ФГБОУ ВО «Саратовский национальный исследовательский университет имени Н.Г. Чернышевского»

МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

работа выполнена в рамках курсового проекта по дисциплине «Структуры и алгоритмы компьютерной обработки данных»

Выполнила: Лукашова М.А., студентка факультета компьютерных наук и информационных технологий

Научный руководитель: Кудрина Е.В., доцент кафедры информатики и программирования

Саратов 2016

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Как все созданное человеком, тестирование программного обеспечения (ПО) проходило несколько этапов развития. Начальное понятие - "исчерпывающее" тестирование, включало проверку кода с использованием всех возможных входных данных. Оно было отклонено, по причинам невозможности провести полное тестирование, а как следствие и поиск проблемы в архитектуре и спецификациях.

В 70-х годах прошлого века возник, так называемый, "парадокс тестирования", в его основе лежат два противоположных понятия: с одной стороны, тестирование позволяет доказать правильность работы продукта, а с другой - находит ошибки в программном обеспечении, показывая отрицательную работу продукта. Второе понятие более значимо, так как позволяет не игнорировать недостатки ПО.

До начала 80-х тестирование относилось только к скомпилированной системе, далее элементы тестирования внедряются во все этапы жизненного цикла разработки. Появляется автоматическое тестирование и понятия позволяющие сделать переход от тестирования к обеспечению качества, включающего весь цикл разработки ПО.

Все это привело к следующему определению тестирования:: "Тестирование - это процесс исполнения программы с целью обнаружения ошибок"[6], а так же к разработке классификации видов тестирования разделенных на группы: по объекту тестирования, по знанию системы, по степени автоматизированности, по степени изолированности компонентов, по времени проведения тестирования, по признаку позитивности сценариев, по степени подготовленности к тестированию и т. д.[14].

Рост конкуренции на рынке ПО потребовал внимания к качеству продуктов. Сейчас компьютеризации подвержены практически все области жизни и вопрос о качестве приобретает особую важность. Например, сегодня ПО управляет оборудованием в больницах, диспетчерскими системами в

аэропортах, атомными реакторами, космическими кораблями, и для всего этого требуется комфортная, устойчивая и бесперебойная работа.

Сегодня тестирование стало обязательной частью процесса производства ПО. Оно направлено на обнаружение и устранение как можно большего числа ошибок. Следствием такой деятельности является повышение качества ПО по всем его характеристикам, это и определяет цель моей работы.

ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Место и роль тестирования в различных моделях разработки ПО

Процесс имеющий отношение к созданию того или иного продукта, часто называется жизненным циклом. Стандартные модели разработки ПО - это каскадная модель, каскадная модель с промежуточным контролем, итеративная модель, V-образная модель.

1.1.1 Каскадная модель жизненного цикла

В 1970 году первой была использована каскадная (или водопадная) модель жизненного цикла программного продукта, изображенная на рисунок 1.1. Её применяют когда требования строго прописаны.

Рисунок 1.1 Каскадная модель жизненного цикла

Преимущества данной модели:

- Каждая стадия представляет собой законченный результат: следующая стадия начинается, когда заканчивается предыдущая;
- Фиксированный набор стадий;
- Оценка качества на каждом этапе;

Недостатки:

- Фаза должна быть закончена;
- Отсутствуют обратные связи между этапами, нельзя вернуться на стадию выше;

- Результат тестирования можно увидеть только в конце разработки;
- Исправление ошибок происходит лишь во время тестирования;
- Тяжело реагировать на изменения требований;

В книге [10] утверждается, что главным минусом каскадной модели является то, что она рассматривает программное обеспечение не как процесс решения задачи. Она представляет собой конвейерный принцип разработки программного обеспечения, по данным которого компонент сначала разрабатывается, а затем многократно копируются. Но нельзя построить точную модель процесса разработки программного продукта в виде некоторого набора независимых стадий, что подразумевает каскадная модель[1]. Некоторые недостатки в этой модели были исправлены в следующей модели.

1.1.2 Каскадная модель с промежуточным контролем (или водоворот)

Эта модель основывается на предыдущей, но предполагает обратные связи с каждым этапом жизненного цикла см. рисунок 1.2.

Рисунок 1.2 Каскадная модель с промежуточным контролем.

