получения функции (X,Y) для вычисления эталонных значений Y

В процессе тестирования Оракул последовательно получает элементы множества (X,Y) и соответствующие им результаты вычислений $(X,Y_в)$ для идентификации фактов несовпадений (test incident).

При выявлении $(x,y_в)$, не принадлежащего (X,Y), запускается процедура исправления ошибки, которая заключается во внимательном анализе (просмотре) протокола промежуточных вычислений, приведших к $(x,y_в)$, с помощью следующих методов:

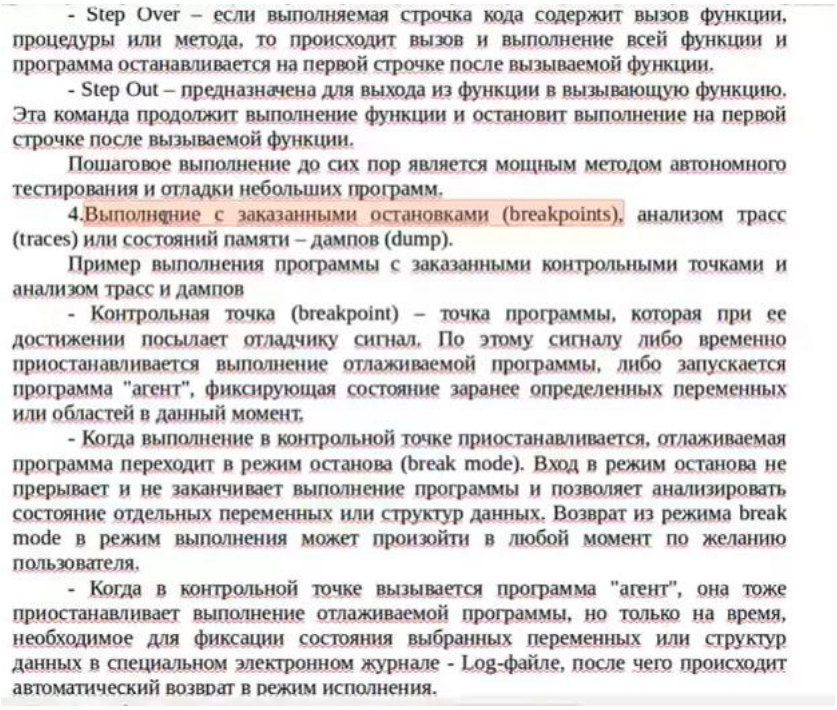
1."Выполнение программы в уме" (deskchecking).

2.Вставка операторов протоколирования (печати) промежуточных результатов (logging).

3.Пошаговое выполнение программы (single-step running).

При пошаговом выполнении программы код выполняется строчка за строчкой. Используются следующие команды пошагового выполнения:

- Step Into – если выполняемая строчка кода содержит вызов функции, процедуры или метода, то происходит вызов, и программа останавливается на



- Step Over – если выполняемая строчка кода содержит вызов функции, процедуры или метода, то происходит вызов и выполнение всей функции и программа останавливается на первой строчке после вызываемой функции.

- Step Out – предназначена для выхода из функции в вызывающую функцию. Эта команда продолжит выполнение функции и остановит выполнение на первой строчке после вызываемой функции.

Пошаговое выполнение до сих пор является мощным методом автономного тестирования и отладки небольших программ.

4.Выполнение с заказанными остановками (breakpoints), анализом трасс (traces) или состояний памяти – дампов (dump).

Пример выполнения программы с заказанными контрольными точками и анализом трасс и дампов

- Контрольная точка (breakpoint) – точка программы, которая при ее достижении посылает отладчику сигнал. По этому сигналу либо временно приостанавливается выполнение отлаживаемой программы, либо запускается программа "агент", фиксирующая состояние заранее определенных переменных или областей в данный момент.

- Когда выполнение в контрольной точке приостанавливается, отлаживаемая программа переходит в режим останова (break mode). Вход в режим останова не прерывает и не заканчивает выполнение программы и позволяет анализировать состояние отдельных переменных или структур данных. Возврат из режима break mode в режим выполнения может произойти в любой момент по желанию пользователя.

- Когда в контрольной точке вызывается программа "агент", она тоже приостанавливает выполнение отлаживаемой программы, но только на время, необходимое для фиксации состояния выбранных переменных или структур данных в специальном электронном журнале - Log-файле, после чего происходит автоматический возврат в режим исполнения.

Виды тестирования:

Статическое тестирование выявляет формальными методами анализа без выполнения тестируемой программы неверные конструкции или неверные отношения объектов программы (ошибки формального задания) с помощью специальных инструментов контроля кода - CodeChecker.

Динамическое тестирование (собственно тестирование) осуществляет выявление ошибок только на выполняющейся программе с помощью специальных инструментов автоматизации тестирования - Testbed или Testbench.

