Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Филиал федерального государственного бюджетного образовательного учреждения высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.» в г. Петровске

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

по дисциплине

МДК. 01.02 Поддержка и тестирование программных модулей

направление подготовки

09.02.07«Информационные системы и программирование»

Петровск 2019

**Пояснительная записка**

Данные методические указания для студентов по выполнению практических работ согласно программе дисциплины МДК 01.02 «Поддержка и тестирование программных модулей» предназначены для реализации государственных требований к минимуму содержания и уровню подготовки выпускников специальности 09.02.07 «Информационные системы и программирование» с целью закрепления теоретических знаний и практических умений.

**Требования к знаниям и умениям при выполнении практических работ**

С целью овладения указанным видом профессиональной деятельности и соответствующими профессиональными компетенциями обучающийся в ходе

освоения профессионального модуля должен:

**иметь практический опыт:**

- разработки алгоритма поставленной задачи и реализации его средствами автоматизированного проектирования;

- разработки кода программного продукта на основе готовой спецификации на уровне модуля;

- использования инструментальных средств на этапе отладки программного продукта;

- проведения тестирования программного модуля по определенному

сценарию;

**уметь:**

- осуществлять разработку кода программного модуля на современных языках программирования;

- создавать программу по разработанному алгоритму как отдельный модуль;

- выполнять отладку и тестирование программы на уровне модуля;

- оформлять документацию на программные средства;

- использовать инструментальные средства для автоматизации оформления документации;

**знать:**

- основные этапы разработки программного обеспечения;

- основные принципы технологии структурного и объектно-ориентированного программирования;

- основные принципы отладки и тестирования программных продуктов;

- методы и средства разработки технической документации.

**Перечень лабораторных работ**

**МДК 01.02Поддержка и тестирование программных модулей**

**Перечень практических работ**

**МДК.01.02 Поддержка и тестирование программных модулей**

ПРАКТИЧЕСКАЯ РАБОТА №1

Тема:Тестирование «белым ящиком»

ПРАКТИЧЕСКАЯ РАБОТА №2

Тема:Тестирование «черным ящиком»

ПРАКТИЧЕСКАЯ РАБОТА №3

Тема:Модульное тестирование

ПРАКТИЧЕСКАЯ РАБОТА №4

Тема:Интеграционное тестирование

ПРАКТИЧЕСКАЯ РАБОТА №5

Тема:Оформление документации на программные средства с использованием инструментальных средств.

ПРАКТИЧЕСКАЯ РАБОТА №1,2

**Тема:** Тестирование «белым ящиком», Тестирование «черным ящиком»

* + - 1. **Введение**

Новые методологии программирования, таких как ускоренная разработка приложений (RAD), экстремальное программирование, привели к интенсивным разработкам средств автоматизированного тестирования. Главной особенностью этих методологий является возможность получения различных версий программного продукта всё возрастающего объёма с высокой частотой. Отсюда, современное тестирование приобрело итеративную природу - каждая новая версия сопровождается значительным количеством новых тестов, а так же переработкой существующих автоматизированных средств тестирования. Поскольку для каждой версии проекта необходимо разработать средство позволяющее наиболее полно протестировать все его особенности, актуальны методы и алгоритмы проектирования автоматизированных средств тестирования, с помощью которых можно быстро и с наименьшими затратами разработать подобное средство. Отличительные особенности проектирования автоматизированных средств тестирования вытекает из специфики задач, которые ставятся перед ними:

*- объективная трудность тестирования: это деструктивный (т.е. обратный созидательному) процесс. Поэтому, проектируемые средства должны вместо сбора и обработки информации выполнять её разбиение на части и проводить анализ этих частей,*

*- тестирование абсолютно всех возможных ситуаций бесконечно велико, поэтому проектируемые средства тестирования должны учитывать невозможность перебора всех возможных тестов.*

В свою очередь, при разработке программных продуктов, в целях уменьшения затрат, широко применяются методы и алгоритмы проектирования программных или информационных систем. Но не все они подходят для проектирования средств тестирования, кроме того в условиях быстрого и постоянного изменения тестируемого проекта приходится постоянно перепроектировать существующие средства. Интенсивная разработка и модернизация автоматизированных средств тестирования делает актуальной задачу поиска и применения наиболее оптимальных методов и алгоритмов проектирования, с помощью которых можно быстро и с минимальными затратами спроектировать новое средство тестирования.

1. **Тестирование программного обеспечения.**

Тестирование проводится с целью обеспечить качество разрабатываемого программного продукта. Стандарт ISO-8402, посвященный описанию систем обеспечения качества программного обеспечения, под качеством понимает "совокупность характеристик программного продукта, относящихся к его способности удовлетворять установленные и предполагаемые потребности клиента". Основным параметром качества программы является надёжность. Надёжность определяется как вероятность его работы без отказов в течении определённого периода времени, рассчитанная с учётом стоимости для пользователя каждого отказа. Отказ программного обеспечения - это проявление ошибки в нём. Отсюда тестирование ПО - это процесс выполнения программы с целью обнаружения в ней ошибок. "Удачным" тестом является такой, на котором выполнение программы завершилось с ошибкой. Напротив, "неудачным" называется тест, не позволивший выявить ошибку в программе. Основные принципы организации тестирования:

*1. необходимой частью каждого теста должно являться описание ожидаемых результатов работы программы;*

*2. программа не должна тестироваться её автором;*

*3. организация - разработчик программного обеспечения не должна "единолично " его тестировать;*

*4. необходимо подбирать тесты не только для правильных (предусмотренных) входных данных, но и для неправильных (непредусмотренных);*

*5. при анализе результатов каждого теста необходимо проверять, не делает ли программа того, что она не должна делать;*

*6. "принцип скопления ошибок" - вероятность наличия не обнаруженных ошибок в некоторой части программы прямо пропорциональна числу ошибок, уже обнаруженных в этой части;*

*После исправления обнаруженных ошибок, тестер должен проверить: действительно ли они были устранены. После чего он должен провести полное тестирование программного продукта, так как в процессе устранения ошибок могли быть допущены новые.*

Процесс тестирования состоит из трёх этапов:

*1. Проектирование тестов.*

*2. Исполнение тестов.*

*3. Анализ полученных результатов.*

На первом этапе решается вопрос о выборе некоторого подмножества множества тестов, которое сможет найти наибольшее количество ошибок за наименьший промежуток времени. На этапе исполнения тестов проводят, запуск тестов и отлавливают ошибки в тестируемом программном продукте. Существует две методологии тестирования - "чёрного" и "белого" ящика.

*- "Чёрный ящик" - тестирование функционального поведения программы с точки зрения внешнего мира (текст программы не используется).*

*- "Белый ящик" - тестирование кода на предмет логики работы программы и корректности её работы с точки зрения компилятора того языка на котором она писалась.*

Полученные результаты тестирования позволяют сделать вывод о надёжности программного продукта. Они служат основой его сертификации и гарантией качества.

Чтобы облегчить и ускорить процесс тестирования широко применяют автоматизацию одного или ряда сложных этапов тестирования.

Автоматизированные средства разрабатываются в основном для следующих этапов процесса тестирования:   
*- тестирование функциональных требований,   
- тестирование пользовательского интерфейса,   
- тестирование отдельных модулей,   
- комплексное тестирование,  
- анализ сложности программных модулей,   
- тестирование покрытия программного кода,   
- тестирование скорости загрузки системы,  
- тестирование граничных условий,  
- тестирование утечки памяти.*

Существует два основных вида тестирования: функциональное и структурное.

**При функциональном тестировании** программа рассматривается как "черный ящик" (то есть ее текст не используется). Происходит проверка соответствия поведения программы ее внешней спецификации. Критерием полноты тестирования в этом случае является перебор всех возможных значений входных данных, что невыполнимо. Поскольку исчерпывающее функциональное тестирование невозможно, речь может идти о разработке методов, позволяющих подбирать тесты не "вслепую", а с большой вероятностью обнаружения ошибок в программе.

