Лекция 6

Тестирование

Любая программная система перед введением в эксплуатацию должна подвергаться проверке или инспектированию.

Основные формы инспектирование:

- Визуальный контроль. Проверка программы вручную без использования компьютера.
    - Соответствие комментариев тексту
    - Условные операторы
    - Сложные логические выражения
    - Завершенность цикла
- Статический контроль. Проверка программы по тексту с помощью инструментальных средств. Выделяют 4 формы статического контроля:
    - Синтаксический контроль. Выполняет компилятор.
    - Проверка структурированности. Выполняет линковщик, определяет количество модулей и связи
    - Контроль правдоподобия программы. Выделяются синтаксически правильные конструкции, которые могут нести конструкцию.
    - Верификация. Аналитическое доказательство правильности функционирования системы. Определяется модель системы и возможные спецификации. При формальной верификации проверяется ее надежность и корректность. Надежность характеризует как саму программу, так и ее окружение. Допускается определенная вероятность наличия ошибок в программе.
        - Целостность программы: Возможность защиты от отказа
        - Живучесть: способность к входному контролю данных и проверке их во время работы
        - Во время сбоев возможность восстановить состояние

      Частичная корректность и полная корректность.
- Динамический контроль. Заключается в проверке правильности По на компьютере. Чаще всего это называется тестированием.

Основная цель тестирования: доказать, что программа не работает.

При написании текстов необходимо придерживаться принципам Майерса:

- Частью каждого теста должно являться описание ожидаемых результатов работы программы
- Программа не может тестироваться его автором
- Результаты каждого текста должны проверяться. Тексты должны быть грамотно подобраны и многократно использоваться
- Принцип скопления ошибок: вероятность нахождения ошибки в том месте где она была определена прямо пропорционально числу обнаруженных ошибок.

Международный стандарт регламентирует следующий набор ошибок:

- Ошибка. Состояние программы при котором выдаются неправильные результаты, причинами которых является неверная технология обработки данных

- Дефект. Следствие ошибок разработчика, которые могут содержаться в исходных или проектных спецификациях, приводящих к сбою в работе программы. Сбой может трактоваться как отказ или восстанавливаемый сбой.
    - Отказ – отклонение программы от функционирования с невозможностью дальнейшей выполнением функций
    - Сбой – восстанавливаемый отказ, когда после неправильного функционирования программа может вернуться к предыдущему состоянию

Виды тестирования:

- Модульное тестирование. Когда берется отдельный класс или функция, а затем проверяется возможность использования этого компонента.
- Интеграционное тестирование. Проверяется наличие проблем в интерфейсах и в способах взаимодействия, интегрированных в компонент.
- Системное тестирование. Тестируется система на соответствие исходным требованиям
    - Альфа тестирование. Предполагает имитацию реальной работы системы без реального ее выполнения. Выполняется разработчиком в качестве внутреннего приемочного тестирования. В большинстве случаев выполняется под отладчиком для обнаружения ошибок.
    - Бета тестирование. Предполагает распространение версии с ограничением для работы заказчику, выявляются ошибки связанные с восприятием системы, некорректные обработки данных. Цель – получение обратной связи от пользователей программного обеспечения

Любое тестирование обеспечивает обнаружение ошибок, демонстрацию соответствия функции программы и ее назначения, демонстрация реализации требований к характеристикам программ. Тестирование не предназначено для доказательства отсутствия дефектов в программе.

Функциональное тестирование

Функциональное тестирование – тестирование по методу черного ящика, которая базируется на том, что для проверки работоспособности системы используются тесты, основанные на спецификации разрабатываемого ПО. Поведение тестируемой системы определяется путем изучения соответствия входным данным выходным. Методология тестирования черного ящика предполагает, что тестировщик имеет доступ только к внешним элементам, без доступа к внутренним структурам.

Для выполнения функционального тестирования используется 4 метода:

- Эквивалентное разбиение. Выбирается подмножество программы, которое тестируется. Позволяет создавать тесты, включающие в себя подмножество входных данных, обладающих свойствами:
    - Каждый элемент подмножества уменьшает число тестов, которые должны быть разработаны для приемлимого тестирования.
    - Каждый элемент подмножества должен покрывать значительную часть других тестов, подтверждая наличие или отсутствие ошибок в данной функции программы.
    - Каждый тест должен влючать в себя столько условий, сколько необходимо для минимизации общего числа тестов. Входная область данных должна разбиваться на

конечное число классов эквивалентности, при этом тест написанный для одного представителя данного класса должен гарантировать соответствующую обработку другого представителя класса.