Тестирование включает 3 фазы:

1. Создание тестового набора вручную или с помощью автоматической генерации

2. Прогон программы на ранее определённых тестовых данных

3. Оценка результатов тестирования с целью обнаружения ошибок

Проблемы тестирования :

1. Невозможно тестировать на всех вариантах значений входных данных (их может быть неограниченно много)

2. Невозможно тестировать на всех путях выполнения программы

3. Нет способов гарантировать завершение программы на любом тесте

Основная проблема тестирования – определение минимального набора тестов для проверки правильности программ.

Требования к идеальному критерию тестирования

I. Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования

2. Критерий должен быть полным, т.е, в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

3. Критерий должен быть надежным, те, любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы

4. Критерий должен быть легко проверяемым, например вычисляемым на тестах

Критерии тестирования нужны для оценки качества теста.

Классы критериев

1. ***Структурные*** критерии используют информацию о структуре программы (критерии так называемого "белого ящика").

2. ***Функциональные*** критерии формулируются в описании требований к программному продукту (критерии так называемого "черного ящика").

3. Критерии ***стохастического*** тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы.

4.***Мутационные*** критерии ориентированы на проверку свойств программного продукта на основе подхода Монте-Карло.

Структурные критерии:

Используется модель программы – белый ящик (предполагается, что известен код или хотя бы его модель в виде графа потоков управления). Структурные критерии используются при модульном и мутационном тестировании.

Варианты структурных критериев:

• Критерий тестирования команд – набор тестов должен обеспечить прохождения каждой команды не меньше одного раза (применяется в больших системах, когда другие критерии применить невозможно)

• Критерий тестирования ветвей – набор тестов должен обеспечить прохождение каждой ветви не менее одного раза (достаточно экономичный критерий, так как путей на графе потоков управления не так много)

• Критерий тестирования путей – если путь на графе включает циклический оператор, то тест строится так, что бы проходилась только одна итерация цикла

Функциональные критерии:

Рассматривают программный продукт в целом и поэтому учитывают взаимодействие приложения с его окружением. Основная проблема – трудоёмкость (слишком много тестов).

Варианты функциональных критериев:

• Тестирование пунктов спецификации – каждый пункт спецификации проверяется по меньшей мере один раз

• Тестирование классов входных данных (области эквивалентности)

• Тестирование классов выходных данных – аналогично областям эквивалентности входных данных, можно построить их для выходных

• Тестирование функций – каждая функция должна быть проверена хотя бы один раз (могут возникать интеграционные ошибки)

Стохастические критерии:

Применяется для тестирования сложных модулей с большими объемами входных данных, которые нельзя разбить на области эквивалентности. Разрабатываются программы-имитаторы случайных входных данных.

1. Генерация набора входных данные

2. Независимое получение набора ожидаемых выходных данных

3. Тестирование :

3.1. Детерминированны – сравнение ожидаемых с фактическими выходными данными

3.2. Стохастический (если невозможно получить ожидаемые выходные данные) – сравниваются не фактические значения, а их законы распределения. В качестве используются методы проверки гипотез (Стьюдента, хи-квадрат)

Мутационные критерии:

Основывается на том допущении, что программисты пишут «почти» правильные программы (программа не содержит больших логических ошибок, а только незначительные опечатки и недосмотры?). Мутанты – это программы, которые отличаются друг от друга мутациями.

Изначально в программу P вносятся некоторые искусственные мутации (мелкие ошибки в программе). Программа P и её набор мутантов тестируется на одном и том же наборе тестов. Если набор тестов подтверждает правильность программы P и также выявляет все искусственные ошибки в программах мутантах, то этот тест является качественным. Если не все мутации выявлены, то набор тестов нужно расширить.

Модульное тестирование ПО

Модульное тестирование – это тестирование программы на уровне отдельно взятых модулей (минимальная компилируемая единица: ООП – класс, процедурное – функция).

Цель – выявления локализованных внутри модуля ошибок(алгоритмические), проверка готовности системы перейти на следующий уровень тестирования. ***Принцип белого ящика*** – тестирование модуля, зная его внутреннюю структуру.(структурные критерии тестирования)

Модульное тестирование предполагает создание вокруг модуля тестовой среды. Тестовое окружение создаёт заглушки для всех интерфейсов тестируемого модуля. Эти заглушки используются для подачи входных данных и анализа выходных. Также есть заглушки, имитирующие взаимодействие с внешними модулями.

На этапе модульного тестирования обнаруживаются алгоритмические ошибки внутри одного модуля. Но не обнаруживаются ошибки взаимодействия с другими модулями, неверной трактовки данных, некорректной реализации интерфейса, связанные с производительностью и расходом разных видов ресурсов.

Принципы модульного тестирования

Модульные тесты строятся на основе одного из следующих принципов:

1. Анализ потоков управления – граф потоков управления

Может применятся критерий покрытия функций, в соответствии с которым, каждая функция должна быть протестирована хоть раз. Критерий покрытия вызовов – каждый вызов должен быть произведен хотя бы один раз.

Строим граф потоков управления, проверяются вершины графа потока управления, дуги, его пути, условные операторы. Критерий покрытия функций или вызовов.

2. Анализ потоков данных – информационный граф программы

Строиться информационный граф программы, упор на выявление ошибок потока с данными. Внутри модуля.