**При структурном тестировании** программа рассматривается как "белый ящик" (т.е. ее текст открыт для пользования). Происходит проверка логики программы. Полным тестированием в этом случае будет такое, которое приведет к перебору всех возможных путей на графе передач управления программы (ее управляющем графе). Если ограничиться перебором только линейных не зависимых путей, то и в этом случае исчерпывающее структурное тестирование практически невозможно, т. к. неясно, как подбирать тесты, чтобы обеспечить "покрытие" всех таких путей. Поэтому при структурном тестировании необходимо использовать другие критерии его полноты, позволяющие достаточно просто контролировать их выполнение, но не дающие гарантии полной проверки логики программы. Но даже если предположить, что удалось достичь полного структурного тестирования некоторой программы, в ней тем не менее могут содержаться ошибки, т.к.:

*1) программа может не соответствовать своей внешней спецификации, что в частности, может привести к тому, что в ее управляющем графе окажутся пропущенными некоторые необходимые пути;*

*2) не будут обнаружены ошибки, появление которых зависит от обрабатываемых данных (т.е. на одних исходных данных программа работает правильно, а на других - с ошибкой).*

Таким образом, ни структурное, ни функциональное тестирование не может быть исчерпывающим. Чтобы увеличить процент обнаружения ошибок при проведении функционального и структурного тестирования используют средства автоматизации тестирования.

1. **Организация тестирования программ**

Тестирование программного продукта одновременно проводится в 3-ёх направлениях:   
*1****. проверка кода (review):*** *Тестер просматривает исходный код визуально и пытается найти нём ошибки, а так же различные несоответствия кода и требований к нему. Под требованием понимается стандарт, которого придерживается разработчики данного проекта, реакция на те или иные действия со стороны среды воздействия на ПО, поведение программного продукта в различных ситуациях;*

*2.* ***тестирование высокого уровня:*** *Здесь главная цель тестирования - выяснить, удовлетворяет ли разработка всем требованиям заказчика. Для программного продукта пишутся эмуляторы, с помощью которых тестер может наблюдать за работой системы в роли оператора. Он видит, как система осуществляет диалог с пользователем, какие сообщения она выдаёт, как реагирует на различные события, сохраняет информацию и т.д. Большинство обнаруживаемых ошибок на этом этапе связанно с ошибками взаимодействия программного продукта с пользователем - вывод ошибочных сообщений, не правильная реакция на запрос от оператора и т.п.;*

*3****. тестирование низкого уровня:*** *Тестер проверяет, на сколько логически полно исходный код покрывает всё возможные варианты работы системы, для которой он разрабатывается*.

Существуют стандарты тестирования, они зависят от того, в какой области применяется разрабатываемое ПО. Ниже представлены некоторые из них:

**Стандарт ISO 9001**

ISO 9001 - стандарт, основанный на принципах контроля качества. В нём, по существу, задаются ключевые функциональные требования, для каждого из которых нужно сказать, что делается, как сделать то, что сказано, и иметь возможность показать, что было сделано. Реализация данного стандарта в среде ПО - ISO 9000-3.

**Стандарт ISO/IEC 12207 и IEEE/EIA 12207**

ISO/IEC 12207 - это международный стандарт, описывающий структуру процессов жизненного цикла ПО от концепции до изъятия из обращения. Стандарт IEEE/EIA 12207 - адаптация ISO/IEC 12207 для США.

В соответсвии с этими стандартами в той или иной отрасли производства выдвигаются требования к тестрованию ПО. Например в авиации США на основе ISO/IEC 12207 был выработан стандарт RTCA( Requirements and Technical Concepts for Aviation). В нём перечисленны следующие требования к тестированию верхнего и нуижнего уровня: Тестирование верхнего уровня:

- требования высокого уровня должны включать в себя системные требования к ПО;

- требования высокого уровня должны формулироваться с учётом архитектуры ПО;

- программный код должен удовлетворять архитектуре ПО и требованиям низкого уровня;

- откомпилированный и готовый к использованию код должен удовлетворять требованиям к ПО;

- используемые значения должны технически соответствовать поставленным целям и выполнять их для всех уровней ПО;

Тестирование нижнего уровня:

- проверку (Verification) требований нижнего уровня,

- проверку архитектуры программного обеспечения (ПО),

- проверку логического покрытия для всех функций написанных в ПО,

- контроль процедур тестирования,

- независимость ПО от тестирования. Т.е. ПО не должно перестраиваться особым образом под тесты,

- тестирование должно несколько раз покрывать исходный код, для обнаружения определённого класса ошибок,

- тестирование на предмет косвенного обнаружения ошибок. Например: соответствие стандартам разработки ПО.

**Задание:**

Разработать тесты к программе;

1. **Разработка тест-диспетчера на VisualTest**

Visual Test обеспечивает полнофункциональное, не зависящее от языка реализации тестирование 32-битных приложений, написанных для Windows, а также компонентов ActiveX.

Что тестировать с помощью данного средства - не имеет значения. Это может быть и 32-битное приложение, и компонент ActiveX, и сервер OLE, и даже Web приложение. Visual Test позволит решить каждую из поставленных задач, поскольку сам является автоматизированным средством тестирования всех описанных типов приложений. Гибкость Visual Test дает возможность создавать поддерживаемые, расширяемые и пригодные для повторного применения компоненты тестирования, которые можно приспосабливать ко многим версиям, а после некоторого планирования - ко многим проектам.

**Самый мощный и доступный инструмент программирования тестов**

В основе гибкости и мощи Visual Test лежит производный от Visual Basic экстенсивный язык программирования TestBasic, с сотнями специфических для тестирования функций, специальных конструкций, простого доступа к Windows API и открытой архитектурой, которая делает этот язык сильно расширяемым. С этим инструментом команда получает ключевую составную часть, необходимую для создания определенной разновидности описаний тестов для их повторного использования, что исключает утомительную работу, экономит много времени и понижает затраты на поставку высококачественного программного обеспечения. Такой подход позволит сосредоточиться на осуществлении всех возможных работ по тестированию программного обеспечения.

**Полная интеграция с Microsoft Visual Studio 6.0**

Даже самые мощные языки программирования нуждаются в полной среде разработки для того, чтобы быть продуктивными инструментами. Rational Visual Test 6.5 является единственным средством функционального тестирования, тесно интегрированного с Visual Studio и Integrated Development Environment (IDE) от Microsoft.

**Комплексное тестирование Web-приложений**

Для всех стало очевидно, что своевременный выпуск качественных WWW–приложений - задача трудновыполнимая, а досадное свойство проектов “сжиматься” в сроках отрицательно сказывается на качестве приложения. Visual Test предлагает тестировать логические схемы как клиентской части, так и серверной при помощи более чем 150 новых функций TestBasic, разработанных специально для тестирования в среде Web. Visual Test дает возможность запрашивать и устанавливать атрибуты стандартных элементов HTML и программных действий с входными элементами, “гулять” по ссылкам, выверять состояние и данные любого элемента. Кроме того, добавились новые возможности для решения следующих задач:

* тестирование компонентов ActiveX, выполняемое внутри броузера как часть Web – приложения,
* тестирование Dynamic HTML (DHTML) посредством управления пользовательским интерфейсом для отработки клиентской логики,
* управление специфическим для браузера расширением HTML, определенными пользователем тегами и последующим расширением стандарта HTML,
* тестирование приложений, для которых требуется множество экземпляров браузера или встроенных объектов управления в броузере,
* действия с синхронизационными выпусками, необходимость которых вызвана непредсказуемым хронометражем Web.

