1. АНАЛИТИЧЕСКАЯ ЧАСТЬ
   1. Обоснование актуальности задачи
      1. Актуальность темы

В современном мире существует необходимость в быстром и качественном контроле знаний людей из различных социальных категорий (школьники, студенты, соискатели и др.), а также в точной и независимой оценке полученных результатов. Выполнение этих работ требует больших трудовых затрат и привлечение квалифицированных специалистов. В связи с этим требуется значительно повысить производительность труда при проведении всех видов контроля знаний.

Использование программного обеспечения облегчает и ускоряет многие процессы, и при этом не требует хорошего усвоения определённых приёмов и навыков, знания устройства компьютера и умения пользоваться современными приспособлениями, инструментами и программами.

* + 1. Анализ существующих разработок по теме

В настоящее время существуют автоматизированные системы для учебных заведений, такие как: MyTestXPro, NetTest и Ассистент II.

MyTestX это - система программ (программа тестирования учащихся, редактор тестов и журнал результатов) для создания и проведения компьютерного тестирования, сбора и анализа результатов, выставления оценки по указанной в тесте шкале.

Программа MyTestX работает с десятью типами заданий: одиночный выбор, множественный выбор, установление порядка следования, установление соответствия, указание истинности или ложности утверждений, ручной ввод числа, ручной ввод текста, выбор места на изображении, перестановка букв, заполнение пропусков (MyTestXPro). В тесте можно использовать любое количество любых типов, можно только один, можно и все сразу. В заданиях с выбором ответа (одиночный, множественный выбор, указание порядка, указание истинности) можно использовать до 10 (включительно) вариантов ответа.

Программа состоит из трех модулей: Модуль тестирования (MyTestStudent), Редактор тестов (MyTestEditor) и Журнал тестирования (MyTestServer). С помощью программ MyTestX можно организовать как локальное так и сетевое тестирование.

Программный комплекс NetTest предназначен для проведения массового компьютерного тестирования знаний в локальной сети под управлением операционных систем Windows и Linux. Он полностью автоматизирует:

1. процедуру выбора вопросов из базы данных;
2. тестирование на рабочих станциях с ограничением времени;
3. обработку результатов теста и их оформление;
4. анализ результатов (какие вопросы вызвали сложности).

Комплекс состоит из двух программ — серверной и клиентской. Серверная программа позволяет создавать и редактировать тесты, она также осуществляет полное управление компьютерным тестированием, обработку и вывод результатов. Клиентская программа запускается с рабочих станций и предназначена для работы учащегося.

Обе программы построены по принципу «минимальной достаточности» — реализуют необходимые функции оптимальным способом. На рабочем месте учащихся не требуется установка каких-либо программ, достаточно сделать ярлык на клиентскую программу, размещенную на сервере.

Таким образом, решения, которые существуют на данный момент, являются многофункционалными, хорошо отлаженными, но специализированными для тестирования только в учебных целях. Однако они платные, сложные в освоении и достаточно громоздкие. Также, в некоторых много лишних функций, в других - нет некоторых востребованных функций. Поэтому было принято решение о необходимости собственной разработки, которая будет отвечать всем поставленным требованиям и иметь интуитивно понятный интерфейс. А также не потребует систематической покупки лицензий. Программный продукт создаваемый в рамках практической работы, прежде всего, ориентирован на пользователей, не имеющих специализированных компьютерных навыков.

* 1. Обоснование выбора и описание методологии разработки программы решения задачи

Для данной работы был выбран объектно-ориентированный подход, как более близкий к реальным сущностям предметной области.

ООП — это подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что в свою очередь особенно важно при реализации крупных проектов. Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

1. абстрагирование для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счете — контекстное понимание предмета, формализуемое в виде класса;
2. инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
3. наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя все остальное, учтенное на предыдущих шагах;
4. полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

То есть фактически речь идет о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». Прогрессирование, в частности, на последнем этапе дает возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает выражение представления через имя предмета до полноценного понятия-класса.