Выявление аномалий потока данных (ссылки на не инициализированные переменные, избыточные присваивания и т.д.)

Для построения тестов модульного тестирования с требуемой степенью тестируемости необходимо:

1. Построить граф потоков управления – на основе статического анализа программного кода, при этом учитывается критерий

2. Определить на нём тестовые пути – определяются тестовые пути на основе статических, динамических или с помощью метода реализуемых путей

2.1. ***Статические методы*** – путь строится от входной вершины графа, постепенно добавляя по одной дуге к пути, пока не будет достигнута выходная вершина графа. Основной недостаток – не учитываются не реализуемые пути

+ Не высокие требования к ресурсам как для построения тестов и самого тестирования

– Непредсказуем процент бракованных тестовых случаев (не реализуемых путей)

– Переход от покрывающего множества путей к множеству тестовые необходимо произвести вручную

2.2. ***Динамические методы*** – предполагают построение полной системы тестов, удовлетворяющих заданному критерию, путём одновременного решения задачи построения покрывающего множества путей и тестовых данных, при этом автоматически учитывается реализуемость или не реализуемость рассматриваемых путей на графе.

В отличии от статических предполагается, что путь на графе строится на основе постепенного добавления дуг к пути, при этом не нарушая реализуемость и покрывая тестами структурные элементы программы

– Требуется больше ресурсов как при разработке, так и при тестирования

+ Более качественный набор тестов (без не реализуемых путей)

2.3. ***Методы реализуемых путей*** – из множества возможных путей на графе потоков управления выделяется множество реализуемых путей, из которых выбирается подмножество, покрывающее все необходимые структурные критерии

– Ещё больше ресурсов

3. Сгенерировать тесты, соответствующие каждому из тестируемых путей – для известных тестовых путей определяются данные, которые обеспечивают прохождение этих путей.

1) Stub

2) Mock

Интеграционное тестирование

Интеграционное тестирование – это тестирование части системы, состоящей из двух и более модулей.

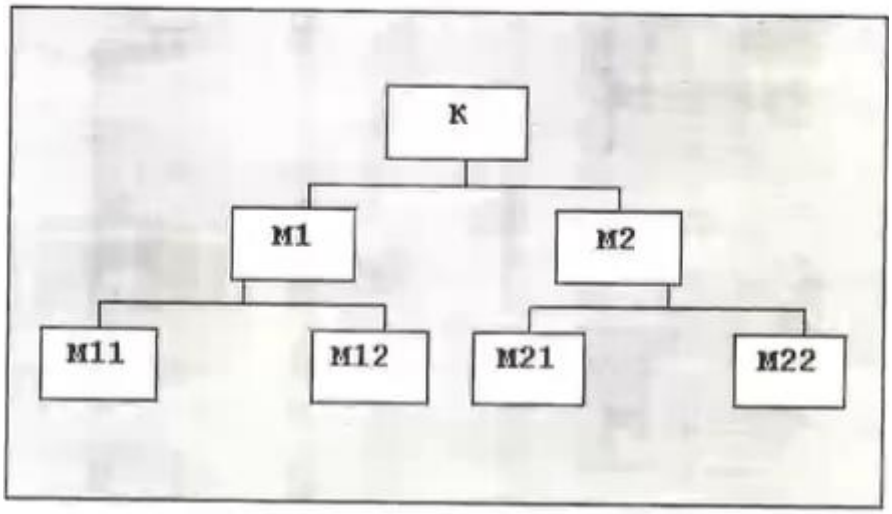
Цель – поиск дефектов, связанные с взаимодействием модулей, включая ошибки реализации и интерпретации интерфейсов.

Вокруг тестируемой части программы создаётся тестовое окружение, которое оперирует интерфейсами модулей, отсутствующие модули заменяются программными заглушками, посредством интерфейса подаются какие-то данные на вход интерфейса.

Интеграционное тестирование – это количественное развитие модульного. В модульном – оперируем интерфейсом модуля. В интеграционном – оперируем объединёнными интерфейсами нескольких модулей.

Интеграционное отличается от модульного только целями тестирования, методика тестирования не отличается.

При построении интеграционных тестов – используются методы с покрытием интерфейсов (покрытие вызовов функции) т.е функциональные критерии.



Корневой модуль включает два модуля, эти модули включают подмодули.

Задача – тестирование межмодульных связей.

При построении модульных тестов, используем модель использует модель белого ящика на модульном уровне.

Два подхода к интеграции модулей(сборке):

1. метод большого взрыва –все модули объединяются одновременно
2. метод последовательной интеграции – добавляем модули по-одному

Интеграция сверху-вниз и снизу-вверх

Интеграционное тестирование отличается от модульного целями тестирования (типами обнаруживаемых дефектов), а разные типы обнаруживаемых дефектов в свою очередь определяют порядок выбора тестовых данных. Применяются методы, связанные с покрытием интерфейсов. Используется подход белого ящика, но до модульного уровня. Поскольку интеграционное тестирование выполняется на этапе сборки модулей, необходимо учитывать основные методы интеграции: метод большого взрыва (всё интегрируется одновременно) и инкрементарный (пошаговый) метод (после добавления очередного модуля выполняется модульное тестирование).