**Первый инструмент тестирования, поддерживающий Microsoft Active Accessibility**

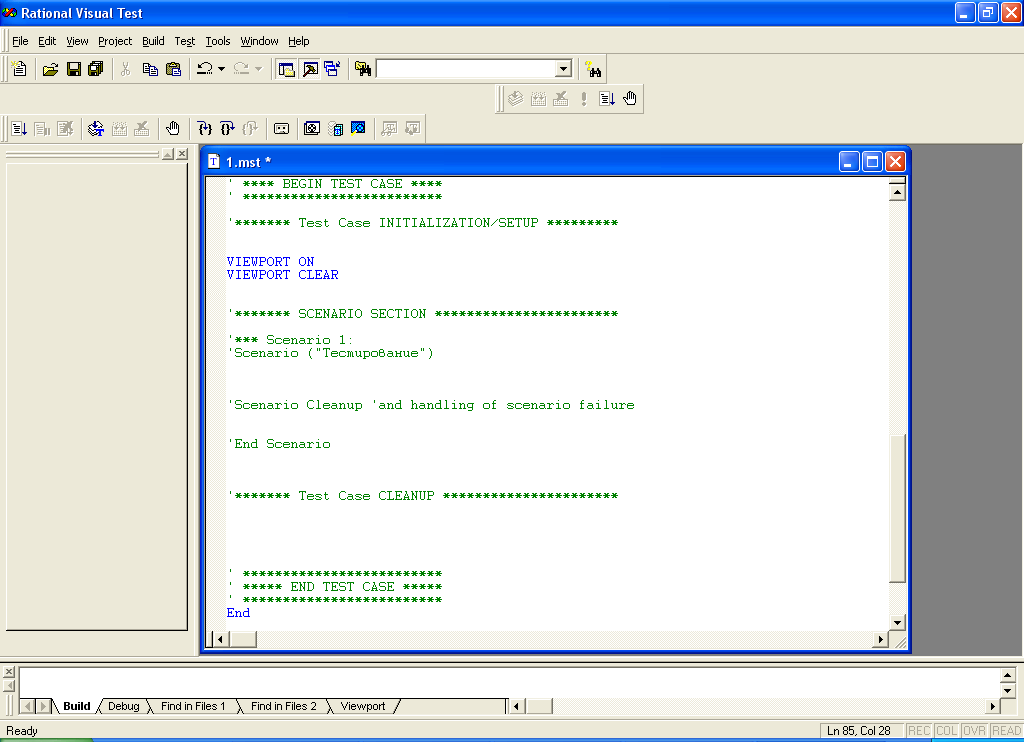
Rational Software – первая компания, предложившая инструмент тестирования, который поддерживает архитектуру Microsoft Active Accessibility (эта архитектура допускает управление заблокированными приложениями с помощью вспомогательных средств доступа). Выясняется, что когда к программному обеспечению есть доступ, оно более пригодно к тестированию. Visual Test содержит 30 функций, специально разработанных для того, чтобы воспользоваться преимуществами этой важной технологии подключения.

За счет использования в целях тестирования этого стандартного СОМ-интерфейса, Rational Software становится первой компанией в области распознавания, выбора и поддержки всего, что является основой стандарта производства более пригодных для тестирования приложений. Возможности, которые Visual Test обеспечивает для Active Accessibility, можно немедленно применить, с целью значительного упрощения тестирования разнообразных меню, написанных владельцем для компонентов операционных систем Windows и приложений, сконструированных с помощью Microsoft Office.

Также значительное преимущество возникает благодаря возможности тестирования компонентов настройки производства третьих фирм и окрашенных компонентов, а также трудных в тестировании компонентов.

**Основные свойства продукта:**

* обеспечивает полное, не зависящее от языка тестирование 32-битных Windows-приложений, компонентов ActiveX и DLL,
* автоматизирует повторяющиеся задачи тестирования с возвратом на более ранние стадии для увеличения эффективности проекта тестирования и дает возможность создавать высококачественные приложения за меньшее время,
* использует TestBasic, самый мощный язык программирования автоматизированного тестирования, чтобы дать возможность разрабатывать базу данных для постоянно используемых, поддерживаемых и расширяемых тестов,
* основывается на существующих возможностях Visual Test с массой новых функций для полного тестирования Web-приложений,
* вводит поддержку Microsoft Active Accessibility как первого шага в продвижении производных от него стандартов тестируемости в высших целях,
* предлагает компоненты, которые можно распространять дальше, так что сконструированные и разработанные инженерами QA тесты пригодны для дальнейшего распространения и обеспечивают максимальную эффективность издержек,
* тестирует существование и расположение компонентов, собирает качественные показатели и допускает обновление свойств, обеспечивая тщательное тестирование в вашем приложении компонентов OLE (OCXs) и ActiveX,
* содержит Suite Manager чтобы конструировать, контролировать и выполнять серии автоматизированных тестов.



**Начало работы:**

Файлы с входными и эталонными данными – текстовые файлы, которые должны содержать в себе тестируемые и эталонные значения. Файлы организованы следующим образом:

* входные и эталонные данные расположены в виде столбца. Если входных данных несколько, то перечисляются через “Enter” (1 строка – 1 данное),
* название файлов должно быть строго определено и они должны создаваться в определенном месте, иначе тестируемый файл не сможет их найти.

Приведем пример для задачи:

Задать функцию **fun16** с параметрами **R**, **r**, **S** (**R**, **r** – shortint; **S** - float). **R** и **r** - радиусы двух кругов, центры которых совпадают. Функция **fun16** вычисляет площадь **S** образуемого ими кольца. Результат выводить в Edit3.

Файлы с входными и эталонными значениями будут выглядеть:

|  |  |
| --- | --- |
|  |  |

В 1 тесте входными данными будут 2 пустые строки и эталонное значение будет соответствовать 1 строке из файла – эталона: «Введите целые положительные числа в оба поля для ввода!».

Во 2 тесте входными данными будут 1 и пустая строка, эталонное значение будет диагностическое сообщение: «Введите целые положительные числа в оба поля для ввода!».

Для описанного ниже кода: чтобы открыть файл проекта, тест-диспетчер должен находится в той же папке:

*DimFname$*

*Fname="Project1.exe" //название файла тестируемого проекта*

*if Exists(Fname) then*

*Open "TestT.txt" for input As #1*

*Open "TestE.txt" for input As #2*

*Dim Tst as String*

*Dim Etl as String*

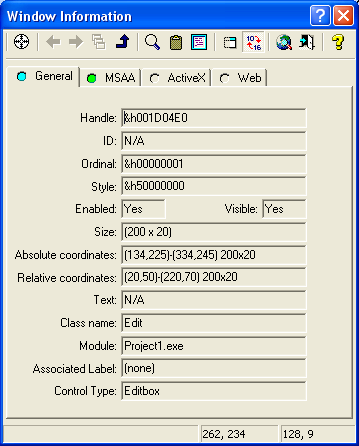
*LINE INPUT #1, Tst$*

*LINE INPUT #2, Etl$*

*Run Fname, NOWAIT*

*hwnd=WFndWnd("Практическая работа №") //поиск и фокус на окне проекта*

Дляполучениеинформации(Handle, Class…) обокне, edit, кнопки (итд), в Visual Test есть Window Information (Test-> Window Information)



Для проверки наличия edit используется функция

***WEditExists(****control$ [* ***,****timeout& ]****)***

Для проверки наличия кнопки используется функция

***WButtonExists(****control$ [* ***,****timeout& ]* ***)***

Для ввода текста в Edit и фокус на нем

***WEditSetText(****control$****,****buffer$ [* ***,****timeout& ]****)***

Эмуляциящелчкамышинаокне

***WClkWnd(****hwnd&****,*** *[ x& ]* ***,*** *[ y& ]* ***,*** *[ button& ]* ***)***

Эмуляциящелчкамышинакнопку

***WButtonClick(****control$****,*** *[ timeout& ]* ***)***

Чтениетекста

***GetText(*** *[ hwnd& ]* ***)***

Редактированиетекста

***EditText(****control$ [* ***,****timeout& ]* ***)***

Установказадержек

***Sleep(****seconds)*

**Подробнее об этих функциях вы можете прочитать в help**

Для закрытия окна проекта используется дополнительная функция, содержащаяся в файле «*KillProcess.inc».* Файл должен быть в той же папке, где и тест-диспетчер.

*' '$DEFINE and '$INCLUDE metacommands for test case*

*'-------------------------------------------------------------------*

*'$include 'KillProcess.inc'*

*....*

*'\*\*\* Scenario 1:*

*…*

*bKillProcess(Fname)*

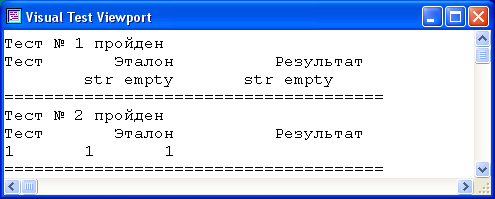
*…*

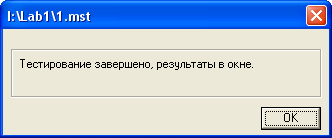
*'End Scenario*

**Задание:**

Написать тест-диспетчер на VisualTest и проверить программу, разработанными тестами.