Виды данных:

- Корректные. Данные, которые обязана обрабатывать программа.
- Не корректные. Данные, которые не должна обрабатывать данные.

Разработка тестов делится на 2 этапа:

- Выделение классов эквивалентности.
  - Если входное условие описывает некую область значения, то определяется правильный класс эквивалентности и 2 неправильных
  - …
- Построение тестов.

- Анализ граничных значений. Основан на методе эквивалентных разбиений, но применение обусловлено тем, что большинство ошибок возникают на границе. Основные отличия в том что тестовый вариант создаются для проверки границ и для создания тестовых вариантов учитывается область ввода и вывода (допустимое отклонение). Правила анализа:
  - Если условие ввода задается диапазоном, то строятся тестовые наборы для границ диапазона, и чуть ниже и чуть выше.
  - Если задается дискретное множество, то задаются тесты для проверки максимального из значений и минимальное. И значения чуть меньше минимума и максимума
  - Если структуры данных имеют границы, то тесты проверяют все границы для структуры. Если данные – упорядоченное множество, то проверятся первое и последнее значение множества.
- Функциональные диаграммы. Позволяет решит недостаток анализа граничных значений связанных с возможностью. Позволяет обнаружить неполноту и неоднозначность исходных спецификаций. Сама фукнциональная диаграмма представляет собой формальный язык на оторый транслируется спецификация написанная на естественном языке. На основании входных и ожидаемых результатов строятся тесты. Шаги:
  - Определение причины и следствия
  - Присвоение идентификатора
  - Граф причинно-следственных связей
  - Таблица решений из графа
  - Столбцы в тестовый набор
- Метод предположения об ошибке.

Структурное тестирование

Базируется на анализе логике ПО и подбирается набор данных, позволяющий протестировать все маршруты функционирования системы. Структурный подход выполняется:

- Метод покрытия операторов
- Метод покрытия решений
- Покрытия условий
- Комбинация методов

Недостатки:

- Не обнаруживает пропущенных маршрутов
- Не обнаруживает ошибок функционала
- Не дается гарантия, что выполняются функциональные требования

Управляющий граф программы – граф отображающий поток управления программы.

Каждый из методов предполагает соблюдение критерия. Критерий покрытия оператора подразумевает, что необходимо создать набор тестов, чтобы каждый оператор программы выполнялся по крайней мере только один раз. Значит, нужно такие маршруты, которые проходили через все вершины

Критерий покрытия решений. Для реализации данного критерия необходимо такое количество и состав теста, чтобы в результате было проверено каждое решение т.е. условная вершина принимала значение истина или ложь. Покрытие решения заключается в том, что необходимо подобрать тесты, чтобы каждая дуга содержалась по крайней мере в одном из пути тестирования.

Необходимо задать такое количество тестов, что результат каждого условия, внутри каждого решения принимали значения истина или ложь хотя бы 1 раз. Данный критерий дополняется предположением, что в каждой точке входа управление должно быть передано хотя бы 1 раз.

Комбинаторное покрытие условий предполагает выполнение критерия при котором формируется множество тестов в котором каждая комбинация условий выполняется хотя бы 1 раз.

Построение наборов тестов

Состоит из этапов:

- Конструирование управляющего графа
- Выбор тестовых путей
    - Статические методы
        - Эвристические методы
        - Адаптивные методы
    - Динамические методы
    - Методы путей
- Генерация тестов, соотвествующих тестовым путям

Статические методы предполагают, что каждый тестовый путь строится посредством постеменного удлинения пути за счет добавления следующих дуг, которые являются выходными для вершины, удлинение выполняется до достижения конечной вершины графа.

Основным достоинством является использование сравнительно небольшое число ресурсов.

Недостатки: не рассматриваются не реализуемые пути, сложно обойти все пути.

Статические методы реализуются с помощью эвристик и адаптивно.

Эвристики позволяют строить каждый тестовый набор по аналогии с ранее проверенными программами. Таким образом сокращается число генерируемых тестов.

Адаптивные методы заключаются в том, что каждый раз во входной тест добавляется только один тестовый путь, причем при выборе следующего пути руководствуются индуктивной стратегией. Чаще всего адаптивные методы используются для критериев покрытия операторов. С практической точки зрения реализация каждой покрытой вершины позволяет определить достигаемость состояния той или иной системы.