При интеграции методом большого взрыва требуются ***большие затраты на интеграционное тестирование***, связанные со следующим: необходимо разработать большое количество тестовых драйверов и заглушек, повышается сложность идентификации ошибок (слишком большое пространство кода).

Пошаговый метод может быть либо сверху-вниз (нисходящее – сначала модули первого уровня при заглушках второго уровня и так далее), либо снизу-вверх (восходящее – сначала разрабатывается модули нижнего уровня, заменяя заглушками модули на уровень выше, постепенно доходя до модулей первого уровня). Трудоёмкость ниже.

Нисходящее тестирование.

При тестировании модулей верхнего уровня нет модулей уровнем ниже, что требует создавать их сложные программные заглушки. Проблема определения приоритетов модулей. Затрудняется параллельная разработка модулей разных уровней.

Восходящее тестирование.

Тестирование концептуальных особенностей системы в целом выполняется на последнем этапе, когда что-то менять уже поздно.

Виды ошибок

1. Ошибки в реализации интерфейсов
2. Ошибки интерпретации интерфейсов – виноват клиентский код

Процедурные подход подходит для JS.Представляет собой иерархию вызовов. Древовидная структура. Каждый модуль должен иметь одну точку входа и одну точу выхода.

Области эквивалентности входных данных — это множества данных, все элементы которых обрабатываются одинаково.

Области эквивалентности выходных данных — это данные на выходе программы, имеющие общие свойства, которые позволяют считать их отдельным классом. Корректные и некорректные входные данные также образуют две области эквивалентности. Области эквивалентности определяются на основании программной спецификации или документации пользователя и опыта испытателя, выбирающего классы значений, входных данных, пригодные для обнаружения дефектов.

11.11.2020

Тестирование интерфейса(интеграционный)ТИ

Каждый модуль и каждая система имеет определённый интерфейс.

Цель тестирования интерфейса – нахождение ошибок,которые возникают в системе из-за ошибок интерфейсов или ошибок из-за неправильных предположений об интерфейсах.

ТИ важен в ООП

Во время тестирования отдельного объекта невозможно выявить ошибки интерфейсов,т.к. его ошибки могут увидеть при общем тестировании ,т .к это взаимосвязи.

(Проводится, когда модули объединяются в более крупные структурные элементы. Каждая система имеет интерфейс для взаимодействия с другими объектами системы. Даже если модули написаны верно, могут возникнуть ошибки взаимодействия.

Ошибки интерфейсов, возникают, как правило, неправильным понимаем того, то делают модули. Тестирование интерфейсов очень важно в тестировании объектно-ориентированных программ.)

Виды интерфейсов

Параметрические интерфейсы – ссылки на данные/функции передаются в виде параметров от одного компонента к другому.

Интерфейс разделямый памяти – блок памяти совместно используется разными подсистемами

Процедурные интерфейсы – одна система содержит(инкапсулирует) в себе набор процедур,вызываемые другими подсистемами.

Интерфейсы передачи сообщений – одна подсистема, запрашивая сервис у другой, передавая ей сообщение и получая сообщение с результатом. Для асинхронных систем и распределённых, где на первом месте надёжность сообщений.

Классификация ошибок интерфейсов

1.Неправильное использование интерфейсов – один компонент вызывает другой компонент и совершает ошибку. Такая ошибка чаще всего при параметрических интерфейсах (параметры передаются не в том порядке, не то количество)

2.Неправильное понимание интерфейсов – вызывающий компонент ошибочно интерпретирует интерфейс вызываемого компонента. Вызывающий компонент, в котором заложена неправильная интерпретация интерфейса вызываемого компонента, предполагает определенное поведение этого компонента. Пример: мы ожидаем, что функция вернёт значение одной переменной, а она возвращает значение другой.

3.Ошибки синхронизации. В системах реального времени используют интерфейсы разделяемой памяти и интерфейсы передачи сообщений.

В многопоточных системах.

Проблема при тестировании интерфейсов — ошибки интерфейсов проявляются в необычных условиях работы программы, которые трудно моделировать на этапе тестирования.

Ещё одна проблема — при тестировании интерфейсов ошибки могут обусловлены взаимодействием ошибок в нескольких модулях.

Общий порядок тестирования интерфейсов следующий:

1) Посмотреть тестируемый код и составить список вызовов, направленных к внешним компонентам.

2) Предусмотреть такие наборы тестовых данных, при которых параметры принимают свои предельные значения.

3) Если посредством интерфейса передаются указатели на данные, необходимо его протестировать с нулевыми указателями.

4) Если тестируется компонент с процедурным интерфейсом, который вызывает сбой в работе компонента.

5) В интерфейсах передачи сообщений следует использовать нагрузочное тестирование, то есть подавать тестовые данные со скоростью, превышающей предельно допустимую.