* 1. Обоснование выбора и описание инструментальных средств разработки программы решения задачи

При выборе инструментальных средств разработки, прежде всего, были приняты во внимание все их доступность и надёжность, а также их функционал. Для решения поставленной задачи было решено использовать: CASE-средства для формализации задачи и проектирования, а также IDE для непосредственного написания программы.

Проанализировав доступную информацию, был сделан вывод, что наиболее надежными CASE-средствами будут BPWin и Rational Rose.

Визуальное моделирование оказало большое влияние на развитие ТС ПО вообще и CASE-средств в частности. Понятие CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение этого понятия, ограниченное только задачами автоматизации разработки ПО, в настоящее время приобрело новый смысл, охватывающий большинство процессов жизненного цикла ПО. CASEтехнология представляет собой совокупность методов проектирования ПО, а так же набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки и сопровождения ПО и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE - средств основано на методах структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

BPWin.

BPwin является мощным инструментом для создания моделей, позволяющих анализировать, документировать и планировать изменения сложных бизнес-процессов. BPwin предлагает средство для сбора всей необходимой информации о работе предприятия и графического изображения этой информации в виде целостной и непротиворечивой модели. Причем, поскольку модель является некоторым графическим представлением действительности, можно утверждать, что человек вернулся к своему излюбленному средству документирования бизнес-процессов - к рисунку. Но возвращение это произошло на новом уровне - целостность и непротиворечивость модели-рисунка (качества, о которых раньше не было и речи) гарантируются рядом методологий и нотаций, которым следуют создатели модели. BPwin поддерживает три таких методологии: IDEF0, DFD и IDEF3, позволяющие анализировать ваш бизнес с трех ключевых точек зрения:

С точки зрения функциональности системы. В рамках методологии IDEF0(Integration Definition for Function Modeling) бизнес-процесс представляется в виде набора элементов-работ, которые взаимодействуют между собой, а также показывается информационные, людские и производственные ресурсы, потребляемые каждой работой.

С точки зрения потоков информации (документооборота) в системе. Диаграммы DFD (Data Flow Diagramming) могут дополнить то, что уже отражено в модели IDEF3, поскольку они описывают потоки данных, позволяя проследить, каким образом происходит обмен информацией между бизнес-функциями внутри системы. В тоже время диаграммы DFD оставляют без внимания взаимодействие между бизнес-функциями.

С точки зрения последовательности выполняемых работ. И еще более точную картину можно получить, дополнив модель диаграммами IDEF3. Этот метод привлекает внимание к очередности выполнения событий. В IDEF3 включены элементы логики, что позволяет моделировать и анализировать альтернативные сценарии развития бизнес-процесса.

Bpwin умеет проверять создаваемые модели с точки зрения синтаксиса выбранной методологии, проверяет ссылочную целостность между диаграммами, а также выполняет ряд других проверок, чтобы помочь вам создать правильную модель, а не просто рисунок. При этом сохраняются главные преимущества рисунка - простота создания и наглядность.

Rational Rose.

Среди всех фирм-производителей CASE-средств именно компания Rational Software Corp. одна из первых осознала стратегическую перспективность развития объектно-ориентированных технологий анализа и проектирования программных систем. Эта компания выступила инициатором унификации языка визуального моделирования в рамках консорциума OMG, что, в конечном итоге, привело к появлению первых версий языка UML. И эта же компания первой разработала инструментальное объектно-ориентированное CASE-средство, в котором был реализован язык UML как базовая нотация визуального моделирования.

Rational Rose - CASE-средство фирмы Rational Software Corporation (США) - предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Разработанная ими универсальная нотация для моделирования объектов (UML - Unified Modeling Language) претендует на роль стандарта в области объектно-ориентированного анализа и проектирования. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ.

Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем и других систем различной природы. UML содержит стандартный набор диаграмм и нотаций самых разнообразных видов. UML - это преемник того поколения методов ООАП, которые появились в конце 1980х и начале 1990х годов. Создание UML фактически началось в конце 1994 г., когда Гради Буч и Джеймс Рамбо начали работу по объединению их методов Booch и OMT (Object Modeling Technique) под эгидой компании Rational Software. К концу 1995 г. они создали первую спецификацию объединенного метода, названного ими Unified Method, версия 0.8. Тогда же в 1995 г. к ним присоединился создатель метода OOSE (ObjectOriented Software Engineering) Ивар Якобсон. Таким образом, UML является прямым объединением и унификацией методов Буча, Рамбо и Якобсона, однако дополняет их новыми возможностями. Главными в разработке UML были следующие цели:

1. предоставить пользователям готовый к использованию выразительный язык визуального моделирования, позволяющий им разрабатывать осмысленные модели и обмениваться ими;
2. предусмотреть механизмы расширяемости и специализации для расширения базовых концепций;
3. обеспечить независимость от конкретных языков программирования и процессов разработки;
4. обеспечить формальную основу для понимания этого языка моделирования (язык должен быть одновременно точным и доступным для понимания, без лишнего формализма);
5. стимулировать рост рынка объектно-ориентированных инструментальных средств;
6. интегрировать лучший практический опыт.

UML находится в процессе стандартизации, проводимом OMG (Object Management Group) - организацией по стандартизации в области объектноориентированных методов и технологий, в настоящее время принят в качестве стандартного языка моделирования и получил широкую поддержку в индустрии ПО. UML принят на вооружение практически всеми крупнейшими компаниями - производителями ПО (Microsoft, Oracle, IBM, HewlettPackard, Sybase и др.). Кроме того, практически все мировые производители CASE-средств, помимо IBM Rational Software, поддерживают UML в своих продуктах (Oracle Designer, Together Control Center (Borland), AllFusion Component Modeler (Computer Associates), Microsoft Visual Modeler и др.).

В результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

1. диаграммы классов;
2. диаграммы состояний;
3. диаграммы сценариев;
4. диаграммы модулей;
5. диаграммы процессов;
6. спецификации классов, объектов, атрибутов и операций
7. заготовки текстов программ;
8. модель разрабатываемой программной системы.

Последний из перечисленных документов является текстовым файлом, содержащим всю необходимую информацию о проекте (в том числе необходимую для получения всех диаграмм и спецификаций).

Microsoft Visual Studio - это программная средапоразработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом.

В комплект входят следующие основные компоненты:

1. Visual Basic.NET - для разработки приложений на VisualBasic;
2. Visual C++ - на традиционном языке C++;
3. Visual C# - на языке C# (Microsoft);
4. Visual F# - на F# (Microsoft Developer Division).

Функциональная структура среды включает в себя:

1. редактор исходного кода, который включает множество дополнительных функций, как автодополнение IntelliSense, рефракторинг кода и т. д.;
2. отладчик кода;
3. редактор форм, предназначенный для упрощённого конструирования графических интерфейсов;
4. веб-редактор;
5. дизайнер классов;
6. дизайнер схем баз данных.

Visual Studio также позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (Subversion, GIT и VisualSourceSafe), добавление новых наборов инструментов (для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения).

* + 1. Выбор языка программирования

В качестве языка программирования выбран язык C#.

C# является языком программирования, который разработан для создания множества приложений, работающих в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую С-подобным языкам.

Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и к другим полезным, хорошо спроектированным классам, что существенно ускоряет цикл разработки.