Динамические методы

Позволяют построить систему тестов, которые удовлетворяют заданным критериям. При использовании динамических методов решается задача построения покрывающего множества путей с одновременным формированием набора тестовых данных. При этом возможно автоматическое учет реализуемых и нереализуемых путей. Основная идея заключается в присоединений начальным реализуемым отрезкам путей дальнейших маршрутов так, чтобы реализуемость вновь полученных путей покрывала ранее протестированные участки программы.

Недостаток: ресурсоемкая методика

Достаток: высокий уровень проверки и оценка реализуемых путей.

Метод реализуемых путей

Основан на том, что прокрывающее множество путей строится на базе всех реализуемых путей в системе. Нереализуемые не рассматриваются, могут быть исключены как возможности неисправностей программы.

Генерация тестов соответствующим тестовым путям

На данной фазе на каждом пути тестирования необходимо найти каждый тест реализующий данный путь. Каждый путь описывается коньюнкцией предикатов. Где C_l предикат. Данный предикат задает значение в данной вершине. Конечный предикат формируется в процессе обратной подстановки. Т.е. формируется набор неравенств, реализующих тот или иной тест, ожидаемое значение предиката соответствует проверяемому значению тестовому пути.

Место конкретизации ошибки выполняется путем обратной раскрутки. Где для каждого введенного предиката решения определяется возможность его выполнения. Степень тестирования программы является показателем качеством программы. На практике построить полный набор тестов невозможно, поэтому проектируются эффективные тесты из соображений, что тестирование должно содержать не более 30% времени разработки ПО. Даже если удалось провести полное тестирование, возможно дальнейшее выявление ошибок.

- Программа может не соотвествовать внешней спецификации
- Возможность возникновения ошибок, которые зависят от обрабатываемых данных

Функциональное + структурное.

Особое внимание уделяется тестирование многомодульных программных коммплексов. Тогда методика тестирования разворачивается в виде спирали. Каждый ее виток определяет тестирование определенного уровня структуры. Тестирование элементов имеет своей целью

индивидуальную проверку каждого закодированного модуля. На этом этапе используется методика структурного тестирования. Тестирование интеграций позволяет выявить ошибки связанные с этапом проектирования. Цель тестирования – определить правильность и целесообразность сборки модулей в программную систему. Тут применяется метод черного ящика, функциональные диаграммы

Тестирование правильности своей целью считает реализация программной системы с точки зрения ее поведения. Таким образом проверяются технические требования, которые выполнены на этапе анализа. Используются тут граничные значения

Системное тестирование проверяет правильность тестирование.

Тестирование многомодульного ПО

- Монолитное тестирование. Каждый модуль проверяется по отдельности, а потом целиком. Для вызова каждого модуля разрабатывается специальный модуль, который называется модулем драйвера. Модуль-драйвер моделирует вызов тестируемого модуля со стороны внешних процедур. Для тестирование механизма взаимодействий пишется набор заглушек, которые моделируют работу вызываемых модулей. Преимущество: возможность параллельного выполнения тестов с минимальными затратами машинного времени.
- Пошаговое. Каждый модуль подключается к оттестированной части программы. Основным достоинством: небольшая трудоемкость тестирования и ускоренная отладка за счет быстрого отслеживания ошибок.
    - Нисходящее. Тестирование начинается с главного модуля. Выбираются модули ввода-вывода, а потом к логике.
    - Восходящее. От не основных модулей, до глобальных. Основной модуль тестируется в конце.
    - Комбинированное. Модули верхних уровней тестируются нисходящим, а нижних восходящим.

Виды тестирования ПО

При тестировании системы выделяются различные направления тестирования:

- Функциональное. Проверка функции системы на требования
    - Функциональное
    - Безопасности
    - Взаимодействие с пользователем
- Не функциональное. Исходит из того, что все функции реализованы и необходимо определить качество реализованных функций. Предполагает тестирование производительности, сложности установки, удобства использования, тестирование на отказ, конфигурационное тестирование
- Тестирование производительности.
    - Нагрузочное. Автоматизированное тестирование, имитирующее работу определенного числа пользователей с общим разделяемым ресурсом. Общей задачи является определение возможности масштабирования системы под

нагрузкой, при этом замеряется время выполнение, количество пользователей, граница приемлемой производительности.
    - Стрессовое. Предполагает максимальной трудозатратной операции на предельной нагрузке. С определением момента деградации производительности.
    - Надежности. Соблюдение работоспособности под нагрузкой заданный период времени.
    - Объемное тестирование. Оценка производительности при увеличений объема данных. Измеряется время отклика и обработки.
- Тестирование установки. Успешность инсталяции и настройки По. Определяем необходимость обновления или удаления ранее установленного ПО.
- Тестирование удобства использования. Выявляется при бете.
- Тестирование на отказ и восстановление. Способность возврата к старому состоянию, бекапы.
- Конфигурационное тестирование. Вид тестирования направленный на проверку работы ПО в различных конфигурациях.
    - Проверка оптимальной конфигурации средств.
    - Совместимость конфигураций.
    - Проверка возможность реализации конфигураций.