6) При тестировании интерфейсов памяти необходим менять время активации взаимодействующих компонентов, чтобы видеть неправильные допущения, которые были использованы при проектировании.

Для тестирования интерфейсов эффективным является статическое тестирование. В языках со строгим контролем типов многие ошибки интерфейсов выявляются ещё на этапе компиляции.

Для языков со слабым контролем типов (Си) необходимо применять специальные статические анализаторы. Кроме того, для проверки интерфейсов целесообразно проводить инспектирование программы (code-review).

Нагрузочное тестирование

После того, как завершено модульное тестирование всех модулей, выполнена сборка и интеграционное тестирование, можно говорить о том, что с функциональной точки зрения программа работает правильно.

Однако, кроме функциональных требований к программному продукту, предъявляются ещё и нефункциональные требования:

— надёжность;

— скорость работы;

Чтобы проанализировать производительность программного продукта, необходимо выполнить особый вид тестирования, который заключается в том, что программа запускается и на вход подаются данные сначала с предельно

расчётной интенсивностью, а затем со всё увеличивающейся, до тех пор, пока не произойдёт сбой программы.

Такое тестирование — и есть нагрузочное тестирование.

У нагрузочного тестирования есть две основных функции/две задачи:

1) Протестировать поведение системы во время сбоя. Сбой в системе не должен приводить к нарушению целостности данных и не должно происходить никаких аппаратных проблем.

2) Выявить дефекты, которые не проявляются в обычных режимах работы системы.

Тестирование ОО-систем Основные уровни тестирования:

1. Тестирование отдельных методов (как чёрный так и белый ящик)

2. Тестирование отдельных классов (класс – чёрный ящик)

3. Тестирование кластеров объектов – тестирование взаимосвязанных групп объектов (замена нисходящего или восходящего тестирования)

4. Тестирование всей системы в целом

Методики тестирования ОО программ:

1. Тестирование сценариев и вариантов использования – наиболее эффективное

2. Тестирование потоков (событий или данных)

3. Тестирование взаимодействий между объектами (интеграционное тестирование для ОО)

Системное тестирование

Проводится после полной сборки ПП. Отличается от всех видов тестирование:

1. Рассматривает систему в целом

2. Тестирование основывается на пользовательском интерфейсе или API (внешние интерфейсы программы)

3. Задача тестирования – проверить работоспособность системы

4. Выявление аномалии: неправильное использование ресурсов системы, несовместимость системы с окружением, непредусмотренные сценарии использования ПП, неудобства использования и т.д.

5. Система рассматривается только как чёрный ящик

Критерии тестов системного тестирования:

1. Полнота решения функциональных задач

2. Стрессовое тестирование – на предельных объемах

3. Тестирование корректности использования ресурсов (утечка памяти, возврат ресурсов после использования)

4. Оценка производительности (скорости работы)

5. Тестируется эффективность защиты от искажённых данных и некорректных действий

6. Тестируется инсталляция и конфигурация на разных платформах

7. Проверяется корректность документации

Системное тестирование выполняется либо вручную, либо с помощью систем для автоматизации тестирования на основе интерфейса пользователя.

Регрессионное тестирование

Написали программу собрали протестировали – работает → У заказчика новые требования →Выполнили их→ Вопрос :Тестирование новой версии ?Тестировать всю систему или изменённый модуль→Выполняется после изменений в системе

Проблема: как найти компромисс между полным повтором всех тестов или тестирование только изменившихся модулей

09.12.2020

Верификация и аттестация

– проверяется соответствие программы спецификации требований и ожиданий заказчика. Охватывают весь жизненный цикл программного продукта

Верификация-правильно ли создан? Проверка программы на соответствие Прог.требований

Аттестация – правильно ли работает ПП?

Два основных подхода для верификации аттестации:

1. Инспектирование программного продукта;
2. Тестирование программного продукта.

Инспектирование ПО

– анализ и проверка система – статически-выполняется на разный этапах-инспектирование документации- не превышает 2х часов.

Инспектирование предполагает проверку и анализ различных представлений системы на всех этапах работы над проектом. Это процесс, в ходе которого инспектор или же группа инспекторов просматривает систему с целью обнаружения ошибок и недоработок.

Методы инспектирования:

1. Инспектирование и проверка программ вручную
2. Автоматическая проверка
3. Метод формальной верификации

Тестирование ПО

– контролируемое выполнение ПП-динамический способ проверки ПП-применяется к программе, которая может быть выполнена.

Все методы тестирования можно разделить на две группы: это тестирование дефектов, статическое тестирование. Статическому тестированию подвергается правильно работящая программа, в которой нет дефектов, цель статического тестирования — анализ некоторых статических показателей системы, таких как производительность, надёжность, работа в различных режимах и так далее.

Инспектирование. Может выполняться на всех этапах жизненного цикла программного продукта. В том смысле, что инспектированию может подвергаться сама программа, так и документация — диаграмма и так далее. Инспектировать можно двумя основными способами: собственное инспектирование с привлечение группы экспертов-программистов; второй способ — это автоматический анализ.