Рассмотрим характеристики данной модели:

- исправление ошибок осуществляется на каждом этапе, сразу при нахождении проблемы;
- этапы пересекаются во времени, из-за существования обратных связей. При первом проходе следующий этап не начнется, пока не

закончится предыдущий, но как только обнаружена ошибка, осуществляется возврат к предыдущим этапам, которые повлекли ошибку. Таким образом, фактически этапы оказываются растянутыми во времени;

- результат получаем только в конце разработки (как и в каскадной модели);

В итоге мы получаем весомый недостаток: десяти кратное увеличение затрат на разработку, а признаком получения результата является оптимальное качество продукта, то есть когда особо острые для клиента ошибки устранены, а с наличием маловажных, для работы системы, ошибок клиент согласился — данные ошибки описаны в документации и практически становятся особенностями системы. Для возможности возвращения на необходимый и уже завершенный этап, была разработана итеративная модель.

1.1.3 Итеративная модель

Итеративная модель предусматривает разбиение разрабатываемого продукта на набор частей, которые создаются с помощью ряда последовательных проходов всех этапов или их части (рисунок 1.3).

Рисунок 1.3 Итеративная модель

Проведя параллель можно сказать что, итеративная модель, это каскадная модель с возможностью возврата на завершенный этап и при необходимости пересмотр его результатов. Итеративная модель предполагает, что различные

типы работ не стоят в строгом порядке, а выполняются по мере необходимости, подчас повторяются, до тех пор, пока мы не получим требуемый результат. Из-за возможности достаточно быстро реагировать на изменения системы, эта модель добавляет лишние трудности в отслеживании хода проекта и его управление. В итоге сложнее предсказать сроки и ресурсы, необходимые для качественного результата.

1.1.4 V-образная модель

V-образная модель позволяет налаживать обмен данными между группами разработчиков и тестировщиков, что создает более эффективную команду по созданию продукта см. рисунок 1.4. Левую сторону V-модели составляют: анализ и проектирование. Правую сторону составляет тестирование, а самая нижняя точка—это кодирование. V- модель является одной из практик экстремального программирования, и предполагает систематическое тестирование продукта во время разработки.

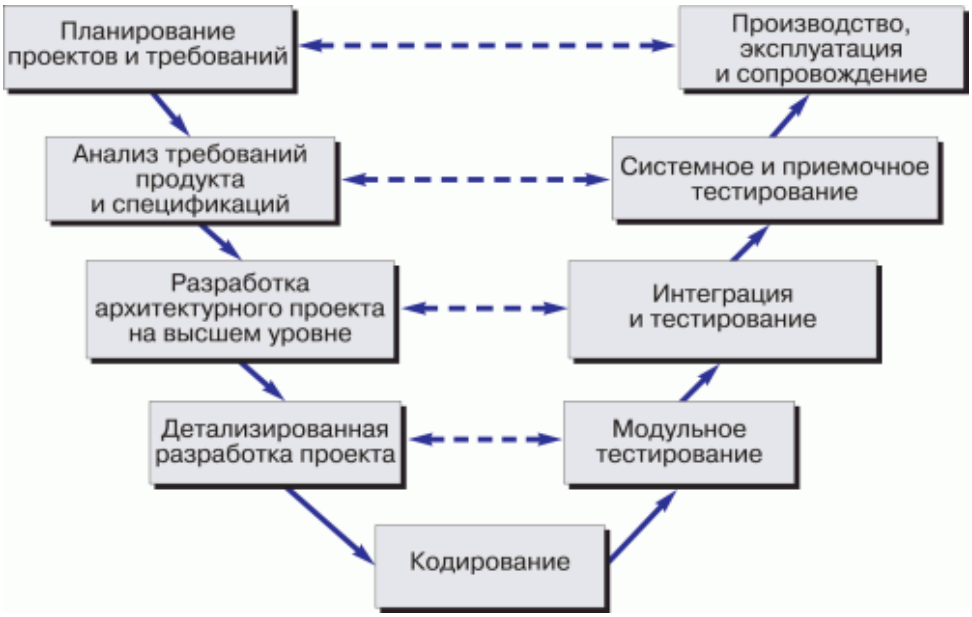

Рисунок 1.4 V-образная модель

Пунктирные линии обозначают отношения между видом тестовой деятельности и видом проектной деятельности. Таким образом, одна из задач V-диаграммы состоит в том, чтобы показать, какова цель видов тестовой деятельности в терминах контроля и аттестации на ранних стадиях разработки.

Между потоками разработки и тестирования этого процесса должен надежно поддерживаться обмен данными, и каждая группа должна участвовать в некоторых контрольных действиях другой группы[1].

