ЭЛЕКТРОННАЯ ТЕТРАДЬ

по МДК 01.02

Поддержка и тестирование программных модулей

№2

ФИ студента Елагин Никита

Номер группы 2-ИП-11-19

\***Примечание.** Данное электронное учебное пособие рекомендуется использовать только в электронном виде, так как в бумажном виде невозможно выполнить интерактивные задания.

**Тема: Виды ошибок. Методы отладки. Методы тестирования**

**Уровни и виды тестирования**

На основании материалов презентации и самостоятельно найденного материала заполните таблицу следующего вида:

|  |  |  |
| --- | --- | --- |
| Вид тестирования | Основная задача | Характерные особенности |
| Модульное тестирование (Unit-testing) | Проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится вызывая код, который необходимо проверить и при поддержке сред разработки | Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это **подготовка автоматизированных тестов** до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешно пройдены. |
| Интеграционное тестирование (Integration testing) | предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами). | Подходы к интеграционному тестированию:  * Снизу вверх (*Bottom Up Integration*)   Все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения   * Сверху вниз   Вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.   * Большой взрыв (*"Big Bang" Integration*)   Все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования |
| Системное тестирование (System testing) | Основной задачей системного тестирования является **проверка**[как функциональных, так и не функциональных требований](http://www.protesting.ru/testing/testtypes.html)**в системе в целом**. При этом выявляются [дефекты](http://www.protesting.ru/testing/bugreport.html), такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи. | Можно выделить два подхода к системному тестированию:   * на базе требований (requirements based)   Для каждого требования пишутся [тестовые случаи (test cases)](http://www.protesting.ru/testing/testcase.html), проверяющие выполнение данного требования.   * на базе случаев использования (use case based)   На основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно определить один или более сценариев. На проверку каждого сценария пишутся [тест кейсы (test cases)](http://www.protesting.ru/testing/testcase.html), которые должны быть протестированы. |
| Приемочное тестирование (Acceptance testing) | Формальный процесс тестирования, который проверяет соответствие системы требованиям | Приемочное тестирование выполняется на основании набора типичных [тестовых случаев](http://www.protesting.ru/testing/testcase.html) и сценариев, разработанных на основании требований к данному приложению.  Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения. |
| Статическое тестирование (Static testing) | тип тестирования, который предполагает, что программный код во время тестирования не будет выполняться. При этом само тестирование может быть как ручным, так и автоматизированным. | Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации. Для этого типа тестирования в некоторых случаях даже не нужен компьютер – например, при проверке требований.  Большинство статических техник могут быть использованы для «тестирования» любых форм документации, включая вычитку кода, инспекцию проектной документации, функциональной спецификации и требований.  Даже статическое тестирование может быть автоматизировано – например, можно использовать автоматические средства проверки синтаксиса программного кода. |
| Динамическое тестирование (Dynamic testing) | тип тестирования, который предполагает запуск программного кода. Таким образом, анализируется поведение программы во время ее работы. | Для выполнения динамического тестирования необходимо чтобы тестируемый программный код был *написан*,*скомпилирован*и*запущен*. При этом, может выполняться проверка внешних параметров работы программы: загрузка процессора, использование памяти, время отклика и т.д. – то есть, ее производительность.  Динамическое тестирование является частью процесса валидации программного обеспечения. |
| Альфа-тестирование | это этап отладки и проверки альфа-версии программы, а люди, которые будут заниматься ее тестированием — альфа-тестерами. Это могут быть штатные тестировщики компании или люди, которые работают по договору, но это квалифицированные специалисты, умеющие работать со специализированным программным обеспечением и пользоваться специальными методиками. | После того, как отдельные программные модули готовы, они объединяются в некое единое целое. Это еще не полнофункциональная программа, но она уже способна работать и выполнять, хотя бы частично, свои главные задачи. Такой вариант программы и называют **альфа-версией.** |
| Бета-тестирование |  |  |
| Регрессионное тестирование (Regression testing) | вид тестирования направленный на проверку изменений, сделанных в приложении или окружающей среде (починка дефекта, слияние кода, миграция на другую операционную систему, базу данных, веб сервер или сервер приложения), для подтверждения того факта, что существующая ранее функциональность работает как и прежде | Как правило, для регрессионного тестирования используются тест кейсы, написанные на ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Рекомендуется делать [автоматизацию регрессионных тестов](http://www.protesting.ru/automation/), для ускорения последующего процесса тестирования и обнаружения дефектов на ранних стадиях разработки программного обеспечения. |
| «Смок-тест» (Smoke Testing, «дымовое тестирование») | В области же программного обеспечения, дымовое тестирование рассматривается как короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода (нового или исправленного) устанавливаемое приложение, стартует и выполняет основные функции. | Вывод о работоспособности основных функций делается на основании результатов поверхностного тестирования наиболее важных модулей приложения на предмет возможности выполнения требуемых задач и наличия быстронаходимых критических и блокирующих дефектов. В случае отсутствия таковых дефектов дымовое тестирование объявляется пройденным, и приложение передается для проведения полного цикла тестирования, в противном случае, дымовое тестирование объявляется проваленным, и приложение уходит на доработку. |