Инспектирование программного обеспечения

Инспектирование применяется из-за ограниченных возможностей тестирования. Так как для тестирования программа должна быть рабочей и как минимум компилироваться, инспектирование позволяет проверять ещё не законченный продукт. Инспектирование позволяет обнаруживать целые семейства взаимосвязанных ошибок, потенциальные ошибки и остальные неприятные моменты. При тестировании ошибка в одном модуле может привести к неправильной работе другого модуля, разрушению данных, проще говоря — тестирование ограничено для выявления сложных ошибок. Инспектирование дешевле (если под тестированием понимать стоимость написания тестов). Статистика говорит, что более 60% ошибок в программе

может быть выявлена с помощью инспектирования. При инспектировании с использованием математических методов (формализовать), то процент выявляемых ошибок можно довести до 90%.

Организация процесса инспектирования

Процесс инспектирования — это формализованный процесс, в том смысле, что есть разбивка на определённые участки и этапы. Классическая модель предполагает следующие роли в команде инспекторов: автор, рецензент, инспектор и координатор.

Для выполнения инспектирования должно выполняться следующие условие: должна быть точная спецификация кода, так как без неё мы не будем знать, что нам нужно инспектировать. Второе требование: при инспектировании представляется синтаксически корректная версия программного кода. Участники инспекционной группы должны знать стандарты написания кода, которые приняты в данной команде.

Этапы инспектирования:

1) Планирование.
2) Этап предварительного просмотра.
3) Индивидуальная подготовка.
4) Собрание инспекционной группы.
5) Исправление ошибок.
6) Доработка.

Координатор составляет план инспектирование, ***подбирает*** участников инспекционной группы.

Координатор организует собрание, контролирует чтобы спецификация и синтаксически верная программа были представлены.

Программа, предоставленная на инспектирование, передаётся на рассмотрение ***инспекционной группы***.

На этапе предварительного просмотра ***автор описывает*** назначение программы.

На этапе индивидуальной подготовки, ***каждый из участников*** инспекционной группы изучает программу и её спецификацию и ***выявляет дефекты.***

Затем проводится ***собрание инспекционной группы***. Это собрание должно быть по возможности коротким — не более двух часов. На этом этапе основное внимание обращается на обнаружение дефектов, аномалий и других отклонений от спецификации. Инспекционная группа не должна предлагать способов устранения дефектов.

После этого (на пятом этапе), ***автор программы*** изменяет и модифицирует программу, исправляя обнаруженные ошибки.

На последнем (шестом) координатор решает, нужно ли проводить инспектирование повторное, либо же хватит уже. Если принято решение, что повторно инспектировать не нужно, то все найденные ранее дефекты документально фиксируются, составленный документ предоставляется руководителям проекта.

На этапе предварительного просмотра — за один час просматривается около 500 операторов программы.

На этапе индивидуальной подготовки проверяется около 125 операторов, так как проверка выполняется более детально.

На собрании инспекционной группы проверяется от 90 до 125 операторов. Эти все цифры условны, всё зависит от многих факторов — классификации продукта, сложности, стиля и так далее.

Статический анализ программы

Выполняется с помощью специализированных программных инструментов. Анализатор анализирует код программы и выявляет потенциальные ошибки. Обычно выполняется перед компиляцией. Статический анализатор — программный инструмент, предназначенный для выявления ошибок, выполнение программы при этом не требуется. Цель автоматического статического анализа

— привлечь внимание разработчика к аномалиям программы.

Статический анализ включает в себя следующие этапы:

1) Анализ потока управления. На этом этапе определяются циклы в программы, точки входа и выхода, обнаруживается по возможность неиспользуемый программный код.

2) Анализ использования данных. Здесь анализируется как используются переменные в программы, определить переменные, которые используются без инициализации, обнаруживаются переменные, значения которых нигде не используются. Здесь же выявляются условные операторы с избыточными условиями и так далее.

3) Анализ интерфейсов. На этом этапе анализируется согласованность отдельных модулей программы, проверяется правильность написания процедур. Проверяется также правильность вызова процедур. Здесь же могут обнаруживаться функции, которые нигде и никогда не вызываются, либо же функции, результаты которых нигде и никогда не используются.

4) Анализ потоков данных. Определяются зависимости между выходными данными (переменными). Ошибок такой не выявляет, но помогает понять, как работает программный модуль. Помогает понять, когда на вход поступают верные данные.

5) Анализ ветвей программы. Выполняется сематический анализ программы, определяются все ветки программы и операторы, выполняющееся в каждой ветке.

Инспектирование дешевле

Автор кода, рецензент –озвучивает код, инспектор – проверяет,координатор – отвечает за организацию процесса

ДЗ

Алгоритмы поиска:

1. B-деревья
2. AVL-деревья
3. Бинарные

Семафор

Mutex

Показатели качества работы программного продукта

Программный продукт можно описать некоторыми количественными характеристиками.

Показатели можно разделить на контрольные и прогнозируемые

- Контрольные показатели обычно связаны с процессом разработки, а прогнозируемые характеризуют конечный программный продукт. Пример контрольного показателя: среднее время на исправление ошибки, длительность разработки.