Таким образом, можно сделать вывод,— чем раньше найден дефект, тем меньше затрат на его исправление. Так, в статье [9] показано, что стоимость исправления дефекта после ввода системы в эксплуатацию вдвое превышает аналогичную стоимость на стадии тестирования продукта и более чем в тысячу раз — в период выработки требований к продукту. Эту зависимость можно проследить на рисунке 1.5, а одновременно с увеличением стоимости исправления, увеличивается и временя разработки продукта.

Рисунок 1.5 Стоимость качества.

1.2 Обзор методов тестирования программного обеспечения

По толковому словарю Ушакова,— "Ошибка – это неправильность в действиях, поступках, высказываниях, мыслях, погрешность." Это достаточно общее и универсальное определение, в нашем случае встречаются, программные ошибки, а также ошибки связанные с разработкой программного продукта. Рассмотрим классификацию ошибок.

По времени появления ошибки разделяются на три вида[3,6]:

- **Структурные ошибки** могут появиться при наборе кода программы. Например, к этому типу ошибок относятся: расхождение числа открывающих скобок и числа закрывающих, неправильное употребление синтаксических знаков и т. д.
- **Ошибки компиляции** обнаруживаются при общей компиляции приложения, т.е. возникают из-за ошибок в тексте кода. К ним можно отнести ошибки в синтаксисе, употребление несуществующих объектов или свойств, методов у объектов.
- **Ошибки периода выполнения** возникают, на этапе выполнения программы и компилятор обнаруживает, что оператор делает попытку выполнить недопустимое или невозможное действие. Примером может послужить, деление на ноль.

По степени нарушения логики на:

- **Синтаксические ошибки –** заключаются в нарушении грамматических правил языка. К примеру, пропуск нужных скобок, отсутствие инициализации массива и т.д. Ошибки данного типа обнаруживаются компилятором.
- **Семантические ошибки** — это нарушении порядка операторов, параметров функций и использования выражений. Эти ошибки также обнаруживаются компилятором. Но некоторые исследователи считают семантические ошибки частью следующей группы.

- **Прагматические ошибки** (или логические) – заключаются в ошибочной логике алгоритма. Являются сложно обнаруживаемыми, а компилятор определяет только следствие прагматической ошибки.

А также ошибки можно разделить на классы:

- **Ошибка адресации** — например, выход за пределы участка памяти.
- **Ошибка ввода-вывода** — возникает в процессе обмена данными между устройствами.
- **Ошибка вычисления** — появляется при выполнении арифметических операций (например, деление на ноль). Могут возникнуть из-за некорректных исходных данных и реализованных формул, неправильного применения операций вычислений или операндов.
- **Ошибка интерфейса** — ошибка, вызванная несоответствием характеристик фактических и формальных параметров, относится к ошибкам взаимосвязи отдельных элементов друг с другом.
- **Ошибка обращения к данным** — появляется при обращении программы к данным (например, выход индекса за пределы массива).
- **Ошибка описания данных** — ошибка, допущенная в ходе описания данных.

А теперь исследуем методы тестирования используемые для выявления ошибок, описанных выше. Методы тестирования делятся на группы[3, 6 ,14]:

По объекту тестирования:

- **Функциональное тестирование** – направлено на проверку корректности работы функциональности приложения.
- **Избыточное тестирование** – тестирование приложения со всеми условиями выполнения. Может применяться для проверки простых компонентов.
- **Тестирование производительности** – исследует показатели скорости реакции приложения на внешние воздействия при нагрузке.

- **Тестирование данных и баз данных** – направлено на исследование характеристик данных, например, полнота, непротиворечивость, цельность, структурированность и т.д. (Находит ошибки: обращение к данным, описания данных, ввода-вывода).
- **Тестирование удобства использования** – настроено на исследование того, насколько конечному пользователю понятно, как работать с продуктом, а также на то, насколько ему приятно использовать продукт.
- **Тестирование интерфейса пользователя** – направлено на проверку интерфейсов приложения или его компонентов. (Обнаруживает ошибки интерфейса).
- **Тестирование безопасности** – проверяет способности приложения противостоять хакерским попыткам получения доступа к данным или функциям.
- **Тестирование локализации** – исследует готовность продукта к работе с использованием различных языков и с учетом различных национальных и культурных особенностей.
- **Тестирование совместимости** – направлено на проверку способности приложения, работать в указанном окружении.

