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**

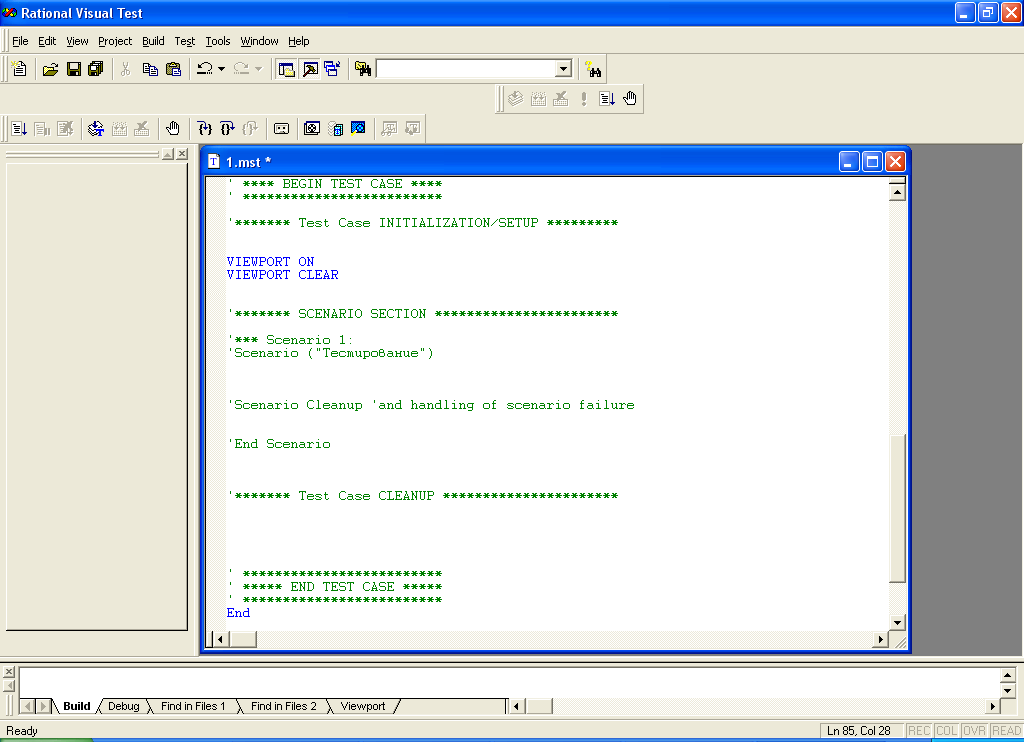
Rational Software – первая компания, предложившая инструмент тестирования, который поддерживает архитектуру Microsoft Active Accessibility (эта архитектура допускает управление заблокированными приложениями с помощью вспомогательных средств доступа). Выясняется, что когда к программному обеспечению есть 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*

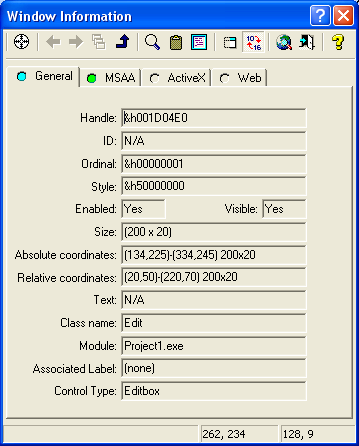
*LINE INPUT #1, Tst$*

*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)*

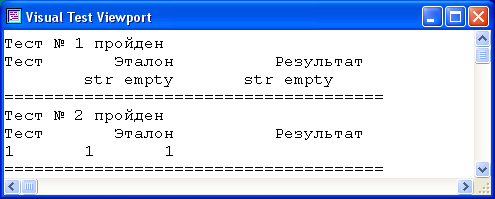
*…*

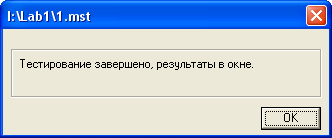
*'End Scenario*

**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) |