- Прогнозируемые показатели — цикломатическая сложность модулей, средняя длина идентификаторов в программе и так далее.

Основная сложность, связанная с определением основных внешних показателей — нельзя измерить непосредственно. Эти показатели — внешние показатели. Применяется выход: измеряются внутренние показатели (цикломатическое число и так далее), на их основе измеряются внешние показатели.

Показатели бывают динамические и статические

- Динамические показатели — показатели, которые могут быть определены только результатам выполняя программы. Расход памяти сложно определить без выполнения программы и так далее.

- Статические показатели — определяются статическим представлением системы, они характеризуются структуры программы, документацию, для этого выполнение программы не требуется.

Главным образом именно динамические показатели характеризируют качество программного продукта.

https://jugsev.ru/events.html#10zagalskii

перехода от одних формул к другим по строгим правилам, которые позволяют свести процедуру перехода от формулы к формуле к последовательности текстовых подстановок.

Преимущество формального подхода заключается в том, что с его помощью удается избегать обращений к бесконечной области значений и на каждом шаге доказательства оперировать только конечным множеством символов.

2) Интерпретационный подход применяется, когда осуществляется подстановка констант в формулы, а затем интерпретация формул как осмысленных утверждений в элементах множеств конкретных значений. Истинность интерпретируемых формул проверяется на конечных множествах возможных значений. Сложность подхода состоит в том, что на конечных множествах комбинации возможных значений для реализации исчерпывающей проверки могут оказаться достаточно велики.

Именно интерпретационный подход используется при экспериментальной проверке соответствия программы своей спецификации

Применение интерпретационного подхода в форме экспериментов над исполняемой программой составляет суть отладки и тестирования.

Основные понятия

Отладка (debug, debugging)– процесс поиска, локализации и исправления ошибок в программе [IEEE Std.610-12.1990].

Термин "отладка" в отечественной литературе используется двояко: для обозначения активности по поиску ошибок (собственно тестирование), по нахождению причин их появления и исправлению, или активности по локализации и исправлению ошибок.

Тестирование обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок).

Как правило, на фазе тестирования осуществляется и исправление идентифицированных ошибок, включающее локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы тестируемого приложения (Application Under Testing (AUT) или Implementation Under Testing (IUT)).

Если программа не содержит синтаксических ошибок (прошла трансляцию) и может быть выполнена на компьютере, она обязательно вычисляет какую-либо функцию, осуществляющую отображение входных данных в выходные. Это означает, что компьютер на своих ресурсах доопределяет частично определенную программой функцию до тотальной определенности. Следовательно, судить о правильности или неправильности результатов выполнения программы можно, только сравнивая спецификацию желаемой функции с результатами ее

Если программа не содержит синтаксических ошибок (прошла трансляцию) и может быть выполнена на компьютере, она обязательно вычисляет какую-либо функцию, осуществляющую отображение входных данных в выходные. Это означает, что компьютер на своих ресурсах доопределяет частично определенную программой функцию до тотальной определенности. Следовательно, судить о правильности или неправильности результатов выполнения программы можно, только сравнивая спецификацию желаемой функции с результатами ее вычисления, что и осуществляется в процессе тестирования.

Судить о правильности или неправильности результатов выполнения программы можно только сравнивая спецификацию желаемой функции с результатами ее вычисления.

Тестирование разделяют на статическое и динамическое:

Статическое тестирование выявляет формальными методами анализа без выполнения тестируемой программы неверные конструкции или неверные отношения объектов программы (ошибки формального задания) с помощью специальных инструментов контроля кода – CodeChecker.

Динамическое тестирование (собственно тестирование) осуществляет выявление ошибок только на выполняющейся программе с помощью специальных инструментов автоматизации тестирования – Testbed или Testbench.

Тестирование обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок).

Как правило, на фазе тестирования осуществляется и исправление идентифицированных ошибок, включающее локализацию ошибок, нахождение причин ошибок и соответствующую корректировку программы тестируемого приложения (Application Under Testing (AUT) или Implementation Under Testing (IUT)).

Если программа не содержит синтаксических ошибок (прошла трансляцию) и может быть выполнена на компьютере, она обязательно вычисляет какую-либо **функцию**, осуществляющую **отображение входных данных в выходные**. Это означает, что компьютер на своих ресурсах доопределяет частично определенную программой функцию до тотальной определенности. Следовательно, судить о правильности или неправильности результатов выполнения программы можно, только сравнивая спецификацию желаемой функции с результатами ее вычисления, что и осуществляется в процессе тестирования.

Оракул дает заключение о факте появления неправильной пары $(x,y_в)$ и ничего не говорит о том, каким образом она была вычислена или каков правильный алгоритм – он только сравнивает вычисленные и желаемые результаты. Оракулом может быть даже Заказчик или программист, производящий соответствующие вычисления в уме, поскольку Оракулу нужен какой-либо альтернативный способ получения функции (X,Y) для вычисления эталонных значений Y

В процессе тестирования Оракул последовательно получает элементы множества (X,Y) и соответствующие им результаты вычислений $(X,Y_в)$ для идентификации фактов несовпадений (test incident).