C# упрощает разработку компонентов программного обеспечения благодаря нескольким инновационным конструкциям языка, в число которых входят следующие:

1. инкапсулированные сигнатуры методов, называемые делегатами, которые поддерживают типобезопасные уведомления о событиях;
2. свойства, выступающие в роли методов доступа для закрытых переменных-членов;
3. атрибуты с декларативными метаданными о типах во время выполнения;
4. встроенные комментарии XML-документации;
5. LINQ, предлагающий встроенные возможности запросов в различных источниках данных.
   * 1. Общие сведения о языке программирования

Использованный язык C# является объектно-ориентированным языком программирования**.**

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

1.3.3.1. Элементы языка

1) Переменные

Синтаксис объявления переменных в C# представлен на рисунке 1.

ТипДанных Идентификатор;

Рис. 1

Пример объявления переменной представлен на рисунке 2.

int i;

Рис. 2

Объявить можно переменную любого действительного типа. Важно подчеркнуть, что возможности переменной определяются ее типом. Например, переменную типа bool нельзя использовать для хранения числовых значений с плавающей точкой. Кроме того, тип переменной нельзя изменять в течение срока ее существования. В частности, переменную типа int нельзя преобразовать в переменную типа char.

Все переменные в C# должны быть объявлены до их применения. Это нужно для того, чтобы уведомить, компилятор о типе данных, хранящихся в переменной, прежде чем он попытается правильно скомпилировать любой оператор, в котором используется переменная. Это позволяет также осуществлять строгий контроль типов в C#.

Начиная с версии C# 3.0, компилятору предоставляется возможность самому определить тип локальной переменной. Такая переменная называется неявно типизированной и объявляется с помощью ключевого слова var (рис. 3).

var i = 12;

Рис. 3

2) Операция присваивания

Задать значение переменной можно, в частности, с помощью оператора присваивания. Кроме того, задать начальное значение переменной можно при ее объявлении. Для этого после имени переменной указывается знак равенства и присваиваемое значение (рис.4).

int i = 42;

Рис. 4

3) Комментарии

Комментарий - строка, которая не выполняется программой. Служит для записи пояснений к коду, а также исключения (комментирования) строк кода, которые не надо выполнять, но и удалять нельзя (например, чтобы были видны проведенные изменения). В C# используются традиционные комментарии в стиле С — однострочные и многострочные (рис.5).

//Однострочный комментарий

/\*Многострочный

Комментарий\*/

Рис. 5

4) Конструкция перехода по условию

Если условие выполняется, то выполняется блок кода 1, а блок кода 2 игнорируется. Если условие не выполняется, то выполняется блок кода 2, а блок кода 1 игнорируется:

Условные конструкции - один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: if..else и switch..case.

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код (рис.6).

int num1 = 8;int num2 = 6;

if(num1 > num2)

{

    Console.WriteLine("Число {0} большечисла {1}", num1, num2);

}

else

{

    Console.WriteLine("Число {0} меньшечисла {1}", num1, num2);

}

Рис. 6

После ключевого слова if ставится условие. И если это условие выполняется, то срабатывает код, который помещен в блоке if после фигурных скобок. Если условие не выполняется, срабатывает кусок кода после слова else.

Конструкция switch/case аналогична конструкции if/else, так как позволяет обработать сразу несколько условий (рис.7).

Console.WriteLine("Нажмите Y или N");

string selection = Console.ReadLine();

switch (selection)

{

    case "Y":

        Console.WriteLine("Вынажалибукву Y");

        break;

    case "N":

        Console.WriteLine("Вынажалибукву N");

        break;

    default:

        Console.WriteLine("Вынажалинеизвестнуюбукву");

        break;

}

Рис. 7

После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора case. И если совпадение будет найдено, то будет выполняться определенный блок case.

В конце каждого блока case должен ставиться один из операторов перехода: break,goto case,return или throw. Как правило, используется оператор break. При его применении другие блоки case выполняться не будут.

Однако если нужно, чтобы, после выполнения текущего блока case выполнялся другой блок case, то можно использовать вместо break оператор goto case.

Если мы необходимо также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default, как в примере выше.