Входные данные будут вводиться из текстового файла TestT.txt, и результаты сравниваться с эталонным файлом TestE.txt. Результат сравнения выводить в окне (см рис ниже).





Примеры задач с решением и тестами

**Вариант 1**

Написать программу используя функции WinAPI (WinMain, MessageBox, CreateWindow, ShowWindow, TextOut) которая при запуске создает окно, которое используется для вывода результатов работы, и завершает свое выполнение при его закрытии.

Задать функцию**fun11** с параметром **str** (строка с произвольными символами, содержащая не более 256 элементов). Функция **fun11** переводит строку **str** в верхний регистр. Результаты выводить в Edit2.

*Примечание: Заголовок окна для варианта 1 должен называться “Практическая работа № 1\_1”, для варианта 2 -“Практическаая работа № 1\_2” и тд.*

*Входные данные вводятся при помощи Edit, если несколько входных данных, то для каждой переменной свой Edit. Например, по заданию нужно ввести переменные a,b,c. Для переменной a создаем Edit1, для переменной b – Edit2, для переменной c – Edit3.*

*Еxe файл должен называться: «Project1.exe».*

*Вывод результирующих значений для каждого варианта описывается отдельно.*

Решение:

'\*\*\* Scenario 1:

'Scenario ("Тест №1")

Dim Dir$,Fname$,EdVal$,RezVal$,hwnd&,hwnd1&,hwnd2&

Fname="Lab1\_1\_ASU.exe"

EdVal=""

hwnd=0

if Exists(Fname) then

Open "TestT.txt" for input As #1 'Файлтестов

Open "TestE.txt" for input As #2 'Файлэталонныхрезультатов

Dim n as Integer 'Переменнаяцикла

Dim Tst as String 'Тестовоезначение

Dim Etl as String 'Этплонноезначение

Dim Rsl as String 'Результатпреобразования

n=0

Do While EOF (1) = Null

n=n+1

LINE INPUT #1, Tst$ 'Получаетм тестовое значение

LINE INPUT #2, Etl$ 'Получаем эталонное значение

Run Fname, NOWAIT

Sleep(1)

hwnd=WFndWnd("ПРАКТИЧЕСКАЯ РАБОТА № 1\_1")

if hwnd then

if WEditExists("@1") then

EdVal=EditText ("@1")

hwnd1 = GETHANDLE (1)

WEditSetText("@1", Tst )

WClkWnd(hwnd,100,100)

WButtonClick("ПРЕОБРАЗОВАТЬ") 'Щелчекпокнопке

if WEditExists("@2") then

RezVal=EditText ("@2")

hwnd2 = GETHANDLE (2)

Rsl = GetText(hwnd2)

Rsl = EditText("@2")

If Rsl = Etl Then

MSGBOX ("Тест №"+STR$(n)+" пройден")

Else

MSGBOX ("Тест №"+STR$(n)+" не пройден")

EndIf

print (" ======================================")

If Rsl = Etl Then

print ("Тест №"+STR$(n)+" пройден")

Else

print ("Тест №"+STR$(n)+" непройден")

EndIf

print "Тест", " Эталон", " Результат"

print Tst, Etl, Rsl

else

Print "Ненайден EditBox2"

endif

else

Print "Ненайден EditBox1"

endif

Sleep(1)

bKillProcess(Fname)

else

Print "Файлненайден"

endif

Loop

MSGBOX ("Тестирование завершено, результаты в окне.")

Close #1

else

Print "Ненайденоокно"

endif

'Scenario Cleanup 'and handling of scenario failure

'End Scenario

' \*\*\*\*\* END TEST CASE \*\*\*\*\*

End

|  |  |
| --- | --- |
| Входные данные | Эталонные данные |
| (пустая строка) | Строка не должна быть пустой. |
| 123 | 123 |
| aaa | AAA |
| a1 | A1 |
| ! « » № ; % : ? aa \* ( ) \_ - + = \ / ` ~ @ # $ ^ & { } [ ] ‘ ’ “ ” <> , . | ! « » № ; % : ? AA \* ( ) \_ - + = \ / ` ~ @ # $ ^ & { } [ ] ‘ ’ “ ” <> , . |
| aaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaab (260 символов) | Длина строки не должна превышать 256 символов |
| aaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaaa (256символов) | AAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAAAAABAAAAAA  (256 символов) |

ПРАКТИЧЕСКАЯ РАБОТА №3

**Тема:** Модульное тестирование

1.Цель работы

Изучить технологии модульного тестирования. Получить практические навыки по работе с Unit Testing Framework от Microsoft.

2. Краткие теоретические сведения

Модульное тестирование, или unit-тестирование (англ. unit testing) процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. В самом деле, это следует из практической невозможности трассировки всех возможных путей выполнения программы, за исключением простейших случаев. Кроме того, происходит тестирование каждого из модулей по отдельности. Это означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

Для получения выгоды от модульного тестирования требуется строго следовать технологии тестирования во всём процессе разработки программного обеспечения. Нужно хранить не только записи обо всех проведённых тестах, но и обо всех изменениях исходного кода во всех модулях. С этой целью следует использовать систему контроля версий ПО. Таким образом, если более поздняя версия ПО не проходит тест, который был успешно пройден ранее, будет несложным сверить исходный код вариантов и устранить ошибку. Также необходимо убедиться в неизменном отслеживании и анализе неудачных тестов. Игнорирование этого требования приведёт к лавинообразному увеличению неудачных тестовых результатов.

На данный момент наиболее распространенными инструментами модульного тестирования платформы .NET являются библиотека NUnit и Microsoft Unit Testing Framework.

Visual Studio Unit Testing Framework – инструмент модульного тестирования для платформы .NET, встроенный в среду разработки Visual Studio 2005 и выше. Чтобы определить, что класс является тестирующим, необходимо пометить его атрибутом [TestClass]. Если класс помечен этим атрибутом, то он может содержать в себе тестовые методы. Обычно тестирующий класс называют так же, как и тестируемый, только с префиксом Test. В тестирующем классе могут содержаться тестирующие методы и обычно для всех методов тестируемого класса, которые возвращают значение, создается отдельный тестирующий метод. Тестирующий метод обычно называют, так же как и тестируемый, только с префиксом Test.

Кроме тестирующих методов в тестирующем классе могут быть методы инициализации и очистки. Метод инициализации помечается атрибутом [TestInitialize] и позволяет инициализировать необходимые переменные перед выполнением метода-теста. Метод очистки помечается атрибутом [TestCleanup] и позволяет очистить результаты выполнения теста, например, очистить файл, удалить лишние записи с базы данных, присвоить переменным значения по умолчанию.

Кроме методов инициализации и очистки на уровне теста, в тестирующем классе могут присутствовать методы инициализации и очистки уровня класса. Эти методы вызываются один раз. Методы инициализации уровня класса вызывается один раз перед вызовом первого теста, а метод очистки уровня класса вызывается после выполнения последнего теста. Метод инициализации уровня класса помечается атрибутом [ClassInitialize], а метод очистки уровня класса помечается атрибутом [ClassCleanup].