Сравните модульное и интеграционное тестирование и запишите это сравнение в форме таблицы

|  |  |  |
| --- | --- | --- |
|  | Модульное тестирование | Интеграционное тестирование |
| Основные задачи |  |  |
| Особенности |  |  |
| Какую модель тестирования используют |  |  |
| Какие дефекты обнаруживает |  |  |

Особенности системного тестирования

1. Задача (какие дефекты помогает выявить)

2. Какую модель тестирования использует

3. Какие категории тестов входят в системное тестирование

Заполните таблицу

|  |  |
| --- | --- |
| Вид тестирования | Сущность этого вида тестирования (для чего применяется) |
| Функциональное тестирование (Functional testing) |  |
| Тестирование производительности (Perfomance testing) |  |
| Стрессовое тестирование (Stress testing) |  |
| Нагрузочное тестирование (Load testing) |  |
| Тестирование стабильности (Stability testing) |  |
| Тестирование удобства использования (Usability testing) |  |
| Тестирование интерфейса пользователя (UI testing) |  |
| Тестирование безопасности (security testing) |  |
| Тестирование совместимости (Compatibility testing) |  |

Найдите материал для заполнения таблицы

|  |  |
| --- | --- |
| Вид ошибки | Пример |
| Неправильная постановка задачи |  |
| Неверный алгоритм |  |
| Ошибка анализа |  |
| Семантические ошибки |  |
| Синтаксические ошибки |  |
| Ошибки при выполнении операций |  |
| Ошибки в данных |  |
| Опечатки |  |
| Ошибки ввода-вывода |  |

Ошибки, которые не обнаруживает транслятор.

Приведите примеры ошибок, которые не может обнаружить транслятор

**Логические ошибки:**

* неверное указание ветви алгоритма после проверки некоторого условия;
* неполный учет возможных условий;
* пропуск в программе одного или более блоков алгоритма.

**Ошибки в циклах:**

* неправильное указание начала цикла;
* неправильное указание условий окончания цикла;
* неправильное указание числа повторений цикла;
* бесконечный цикл.

**Ошибки ввода-вывода; ошибки при работе с данными:**

* неправильное задание тип данных;
* организация считывания меньшего или большего объёма даных, чем требуется;
* неправильное редактирование данных.

**Ошибки в использовании переменных:**

* использование переменных без указания их начальных значений;
* ошибочное указание одной переменной вместо другой.

**Ошибки при работе с массивами:**

* массивы предварительно не обнулены;
* массивы неправильно описаны;
* индексы следуют в неправильном порядке.

**Ошибки в арифметических операциях:**

* неверное указание типа переменной (например, целочисленного вместо вещественного);
* неверное определение порядка действий;
* деление на нуль;
* извлечение квадратного корня из отрицательного числа;
* потеря значащих разрядов числа.

Все эти ошибки обнаруживаются с помощью тестирования.

Отладка программы — Это этап разработки компьютерных программ, когда нужно найти и исправить ошибки в коде.

Тестирование программы — это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.