Применение оператора return позволит выйти не только из блока case, но и из вызывающего метода. То есть, если в методе Main после конструкции switch..case, в которой используется оператор return, идут какие-либо операторы и выражения, то они выполняться не будут, а метод Main завершит работу.

Оператор throw применяется для выброса ошибок.

5) Циклы

Циклы также являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

* for;
* foreach;
* while;
* do...while.

Цикл forпредставлен на рисунке 8.

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

    // действия

}

Рис. 8

Цикл foreach:

Цикл foreach предназначен для перебора элементов в контейнерах (рис.9).

foreach (тип\_данныхназвание\_переменнойin контейнер)

{

    // действия

}

Рис. 9

Цикл do:

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется (рис. 10).

do

{

// действия

}

while ([условие]);

Рис. 10

Цикл while:

В отличие от цикла do цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется (рис.11).

while ([условие])

{

    // действия

}

Рис. 11

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае используется оператор break. Если нужно, чтобы при проверке цикл не завершался, а просто переходил к следующему элементу, используется оператор continue.

6) Операторы передачи управления

В C# есть пять операторов, изменяющих естественный порядок выполнения вычислений:

* оператор безусловного перехода goto;
* оператор выхода из цикла break;
* оператор перехода к следующей итерации цикла continue;
* оператор возврата из функции return;
* оператор генерации исключения throw.

Оператор goto-метка передает управление на помеченный оператор. Метка — это обычный идентификатор, областью видимости которого является функция, в теле которой он задан. Метка должна находиться в той же области видимости, что и оператор перехода.

Оператор break используется внутри операторов цикла или выбора для перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится оператор break.

Оператор перехода к следующей итерации текущего цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова.

1.3.3.2. Способы структурирования программы

1) Процедура

Процедура или метод - часть программного модуля, предназначенная для выполнения определенной задачи.

Процедура может иметь входящие параметры, а может их не иметь – все зависит от ее назначения.

В C# определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя метода, затем список аргументов в круглых скобках и далее - тело метода в фигурных скобках (рис. 12).

[модификаторы] тип\_возвратаИмяМетода([параметры])

{

// Тело метода

}

Рис. 12

Каждый параметр состоит из имени типа параметра и имени, по которому к нему можно обратиться в теле метода. Вдобавок, если метод возвращает значение, то для указания точки выхода должен использоваться оператор возврата return вместе с возвращаемым значением.

Если метод не возвращает ничего, то в качестве типа возврата указывается void. Если же он не принимает аргументов, то все равно после имени метода должны присутствовать пустые круглые скобки. При этом включать в тело метода оператор возврата не обязательно — метод возвращает управление автоматически по достижении закрывающей фигурной скобки.

В целом, возврат из метода может произойти при двух условиях. Во-первых, когда встречается фигурная скобка, закрывающая тело метода. И во-вторых, когда выполняется оператор return. Имеются две формы оператора return: одна — для методов типа void (возврат из метода), а другая — для методов, возвращающих конкретные значения (возврат значения).

2) Функция

Тоже что и процедура, но есть одна особенность – функция возвращает некий результат, что оформляется с помощью ключевого слова «Return».

Синтаксис функции представлен на рисунке 13.

[модификаторы] тип\_возвратаИмяФункции([параметры])

{

Return [Возвращаемое значение];// Тело функции

}

Рис. 13

Программа на языке C# может состоять из одного или нескольких файлов. Каждый файл может содержать нуль или несколько пространств имен. Пространство имен может содержать типы, такие как классы, структуры, интерфейсы, перечисления и делегаты, а также другие пространства имен. На рисунке 14 приведена структура программы на C#.

usingSystem;

namespaceYourNamespace

{

classYourClass

{

}

interfaceIYourInterface

{

}

delegate int YourDelegate();

enumYourEnum

{

}

namespaceYourNestedNamespace

{

structYourStruct

{

}

}

classYourMainClass

{

static void Main(string[] args) //Программаначинаетсяздесь

{

}

}

}

Рис. 14

1.3.3.3. Средства обмена данными

Средства обмена данными позволяют создавать общий проект на несколько пользователей через подключение к локальной сети. С помощью этих механизмов можно осуществлять разработку раздельную проекта, т.е. есть возможность разработки отдельной части независимо от других частей проекта.