Виды тестирования ПО

- Связанный с изменением
    - Дымовое тестирование. При вводе в эксплуатацию новой системы тестировалось удачность выполнения и тестирование на сбой. Короткий цикл тестов, который подтверждает работоспособность программ.
    - Регрессионное тестирование. Направленно на проверку изменений, сделанных в приложений или окружающей среде для подтверждения того факта, что существующий ранее функционал остается доступным.
        - Регрессия багов. Нет новых багов
        - Регрессия старых багов. Нет старых багов
        - Регрессия побочного эффекта. Попытка доказать, что изменения не лишило работоспособности другие части системы.
    - Тестирование сборки. Направленно на определение соответствии выпущенной сборки.

Критерии завершения

- Выполнены все методики проведения тестов и все тесты дают отрицательный результат.
- Когда количество выявленных ошибок не превышает допустимого значения.
- Когда количество выявленных тестов значительно снижается по сравнению с числом ошибок, которые выявленные ранее.
- Когда завершено время тестирования

Отладка

Процесс локализации и исправления ошибок, которые обнаружены при тестировании. Локализация – определение точки или операции выполнение которой вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо выявить ее причину. Причины могут быть как очевидными, так и скрытыми. Сложность отладки обусловлена следующими причинами:

1) В ряде случаев требуются глубокие знания специфики управления вычислительными средствами, ОС, средой, в которой реализуются процессы.
2) Одной из причин сложности отладки является психологический дискомфорт так как ошибки должны исправляться в ограниченное время
3) Возможно возобновление ошибок за счет пере размещения программных модулей по выделяемой оперативной памяти.
4) Отсутствуют четко сформулированные методики отладки.

Все ошибки выполнения делятся на:

- Ошибки определения данных. Ошибки, связанные с передачей. Ошибки преобразования. Ошибки перезаписи и неправильные данные.
- Логические ошибки.
    - Неправильное проектирование. Неверная парадигма, неверный алгоритм, неверные структуры
    - Кодирование. Неправильная работа с переменными, организация интерфейса, некорректная реализация алгоритмов и вычислений.
- Ошибки накопления погрешности. Вызваны ограниченной разрядной сеткой, либо игнорирование механизмов уменьшения погрешности.

Сложность отладки увеличивается в следствии влияния следующих факторов: возможность получения внешних появлений разных ошибок.

Методы отладки:

- Ручное тестирование
- Индукция. Анализ симптомов ошибок, которые могут проявляться как неверный результат, либо как сообщение об ошибках. Выявляется место возникновения ошибок, выдвигается гипотеза, почему эта ошибка появилась, если гипотеза не подтверждается, то делается новое предположение.
- Дедукция. Формируется набор причин, которые могли бы вызвать данное проявление ошибки, затем причины анализируют и исключают те, которые противоречат имеющимся данным. Если все причины исключены, то выполняется дополнительное тестирование, в противном случае выбирают наиболее вероятную гипотезу и пытаются ее доказать. Если она объясняет причины появления ошибки, то ошибка считается найденной, иначе проверяют следующую причину.
- Обратное прослеживание. Используется для небольших программ, где устанавливают точку получения неправильного результата. Для этой точки определяется набор основных характеристик. Исходя из полученной гипотезы делают предположение о месте неправильном получения характеристик, продолжается до тех пор, пока не удается получит причину ошибку. Если данных недостаточно, то используют дополнительные средства или отладочные средства. Так же могут использоваться независимые отладчики, которые могут выполнить любой фрагмент программы в пошаговом режиме и проверить значение переменных. Эти отладчики позволяют отлаживать ассемблерные программы.

Отладка программы

1) Изучение появления ошибки.
2) Локализация ошибки.
    a. Отсечение частей программы.
    b. Использование отладчиков.
3) Определение причины ошибки.
4) Исправление.
5) Повторное тестирование.