3 Выполнение работы

Текст TestGenericDAO

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using NHibernate;

using NHibernate.Criterion;

using System.Collections.Generic;

using FluentNHibernate;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using FluentNHibernate.Automapping;

using NHibernate.Tool.hbm2ddl;

using System.Reflection;

using FluentNHibernate.Mapping;

using NHibernate.Cfg;

using lab6.mapping;

using lab6.dao;

using lab6.domain;

namespace lab6

{

[TestClass()]

public abstract class TestGenericDAO<T> where T : EntityBase

{

protected static ISessionFactory factory;

protected static ISession session;

protected DaoFactory daoFactory;

protected TestContext testContextInstance;

/\*\* DAO that will be tested \*/

protected IGenericDAO<T> dao = null;

/\*\* First entity that will be used in tests \*/

protected T entity1 = null;

/\*\* Second entity that will be used in tests \*/

protected T entity2 = null;

/\*\* Third entity that will be used in tests \*/

protected T entity3 = null;

public TestGenericDAO()

{

session = openSession("localhost", 5432, "market",

"postgres", "06031992");

}

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

/\*Getting dao this test case works with\*/

public IGenericDAO<T> getDAO()

{

return dao;

}

/\*Setting dao this test case will work with\*/

public void setDAO(IGenericDAO<T> dao)

{

this.dao = dao;

}

[ClassCleanup]

public static void ClassCleanup()

{

session.Close();

}

[TestInitialize]

public void TestInitialize()

{

Assert.IsNotNull(dao,

"Please, provide IGenericDAO implementation in constructor");

createEntities();

Assert.IsNotNull(entity1, "Please, create object for entity1");

Assert.IsNotNull(entity2, "Please, create object for entity2");

Assert.IsNotNull(entity3, "Please, create object for entity3");

checkAllPropertiesDiffer(entity1, entity2);

checkAllPropertiesDiffer(entity1, entity3);

checkAllPropertiesDiffer(entity2, entity3);

saveEntitiesGeneric();

}

[TestCleanup]

public void TestCleanup()

{

try

{

if ((entity1 = dao.GetById(entity1.Id)) != null)

dao.Delete(entity1);

}

catch (Exception)

{

Assert.Fail("Problem in cleanup method");

}

try

{

if ((entity2 = dao.GetById(entity2.Id)) != null)

dao.Delete(entity2);

}

catch (Exception)

{

Assert.Fail("Problem in cleanup method");

}

try

{

if ((entity3 = dao.GetById(entity3.Id)) != null)

dao.Delete(entity3);

}

catch (Exception)

{

Assert.Fail("Problem in cleanup method");

}

entity1 = null;

entity2 = null;

entity3 = null;

}

[TestMethod]

public void TestGetByIdGeneric()

{

T foundObject = null;

// Should not find with inexistent id

try

{

long id = DateTime.Now.ToFileTime();

foundObject = dao.GetById(id);

Assert.IsNull(foundObject, "Should return null if id is inexistent");

}

catch (Exception)

{

Assert.Fail("Should return null if object not found");

}

// Getting all three entities

getEntityGeneric(entity1.Id, entity1);

getEntityGeneric(entity2.Id, entity2);

getEntityGeneric(entity3.Id, entity3);

}

[TestMethod]

public void TestGetAllGeneric()

{

List<T> list = getListOfAllEntities();

Assert.IsTrue(list.Contains(entity1),

"After dao method GetAll list should contain entity1");

Assert.IsTrue(list.Contains(entity2),

"After dao method GetAll list should contain entity2");

Assert.IsTrue(list.Contains(entity3),

"After dao method GetAll list should contain entity3");

}

[TestMethod]

public void TestDeleteGeneric()

{

try

{

dao.Delete((T)null);

Assert.Fail("Should not delete entity will null id");

}

catch (Exception)

{

}

// Deleting second entity

try

{

dao.Delete(entity2);

}

catch (Exception)

{

Assert.Fail("Deletion should be successful of entity2");

}

// Checking if other two entities can be still found

getEntityGeneric(entity1.Id, entity1);

getEntityGeneric(entity3.Id, entity3);

// Checking if entity2 can not be found

try

{

T foundEntity = null;

foundEntity = dao.GetById(entity2.Id);

Assert.IsNull(foundEntity,

"After deletion entity should not be found with id " + entity2.Id);

}

catch (Exception)

{

Assert.Fail("Should return null if finding the deleted entity");

}

// Checking if other two entities can still be found in getAll list

List<T> list = getListOfAllEntities();

Assert.IsTrue(list.Contains(entity1),

"After dao method GetAll list should contain entity1");

Assert.IsTrue(list.Contains(entity3),

"After dao method GetAll list should contain entity3");

}

protected abstract void createEntities();

protected abstract void checkAllPropertiesDiffer(T entityToCheck1,

T entityToCheck2);

protected abstract void checkAllPropertiesEqual(T entityToCheck1,

T entityToCheck2);

protected void saveEntitiesGeneric()

{

T savedObject = null;

try

{

dao.SaveOrUpdate(entity1);

savedObject = getPersistentObject(entity1);

Assert.IsNotNull(savedObject,

"DAO method saveOrUpdate should return entity if successfull");

checkAllPropertiesEqual(savedObject, entity1);

entity1 = savedObject;

}

catch (Exception)

{

Assert.Fail("Fail to save entity1");

}

try

{

dao.SaveOrUpdate(entity2);

savedObject = getPersistentObject(entity2);

Assert.IsNotNull(savedObject,

"DAO method saveOrUpdate should return entity if successfull");

checkAllPropertiesEqual(savedObject, entity2);

entity2 = savedObject;

}

catch (Exception)

{

Assert.Fail("Fail to save entity2");

}

try

{

dao.SaveOrUpdate(entity3);

savedObject = getPersistentObject(entity3);

Assert.IsNotNull(savedObject,

"DAO method saveOrUpdate should return entity if successfull");

checkAllPropertiesEqual(savedObject, entity3);

}

catch (Exception)

{

Assert.Fail("Fail to save entity3");

}

}

protected T getPersistentObject(T nonPersistentObject)

{

ICriteria criteria = session.CreateCriteria(typeof(T)).Add(Example.Create(nonPersistentObject));

IList<T> list = criteria.List<T>();

Assert.IsTrue(list.Count >= 1,

"Count of grups must be equal or more than 1");

return list[0];

}

protected void getEntityGeneric(long id, T entity)

{

T foundEntity = null;

try

{

foundEntity = dao.GetById(id);

Assert.IsNotNull(foundEntity,

"Service method getEntity should return entity if successfull");

checkAllPropertiesEqual(foundEntity, entity);

}

catch (Exception)

{

Assert.Fail("Failed to get entity with id " + id);

}

}

protected List<T> getListOfAllEntities()

{

List<T> list = null;

// Should get not null and not empty list

try

{

list = dao.GetAll();

}

catch (Exception)

{

Assert.Fail(

"Should be able to get all entities that were added before");

}

Assert.IsNotNull(list,

"DAO method GetAll should return list of entities if successfull");

Assert.IsFalse(list.Count == 0,

"DAO method should return not empty list if successfull");

return list;

}

//Методоткрытиясессии

public static ISession openSession(String host, int port,

String database, String user, String passwd)

{

ISession session = null;

if (factory == null)

{

FluentConfiguration configuration = Fluently.Configure()

.Database(PostgreSQLConfiguration

.PostgreSQL82.ConnectionString(c => c

.Host(host)

.Port(port)

.Database(database)

.Username(user)

.Password(passwd)))

.Mappings(m => m.FluentMappings.Add<TovarsMap>().Add<ProdMap>())

.ExposeConfiguration(BuildSchema);

factory = configuration.BuildSessionFactory();

}

//Открытиесессии

session = factory.OpenSession();

return session;

}

//Метод для автоматического создания таблиц в базе данных

private static void BuildSchema(Configuration config)

{

new SchemaExport(config).Create(false, true);

}

}

}

Текст TestProdDAO

using System;

using System.Collections.Generic;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using lab6.domain;

using lab6.dao;

namespace lab6

{

[TestClass]

public class TestProdDAO : TestGenericDAO<Prod>

{

protected IProdDAO prodDAO = null;

protected Tovars tovar1 = null;

protected Tovars tovar2 = null;

protected Tovars tovar3 = null;

public TestProdDAO()

: base()

{

DaoFactory daoFactory = new NHibernateDAOFactory(session);

prodDAO = daoFactory.getProdDAO();

setDAO(prodDAO);

}

protected override void createEntities()

{

entity1 = new Prod();

entity1.NameProd = "1";

entity1.Surname = "1";

entity1.God = 1992;

entity2 = new Prod();

entity2.NameProd = "2";

entity2.Surname = "2";

entity2.God = 1993;

entity3 = new Prod();

entity3.NameProd = "3";

entity3.Surname = "3";

entity3.God = 1994;

}

protected override void checkAllPropertiesDiffer(Prod entityToCheck1,

Prod entityToCheck2)