Для возможности совместной разработки используется расширение Team Foundation Server.

Team Foundation Server(сокр. TFS) - продукт корпорации Microsoft, представляющий собой комплексное решение, объединяющее в себе систему управления версиями, сбор данных, построение отчётов, отслеживание статусов и изменений по проекту и предназначенное для совместной работы над проектами по разработке программного обеспечения. Данный продукт доступен как в виде отдельного приложения, так и в виде серверной платформы для Visual Studio Team System (VSTS).

Team Foundation Server работает по трёхуровневой архитектуре: клиентский уровень, прикладной уровень и уровень данных. Клиентский уровень используется для создания и управления проектами, а также для доступа к хранимым и управляемым элементам проекта. На этом уровне TFS не содержит никаких пользовательских интерфейсов, но предоставляетвеб-сервисы, которые могут быть использованы клиентскими приложениями для самостоятельной интеграции в функциональность TFS. Эти веб-сервисы используются такими приложениями, как Visual Studio Team System для применения TFS в качестве серверной инфраструктуры хранилища информации или выделенного TFS управления приложениями, наподобие включенного приложения Team Foundation Client. Сами веб-сервисы находятся на прикладном уровне. Прикладной уровень также включает в себя веб-портал и репозиторий (хранилище) документации, поддерживаемые Windows SharePoint Services. Веб-портал, называемый Team Project Portal (портал командного проекта), выступает в роли центра взаимодействия для проектов, управляемых TFS. Репозиторий документов используется как для элементов проекта, так и для отслеживания ревизий (документирование изменений), а также для накопления и обработки данных и генерации отчётов. Уровень данных, основывающийся в первую очередь на установленномSQL Server 2005 Standard Edition, обеспечивает сервисы постоянного хранения данных для репозитория документов. Уровень данных и уровень приложений могут существовать на различных физических или виртуальных серверах при использовании Windows Server 2003 или более специализированных версий. Уровень данных не взаимодействует с клиентским уровнем напрямую, только через прикладной уровень.

1.3.3.4. Встроенные элементы

Встроенные процедуры и функции представлены в таблице 1.

Таблица 1 – Встроенные процедуры и функции языка C#.

|  |  |
| --- | --- |
| Процедура/Функция | Описание процедуры/функции |
| Res = Math.Round(Cost\*0.5, 2); | Округление |
| Res = Math.Floor(Cost); | Возвращает наибольшее целое число, которое не больше заданной переменной |
| var DbResult = db.ProcedureName(); | Передача данных из процедуры |
| db.TableName.ToList() | Передача данных из таблицы БД |
| db.TableName.Add(data); | Добавляет заданную сущность к контексту, поддерживающему набор, в добавленном состоянии, в результате чего она будет вставлена в базу данных при вызове метода SaveChanges |
| db.Entry<TableName>(data).State = System.Data.Entity.EntityState.Modified; | Изменяет выбранную сущность в контексте, поддерживающему набор, в изменённом состоянии, в результате чего она будет изменена в базе данных при вызове метода SaveChanges |
| db.TableName.Remove(data); | Помечает заданную сущность как удалённую, в результате чего она будет удалена из базы данных при вызове метода SaveChanges |
| await db.SaveChangesAsync(); | Асинхронно сохраняет все изменения основной базы данных, произведённые в контексте |
| base.Dispose(); | Устраняет неуправляемые и, если указано, управляемые ресурсы |

1.3.3.5. Средства отладки программы

В случае если на этапе написания какого-либо кода программист допускает ошибку, которая может быть проанализирована ещё ДО компиляции и запуска программы – Visual Studio об этом сообщит в окне «Список ошибок». В основном на этапе разработки фиксируются, если можно так выразиться, элементарные ошибки: пропуск запятой, несоответствие видимости классов, полей и т.д., то ошибки, которые могут быть исправлены быстро.