При выявлении $(x,y_в)$, не принадлежащего (X,Y), запускается процедура исправления ошибки, которая заключается во внимательном анализе (просмотре) протокола промежуточных вычислений, приведших к $(x,y_в)$, с помощью следующих методов:

1."Выполнение программы в уме" (deskchecking).

2.Вставка операторов протоколирования (печати) промежуточных результатов (logging).

3.Пошаговое выполнение программы (single-step running).

При пошаговом выполнении программы код выполняется строчка за строчкой. Используются следующие команды пошагового выполнения:

- Step Into – если выполняемая строчка кода содержит вызов функции, процедуры или метода, то происходит вызов, и программа останавливается на

- Step Over – если выполняемая строчка кода содержит вызов функции, процедуры или метода, то происходит вызов и выполнение всей функции и программа останавливается на первой строчке после вызываемой функции.

- Step Out – предназначена для выхода из функции в вызывающую функцию. Эта команда продолжит выполнение функции и остановит выполнение на первой строчке после вызываемой функции.

Пошаговое выполнение до сих пор является мощным методом автономного тестирования и отладки небольших программ.

4.Выполнение с заказанными остановками (breakpoints), анализом трасс (traces) или состояний памяти – дампов (dump).

Пример выполнения программы с заказанными контрольными точками и анализом трасс и дампов

- Контрольная точка (breakpoint) – точка программы, которая при ее достижении посылает отладчику сигнал. По этому сигналу либо временно приостанавливается выполнение отлаживаемой программы, либо запускается программа "агент", фиксирующая состояние заранее определенных переменных или областей в данный момент.

- Когда выполнение в контрольной точке приостанавливается, отлаживаемая программа переходит в режим останова (break mode). Вход в режим останова не прерывает и не заканчивает выполнение программы и позволяет анализировать состояние отдельных переменных или структур данных. Возврат из режима break mode в режим выполнения может произойти в любой момент по желанию пользователя.

- Когда в контрольной точке вызывается программа "агент", она тоже приостанавливает выполнение отлаживаемой программы, но только на время, необходимое для фиксации состояния выбранных переменных или структур данных в специальном электронном журнале - Log-файле, после чего происходит автоматический возврат в режим исполнения.

который осуществляется после выполнения трассы в режиме off-line, состояния дампа структурируются, и выделенные области или поля сравниваются с состояниями, предусмотренными спецификацией. Например, при моделировании поведения управляющих программ контроллеров в виде дампа фиксируются области общих и специальных регистров, или целые области оперативной памяти, состояния которой определяет алгоритм управления внешней средой.

5.реверсивное (обратное) выполнение (reversible execution)

Обратное выполнение программы возможно при условии сохранения на

каждом шаге программы всех значений переменных или состояний программы для соответствующей трассы. Тогда поднимаясь от конечной точки трассы к любой другой, можно по шагам произвести вычисления состояний, двигаясь от следствия к причине, от состояний на выходе преобразователя данных к состояниям на его входе. Естественно, такие возможности мы получаем в режиме off-line анализа при фиксации в Log – файле всей истории выполнения трассы.

Три фазы тестирования

Реализация тестирования разделяется на три этапа:

- Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment).
- Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver [IEEE Std 829-1983]) с получением протокола результатов тестирования (test log).
- Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

Основная проблема тестирования – определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.

Можно выделить следующие проблемы тестирования:

Реализация тестирования разделяется на три этапа:
- Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment).
- Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver [IEEE Std 829-1983]) с получением протокола результатов тестирования (test log).
- Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.
Основная проблема тестирования – определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.
Можно выделить следующие проблемы тестирования:

- Тестирование программы на всех входных значениях невозможно.
- Невозможно тестирование и на всех путях.
- Следовательно, надо отбирать конечный набор тестов, позволяющий проверить программу на основе наших интуитивных представлений

Требование к тестам: программа на любом из них должна останавливаться,

Требование к тестам: программа на любом из них должна останавливаться, т.е. не зацикливаться. Можно ли заранее гарантировать останов на любом тесте?

- В теории алгоритмов доказано, что не существует общего метода для решения этого вопроса, а также вопроса, достигнет ли программа на данном тесте заранее фиксированного оператора.

Задача о выборе конечного набора тестов(X,Y) для проверки программы в общем случае неразрешима.

Поэтому для решения практических задач остается искать частные случаи решения этой задачи.

Требования к идеальному критерию тестирования

1.Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы.

2.Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку.

3.Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы

4.Критерий должен быть легко проверяемым, например вычисляемым на тестах

Для нетривиальных классов программ в общем случае не существует полного и надежного критерия, зависящего от программ или спецификаций.

Поэтому мы стремимся к идеальному общему критерию через реальные частные.

Классы критериев

1.Структурные критерии используют информацию о структуре программы (критерии так называемого "белого ящика").

2.Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого "черного ящика").

3.Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической гипотезы.

4.Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.