{

Assert.AreNotEqual(entityToCheck1.NameProd, entityToCheck2.NameProd, "Values must be different");

Assert.AreNotEqual(entityToCheck1.Surname,entityToCheck2.Surname, "Values must be different");

Assert.AreNotEqual(entityToCheck1.God,entityToCheck2.God, "Values must be different");

}

protected override void checkAllPropertiesEqual(Prod entityToCheck1,

Prod entityToCheck2)

{

Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname,

"Values must be equal");

Assert.AreEqual(entityToCheck1.God, entityToCheck2.God,

"Values must be equal");

}

[TestMethod]

public void TestGetByIdProd()

{

base.TestGetByIdGeneric();

}

[TestMethod]

public void TestGetAllProd()

{

base.TestGetAllGeneric();

}

[TestMethod]

public void TestDeleteProd()

{

base.TestDeleteGeneric();

}

// [TestMethod]

// public void TestGetProdByName()

// {

// Prod prod1 = prodDAO.getProdByName(entity1.NameProd);

// Assert.IsNotNull(prod1,

// "Service method getGroupByName should return group if successfull");

// Prod prod2 = prodDAO.getProdByName(entity2.NameProd);

// Assert.IsNotNull(prod2,

// "Service method getGroupByName should return group if successfull");

// Prod prod3 = prodDAO.getProdByName(entity3.NameProd);

// Assert.IsNotNull(prod3,

// "Service method getGroupByName should return group if successfull");

// checkAllPropertiesEqual(prod1, entity1);

// checkAllPropertiesEqual(prod2, entity2);

// checkAllPropertiesEqual(prod3, entity3);

// }

[TestMethod]

public void TestGetAllTovarsOfProd()

{

createEntitiesForTovars();

Assert.IsNotNull(tovar1, "Please, create object for student1");

Assert.IsNotNull(tovar2, "Please, create object for student2");

Assert.IsNotNull(tovar3, "Please, create object for student3");

entity1.TovarsList.Add(tovar1);

tovar1.Prod = entity1;

entity1.TovarsList.Add(tovar2);

tovar2.Prod = entity1;

entity1.TovarsList.Add(tovar3);

tovar3.Prod = entity1;

Prod savedObject = null;

try

{

dao.SaveOrUpdate(entity1);

savedObject = getPersistentObject(entity1);

Assert.IsNotNull(savedObject,

"DAO method saveOrUpdate should return entity if successfull");

checkAllPropertiesEqual(savedObject, entity1);

entity1 = savedObject;

}

catch (Exception)

{

Assert.Fail("Fail to save entity1");

}

IList<Tovars> tovarsList =

prodDAO.getAllTovarsOfProd(entity1.NameProd);

Assert.IsNotNull(tovarsList, "List can't be null");

Assert.IsTrue(tovarsList.Count == 3,

"Count of students in the list must be 3");

checkAllPropertiesEqualForTovars(tovarsList[0], tovar1);

checkAllPropertiesEqualForTovars(tovarsList[1], tovar2);

checkAllPropertiesEqualForTovars(tovarsList[2], tovar3);

}

//[TestMethod]

//public void TestDelProdByName()

//{

// try

// {

// prodDAO.delProdByName(entity2.NameProd);

// }

// catch (Exception)

// {

// Assert.Fail("Deletion should be successful of entity2");

// }

// Checking if other two entities can be still found

// getEntityGeneric(entity1.Id, entity1);

// getEntityGeneric(entity3.Id, entity3);

// Checking if entity2 can not be found

// try

// {

// Prod foundProd = null;

// foundProd = dao.GetById(entity2.Id);

// Assert.IsNull(foundProd,

// "After deletion entity should not be found with groupName " +

// entity2.NameProd);

// }

// catch (Exception)

// {

// Assert.Fail("Should return null if finding the deleted entity");

// }

// Checking if other two entities can still be found in getAll list

// List<Prod> list = getListOfAllEntities();

// Assert.IsTrue(list.Contains(entity1),

// "After dao method GetAll list should contain entity1");

// Assert.IsTrue(list.Contains(entity3),

// "After dao method GetAll list should contain entity3");

// }

protected void createEntitiesForTovars()

{

tovar1 = new Tovars();

tovar1.Name = "С";

tovar1.Price = 10;

tovar1.Ves = 100;

tovar2 = new Tovars();

tovar2.Name = "B";

tovar2.Price = 20;

tovar2.Ves = 200;

tovar3 = new Tovars();

tovar3.Name = "D";

tovar3.Price = 30;

tovar3.Ves = 300;

}

protected void checkAllPropertiesEqualForTovars(Tovars entityToCheck1,

Tovars entityToCheck2)

{

Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Price, entityToCheck2.Price,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves,

"Values must be equal");

}

}

}

Текст TestTovarsDAO

using System;

using System.Collections.Generic;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using lab6.domain;

using lab6.dao;

using NHibernate.Criterion;

using NHibernate;

namespace lab6

{

[TestClass]

public class TestTovarsDAO : TestGenericDAO<Tovars>

{

protected ITovarsDao tovarsDAO = null;

protected IProdDAO prodDAO = null;

protected Prod prod = null;

public TestTovarsDAO()

: base()

{

DaoFactory daoFactory = new NHibernateDAOFactory(session);

tovarsDAO = daoFactory.getTovarsDAO();

prodDAO = daoFactory.getProdDAO();

setDAO(tovarsDAO);

}

protected override void createEntities()

{

entity1 = new Tovars();

entity1.Name = "С";

entity1.Price = 10;

entity1.Ves = 100;

entity2 = new Tovars();

entity2.Name = "B";

entity2.Price = 20;

entity2.Ves = 200;

entity3 = new Tovars();

entity3.Name = "D";

entity3.Price = 30;

entity3.Ves = 300;

}

protected override void checkAllPropertiesDiffer(Tovars entityToCheck1,

Tovars entityToCheck2)

{

Assert.AreNotEqual(entityToCheck1.Name, entityToCheck2.Name,

"Values must be different");

Assert.AreNotEqual(entityToCheck1.Price, entityToCheck2.Price,

"Values must be different");

Assert.AreNotEqual(entityToCheck1.Ves, entityToCheck2.Ves,

"Values must be different");

}

protected override void checkAllPropertiesEqual(Tovars entityToCheck1,

Tovars entityToCheck2)

{

Assert.AreEqual(entityToCheck1.Name, entityToCheck2.Name,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Price, entityToCheck2.Price,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Ves, entityToCheck2.Ves,

"Values must be equal");

}

[TestMethod]

public void TestGetByIdTovars()

{

base.TestGetByIdGeneric();

}

[TestMethod]

public void TestGetAllTovars()

{

base.TestGetAllGeneric();

}

[TestMethod]

public void TestDeleteTovars()

{

base.TestDeleteGeneric();

}

[TestMethod]

public void TestGetTovarByProdName()

{

Prod prod = new Prod();

prod.NameProd = "1";

prod.Surname = "1";

prod.God = 1992;

prod.TovarsList.Add(entity1);

entity1.Prod = prod;

prod.TovarsList.Add(entity2);

entity2.Prod = prod;

prod.TovarsList.Add(entity3);

entity3.Prod = prod;

Prod savedProd = null;

try

{

prodDAO.SaveOrUpdate(prod);

savedProd = getPersistentProd(prod);

Assert.IsNotNull(savedProd,

"DAO method saveOrUpdate should return group if successfull");

checkAllPropertiesEqualProd(savedProd, prod);

prod = savedProd;

}

catch (Exception)

{

Assert.Fail("Fail to save group");

}

getTovarByProdName(entity1, prod.NameProd,

entity1.Name);

getTovarByProdName(entity2, prod.NameProd,

entity2.Name);

getTovarByProdName(entity3, prod.NameProd,

entity3.Name);

prod.TovarsList.Remove(entity1);

prod.TovarsList.Remove(entity2);

prod.TovarsList.Remove(entity3);

entity1.Prod = null;

entity2.Prod = null;

entity3.Prod = null;

prodDAO.Delete(prod);

}

protected void getTovarByProdName(Tovars tovar, string nameProd, string name)