Довольно часто могут возникать ситуации, когда на этапе выполнения программы в какой-либо метод передается неверный параметр, скажем, в качестве пути к локальной базе данных, может случайно передаться пустая строка – в этом случае обязательно возникнет исключительная ситуация. Такие моменты студия также фиксирует, если приложение запускается из среды IDE. Visual Studio выводит окно, в котором дается небольшое пояснение об исключительной ситуации. Здесь можно:

* просмотреть сведения об ошибке в справочных материалах;
* поискать решение проблемы в сети;
* просмотреть сведения об ошибке;
* скопировать сведения исключения в буфер обмена.

При выборе опции «Просмотр сведений» откроется ещё одно окно, в котором сдержится подробная информация об исключительной ситуации.

Точка останова или breakpoint – это преднамеренная остановка выполнения программы, при котором выполняется вывод отладчика.

Чтобы установить breakpoint, необходимо навести курсор на нужную строку, и выбрать в меню «Отладка – Точка останова» или нажать [F9]. При этом строка с точкой останова будет помечена. Как только выполнение программы дойдет до точки останова – выполнение прервется и запустится отладчик. Во время выполнения отладчика можно проверить значения переменных.

* 1. Содержательная постановка задачи
     1. Организационно-экономическая сущность задачи
        1. Назначение задачи

Задача предназначена для всех социальных групп и сфер человеческой деятельности, где есть необходимость в контроле знаний.

* + - 1. Цели решения задачи

Автоматизация контроля знаний для всех социальных групп и сфер человеческой деятельности, где есть необходимость в контроле знаний.

* + - 1. Переодичность решения задачи

Контроль знаний – это непрерывная или периодически повторяющеяся задача.

* + - 1. Информационная связь с другими задачами

Задача связана с задачей обучения людей необходимым знаниям, навыкам и умениям.

* + 1. Описательная модель задачи

Описательная модель задачи, представленная в виде интеллектуальной карты, приведена в приложении 1.

* + 1. Математическая модель задачи
    2. Описание входной информации
       1. Описание входных документов

Входные документы представлены в приложении 2.

* + - 1. Описание входных данных

Входными данными являются «Тесты», «Тестируемые», чьи схемы данных представлены в пункте 2.1.1.

* + 1. Описание выходной информации
       1. Описание выходных документов

Выходные документы представлены в приложении 3.

* + - 1. Описание выходных данных

Выходными данными являются результаты тестирования и тесты в готовом для печати виде

* + 1. Логическая модель задачи
       1. Функциональные диаграммы

Функциональная диаграмма приведена в приложении 4.

* + - 1. Диаграммы потоков данных

Диаграмма потоков данных приведена в приложении 5.

* + 1. Требования к программе
       1. Функциональные требования

Программа должна обеспечить возможность выполнения следующих функций:

1. хранение информации в базе данных;
2. предоставление возможности создания и изменения тестов;
3. прохождение тестирования;
4. печать документов в Microsoft Word и Microsoft Excel;
5. настройка программы.
   * + 1. Требования к надёжности

Программа должна предусматривать:

1. контроль вводимой информации;
2. блокировку некорректных действий пользователя при работе с программой;
3. авторизацию доступа к программе.
   * + 1. Требования к информативности

Программа должна иметь понятную справочную систему и подписи к элементам управления.

* + - 1. Требования к удобству эксплуатации

Программа должна:

1. иметь дружественный пользовательский интерфейс со свободной навигацией,
2. реализацией диалоговых окон при вводе и выводе информации.
3. работать на IBM совместимых ПК работать под управлением семейства операционных систем Windows 7 (SP1)/8/8.1/10.