{

Tovars foundTovars = null;

try

{

foundTovars = tovarsDAO.getTovarByProdName(

nameProd,name);

Assert.IsNotNull(tovarsDAO,

"Service method should return student if successfull");

checkAllPropertiesEqual(foundTovars, tovar);

}

catch (Exception)

{

Assert.Fail("Failed to get student with nameProd " +

nameProd + " name " + name);

}

}

protected Prod getPersistentProd(Prod nonPersistentProd)

{

ICriteria criteria = session.CreateCriteria(typeof(Prod))

.Add(Example.Create(nonPersistentProd));

IList<Prod> list = criteria.List<Prod>();

Assert.IsTrue(list.Count >= 1,

"Count of grups must be equal or more than 1");

return list[0];

}

protected void checkAllPropertiesEqualProd(Prod entityToCheck1,

Prod entityToCheck2)

{

Assert.AreEqual(entityToCheck1.NameProd, entityToCheck2.NameProd,

"Values must be equal");

Assert.AreEqual(entityToCheck1.Surname, entityToCheck2.Surname,

"Values must be equal");

Assert.AreEqual(entityToCheck1.God, entityToCheck2.God,

"Values must be equal");

}

}

}

4. Выводы

В ходе выполнения данной работы были изучены технологии модульного тестирования. Были получены практические навыки по работе с Unit Testing Framework от Microsoft.

Модульное тестирование, или unit-тестирование (англ. unit testing) процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

ПРАКТИЧЕСКАЯ РАБОТА №4

**Тема:** Интеграционное тестирование

Цель:

1. Изучение назначения и задач модульного тестирования.

2. Программная реализация тестов, производящих модульное тестирование алгоритма пирамидальной сортировки.

**Юнит-тестирование**

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок. В данной работе мы будем использовать юнит-тесты для проверки функциональных требований программы.

Для использования юнит-тестов будем использовать JUnit. JUnit — библиотека для модульного тестирования программного обеспечения на языке Java. Скачать свежую версию данной библиотеки можно по адресу .

Пример юнит-теста, проверяющего равенство 2+2=4, приведен в листинге 4.1.

Листинг 4.1 – пример юнит-теста

import org.junit.Test;

import junit.framework.Assert;

public class MathTest {

@Test

public void testEquals() {

Assert.assertEquals(4, 2 + 2);

Assert.assertTrue(4 == 2 + 2);

}

@Test

public void testNotEquals() {

Assert.assertFalse(5 == 2 + 2);

}

}

Для модульного тестирования необходимо использовать драйверы и заглушки.

Unit (Элемент) — наименьший компонент, который можно скомпилировать.

Драйверы — модули тестов, которые запускают тестируемый элемент.

Заглушки — заменяют недостающие компоненты, которые вызываются элементом и выполняют следующие действия:

* возвращаются к элементу, не выполняя никаких других действий;
* отображают трассировочное сообщение и иногда предлагают тестеру продолжить тестирование;
* возвращают постоянное значение или предлагают тестеру самому ввести возвращаемое значение;
* осуществляют упрощенную реализацию недостающей компоненты;
* имитируют исключительные или аварийные условия.

**Предварительная подготовка к работе**

Так как алгоритм реализован на языке Java, то юнит-тесты следует писать с использованием библиотеки JUnit (). Допускается использовать любую удобную среду разработки (IDEA / Eclipse / др.).

Для упрощения задачи, предлагается выполнить интеграционное тестирование по восходящему подходу. При использовании этого метода подразумевается, что сначала тестируются все программные модули, входящие в состав системы, и только затем они объединяются для интеграционного тестирования. При таком подходе значительно упрощается локализация ошибок: если модули протестированы по отдельности, то ошибка при их совместной работе есть проблема их интерфейса.

Несмотря на то, что интеграционные тесты не являются юнит-тестами в классическом понимании, при выполнении задач работы необходимо использовать библиотеку JUnit и написать модуль юнит-тестирования вручную (используя Java-аннотации @Test и документацию библиотеки JUnit).

Реализовать юнит-тесты следует так, чтобы они выполнял функциональное тестирование программы сортировки в соответствии с целями работы.

**JUnit**

Таблица 1. Аннотации.

| **Аннотация** | **Описание** |
| --- | --- |
| @Test public void method() | Помечает метод как тестовый |
| @Before public void method() | Метод будет выполняться перед каждым тестом, может быть использован для подготовки тестового окружения (инициализации, чтения данных) |
| @After public void method() | Метод будет выполняться после каждого теста. Может использоваться для отчистки окружения, удаления временных данных. |

Таблица 2. Тестовые методы

| **Сигнатура** | **Описание** |
| --- | --- |
| fail(String) | Вызывает сбой. Может быть использован, чтобы удостовериться, что определенная часть кода не достигнута или пока тестовый метод не реализован. |
| assertTrue(true) / assertTrue(false) | Постоянно будет true / false. Может быть использован, чтобы предопределить результат теста, пока он не реализован. |
| assertsEquals([String message], expected, actual) | Проверяет, равны ли две величины. Важно: для массивов проверяется ссылка а не содержимое. |
| assertsEquals([String message], expected, actual, tolerance) | Проверяет, совпадают ли величины float и double |
| assertNull([message], object) | Проверяет, что объект null. |
| assertNotNull([message], object) | Проверяет, что объект не null. |
| assertSame([String], expected, actual) | Проверяет, что обе переменные ссылаются на один объект. |
| assertNotSame([String], expected, actual) | Проверяет, что переменные ссылаются на разные объекты. |
| assertTrue([message], boolean condition) | Проверяет условие на истинность. |

**Порядок выполнения работы**

1. Откройте выбранную IDE и создайте проект на основе существующих программных кодов, реализующих алгоритм пирамидальной сортировки.
2. Подключите к проекту библиотеку JUnit.
3. Создайте каркас для юнит-тестов (Например, в IDE Eclipse можно выбрать нужный класс, открыть контекстное меню, и выбрать New->Junit test case, в появившемся диалоговом окне выбрать методы, для которых понадобятся юнит-тесты).
4. Создайте юнит-тест согласно описанным требованиям.
5. Отладьте и запустите юнит-тест.
6. Оцените результаты выполнения юнит-тестирования и сделайте соответствующие выводы.

**Содержание отчёта**

1. Необходимо представить исходный код, описание и результаты работы юнит-теста на проверку правильности интеграции модулей системы.
2. Изложить вводы по результатам тестирования и проделанной работе.

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ №5

**Тема**: Оформление документации на программные средства с использованием инструментальных средств.

Стандарты ЕСПД серии ГОСТ 19.ххх регламентируют оформление и содержание программных документов. Разработка программной документации, как правило, осуществляется в соответствии со следующими стандартами:

• ГОСТ 19.001-77 «Общие положения»;

• ГОСТ 19781-90 «Термины и определения»;

• ГОСТ 19.101-77 «Виды программ и программных документов»;

• ГОСТ 19.102-77 «Стадии разработки»;

• ГОСТ 19.103-77 «Обозначения программ и программных документов»;

• ГОСТ 19.104-78 «Основные надписи»;

• ГОСТ 19.105-78 «Общие требования к программным документам»;

• ГОСТ 19.106-78 «Требования к программным документам, выполненным печатным способом»;

• ГОСТ 19.201-78 «Техническое задание, требования к содержанию и оформлению»;

• ГОСТ 19.202-78 «Спецификация. Требования к содержанию и оформлению»;

• ГОСТ 19.301-79 «Программа и методика испытаний. Требования к содержанию и оформлению»;

• ГОСТ 19.401-78 «Текст программы. Требования к содержанию и оформлению»;

• ГОСТ 19.402-78 «Описание программы»;

• ГОСТ 19.403-79 «Ведомость держателей подлинников»;

• ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению»;

• ГОСТ 19.501-78 «Формуляр. Требования к содержанию и оформлению»;

• ГОСТ 19.502-78 «Описание применения. Требования к содержанию и оформлению»;

• ГОСТ 19.503-79 «Руководство системного программиста. Требования к содержанию и оформлению»;

• ГОСТ 19.504-79 «Руководство программиста. Требования к содержанию и оформлению»;

• ГОСТ 19.505-79 «Руководство оператора. Требования к содержанию и оформлению»;

• ГОСТ 19.506-79 «Описание языка. Требования к содержанию и оформлению»;

• ГОСТ 19.507-79 «Ведомость эксплуатационных документов»;

• ГОСТ 19.508-79 «Руководство по техническом обслуживанию. Требования к содержанию и оформлению»;

• ГОСТ 19.601-78 «Общие правила дублирования, учета и хранения»;

• ГОСТ 19.602-78 «Правила дублирования, учета и хранения программных документов, выполненных печатным способом»;

• ГОСТ 19.603-78 «Общие правила внесения изменений»;

• ГОСТ 19.604-78 «Правила внесения изменений в программные документы, выполненных печатным способом»;

• ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения».

Стандарты ЕССАСУ серии ГОСТ 24.ххх регламентируют требования к автоматизированным системам управления и документам по их описанию. Разработка программной документации на автоматизированные системы управления, как правило, осуществляется в соответствии со следующими стандартами:

• ГОСТ 24.104-85 «Единая система стандартов автоматизированных систем управления. Автоматизированные системы управления. Общие требования»;

• ГОСТ 24.103-84 «Автоматизированные системы управления. Общие положения»;

• ГОСТ 24.202-80 «Требования к содержанию документа «Технико-экономическое обоснование»;

• ГОСТ 24.203-80 «Требования к содержанию общесистемных документов»;

• ГОСТ 24.204-80 «Требования к содержанию документа «Описание постановки задачи»;

• ГОСТ 24.205-80 «Требования к содержанию документов по информационному обеспечению»;

• ГОСТ 24.206-80 «Требования к содержанию документов по техническому обеспечению»;

• ГОСТ 24.207-80 «Требования к содержанию документов по программному обеспечению»;

• ГОСТ 24.208-80 «Требования к содержанию документов стадии «Ввод в эксплуатацию»;

• ГОСТ 24.209-80 «Требования к содержанию документов по организационному обеспечению»;

• ГОСТ 24.210-82 «Требования к содержанию документов по функциональной части»;

• ГОСТ 24.211-82 «Требования к содержанию документа «Описание алгоритма»;

• ГОСТ 24.301-80 «Общие требования к выполнению текстовых документов»;

• ГОСТ 24.302-80 «Общие требования к выполнению схем»;

• ГОСТ 24.304-82 «Требования к выполнению чертежей»;

• ГОСТ 24.703-85 «Типовые проектные решения. Основные положения».

Российские и международные стандарты по вопросам оказания ИТ-услуг

Лучшие практики представления ИТ-услуг, включая различные этапы ЖЦ ИС, описаны в томах британской библиотеки ITIL («айтйл», от англ. IT Infrastructure Library- библиотека инфраструктуры ИТ). В настоящее время ITIL становится стандартом де-факто и лежит в основе международного стандарта ISO/IEC 20000:2011, регламентирующего процессы организации ИТ-услуг. Российским аналогом этого стандарта является ГОСТ Р ИСО/МЭК 20000 «Информационная технология. Менеджмент услуг».

Российские и международные стандарты по вопросам информационной безопасности

• ГОСТ Р 51188-98 «Защита информации. Испытания программных средств на наличие компьютерных вирусов. Типовое руководство».

• ГОСТ Р 50739-95 «Средства вычислительной техники. Защита от несанкционированного доступа к информации. Общие технические требования».

• ГОСТ Р ИСО/МЭК 15408-2002 «Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий».

• ГОСТ Р ИСО/МЭК 27001-2006 «Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности. Требования».

• IEEE Std 1228-1994 IEEE Standard for Software Safety Plans.

• IEEE Std 1363a-2004 IEEE Standard Specifications for Public-Key Cryptography- Amendment 1: Additional Techniques.

• IEEE Std 1363.1-2008 IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices.

• IEEE Std 1363.2-2008 IEEE Standard Specifications for Password- Based Public-Key Cryptographic Techniques.

• IEEE Std 1363.3-2013 IEEE Standard for Identity-Based Cryptographic Techniques using Pairings.

Российские и международные стандарты, регламентирующие

общие термины, определения и понятия жизненного цикла ИС

• ГОСТ Р ИСО/МЭК ТО 12182-2002 «Информационная технология. Классификация программных средств».

• ГОСТ Р ИСО/МЭК 12207-2010 «Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств».

• ГОСТ РВ 51717-2001 «Программное средство. Общие требования».

• ГОСТ Р 53622-2009 «Информационные технологии. Информационно-вычислительные системы. Стадии и этапы жизненного цикла, виды и комплектность документов».

• ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания».

• ГОСТ 34.003-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Терминыиопределения».

• ISO/IEC/IEEE 16326 «Systems and software engineering - Life cycle processes - Project management».

Задание.

Для готового программного модуля, разработанного как Win32 API-приложение, создать руководство пользователя программного продукта.

Методические указания:

Документация должна содержать необходимые сведения по установке, обеспечению надёжной работы продукта, справочное пособие для пользователя, демонстрационные версии, примеры документов, создаваемых при помощи данного программного продукта, обучающие программы.

Программная документация, включает:

1. [техническое задание](https://pandia.ru/text/category/tehnicheskie_zadaniya__obshaya_/) (назначение, область применения программы, требования, предъявляемые к программе);

2. текст программы (запись программы с необходимыми комментариями);

3. описание программы (сведения о логической структуре и функционировании программы);

4. [пояснительная записка](https://pandia.ru/text/category/poyasnitelmznie_zapiski/) (схема алгоритма, общее описание алгоритма и/или функционирования программы, обоснование принятых решений);

5. эксплуатационные документы.

5. Эксплуатационные документы

К эксплуатационным документам относят:

· описание применения (сведения о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств);

-руководство системного программиста (сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения);

· руководство программиста (сведения для эксплуатации программы);

· руководство оператора (сведения для обеспечения общения оператора с вычислительной системой в процессе выполнения программы);

· описание языка (описание синтаксиса и семантики языка);

· руководство по техническому обслуживанию (сведения для применения тестовых и диагностических программ при обслуживании технических средств)

Основная часть программной документации составляется на стадии рабочего проекта. Необходимость того или иного документа определяется на этапе составления технического задания. Допускается объединять отдельные виды документов.

Эксплуатационный документ "Описание языка" включается в программную документацию, если разработанный программный продукт реализует некий язык программирования, управления заданиями, организации вычислительного процесса и т. п.

Эксплуатационный документ "Руководство по техническому обслуживанию" включается в программную документацию, если разработанный программный продукт требует использования тестовых или диагностических программ.

Описание применения

Документ "Описание применения" относится к эксплуатационным документам и состоит из следующих разделов:

· назначение программы (возможности, основные характеристики, ограничения области применения);

· условия применения (требования к техническим и программным средствам, общие характеристики входной и выходной информации, а также требования и условия организационного, технического и технологического характера);

описание задачи (указываются определения задачи и методы её решения);

· входные и выходные данные.

Руководство программиста

Документ "Руководство программиста" относится к эксплуатационным документам и включается в программную документацию, если разработанный программный продукт требует обслуживания программистом. Документ состоит из следующих разделов:

· назначение и условия применения программы (назначение и функции программы, сведения о технических и программных средствах, обеспечивающих выполнение данной программы);

· характеристики программы (временные характеристики, режимы работы, средства контроля правильности выполнения и т. п.);

· обращение к программе (способы передачи управления и параметров данных);

· входные и выходные данные (формат и кодирование);

· сообщения (тексты сообщений, выдаваемых программисту или оператору в ходе выполнения программы и описание действий, которые необходимо предпринять по этим сообщениям).

Руководство оператора

Документ "Руководство оператора" относится к эксплуатационным документам и состоит из следующих разделов:

· назначение программы (информация, достаточная для понимания функций программы и её эксплуатации);

· условия выполнения программы (минимальный и/или максимальный набор технических и программных средств и т. п.);

· выполнение программы (последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы; описываются функции, форматы и возможные варианты команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды);

· сообщения оператору (тексты сообщений, выдаваемых оператору в ходе выполнения программы и описание действий, которые необходимо предпринять по этим сообщениям).