По знанию системы:

- **Тестирование чёрного ящика** – доступа к коду нет, тестировщик оказывает на продукт воздействия и проверяет реакцию тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы пользователи. В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов выступает документация и общий здравый смысл.
- **Тестирование белого ящика** – доступ к коду есть, разработчик теста имеет доступ к исходному коду программ и может писать код,

который связан с библиотеками тестируемого ПО. При тестировании белого ящика используются метрики покрытия кода.

- **Тестирование серого ящика** – имеется доступ к части кода.

По степени автоматизированности:

- **Ручное тестирование** – тест-кейсы выполняет человек.
- **Автоматизированное тестирование** – тест-кейсы частично или полностью выполняет специальное инструментальное средство, что позволяет исключить человека из выполнения некоторых задач в процессе тестирования.

По уровню тестирования:

- **Компонентное (модульное) тестирование** – проверяются отдельные небольшие части приложения, например, отдельный класс, функция или небольшие библиотеки.
- **Интеграционное тестирование** – проверяется взаимодействие между несколькими частями приложения, каждая из которых проверена отдельно на стадии модульного тестирования. При наличии резерва времени на данной стадии тестирование ведётся итерационно, с постепенным подключением последующих подсистем.
- **Системное тестирование** – приложение проверяется как единое целое. Выявляются дефекты "на стыках" компонентов, а также есть возможность полноценного взаимодействия с программой с точки зрения конечного пользователя.

По привлечению конечных пользователей:

- **Альфа тестирование** – выполняется внутри организации-разработчика с возможным привлечением конечных пользователей. Суть этого вида в том, что продукт достаточно готов для проверки на целевую аудиторию, но недостаточно готов для вывода тестирования за пределы организации-разработчика.

- **Бета тестирование –** выполняется вне организации-разработчика с активным привлечением конечных пользователей. Суть этого вида в том, что продукт уже достаточно стабилен, и его можно открыто показывать внешним пользователям, но проблемы могут остаться, и для их выявления нужна обратная связь от реальных пользователей.
- **Гамма тестирование** – финальная стадия тестирования перед выпуском продукта, направленная на исправление незначительных дефектов, обнаруженных в бета тестировании. Суть этого вида в том, что продукт уже почти готов, и сейчас обратная связь от реальных пользователей используется для устранения последних недоработок.

По времени проведения тестирования:

- **Приемочное тестирование** – проверяет приложение с точки зрения конечного пользователя и вынесения решения о том, принимает ли заказчик работу у исполнителя.
- **Регрессионное тестирование** – направленно на проверку факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями в приложении или среде его функционирования.

По признаку позитивности сценариев:

- **Позитивное тестирование** – все действия с приложением выполняются строго по инструкции без недопустимых действий, некорректных данных и т.д. Можно образно сказать, что приложение исследуется в "тепличных условиях".
- **Негативное тестирование** – в работе с приложением выполняются некорректные операции и используются данные, потенциально приводящие к ошибкам.

По запуску кода на исполнение:

- **Статическое тестирование** – без запуска кода на исполнение. (Обнаруживает структурные ошибки, а также сюда относится

проверка: документация, графических прототипов, параметров среды исполнения).

- **Динамическое тестирование** – с запуском кода, проверяется реальное поведение программы. (Обнаруживает: ошибки компиляции, ошибки периода выполнения, все ошибки нарушения логики).

Это только часть большой классификации методов тестирования, существует также несколько альтернативных классификаций. Методов достаточно много, и можно сделать вывод, что все виды ошибок можно легко обнаружить, но на деле все решает опыт, навыки, и даже иногда интуиция.

ГЛАВА 2 ПРИМЕНЕНИЕ МЕТОДОВ ТЕСТИРОВАНИЕ ПО НА ПРАКТИКЕ

2.1 Постановка задачи

1) Создать абстрактный класс Клиент с методами, позволяющими вывести на экран информацию о клиентах банка, а также определить соответствие клиента критерию поиска.
2) Создать производные классы:
    - Вкладчик (фамилия, дата открытия вклада, размер вклада, процент по вкладу),
    - Кредитор (фамилия, дата выдачи кредита, размер кредита, процент по кредиту, остаток долга),
    - Организация (название, дата открытия счета, номер счета, сумма на счету),

   со своими методами вывода информации на экран и определения соответствия дате (открытия вклада, выдаче кредита, открытия счета).
3) Создать базу (массив) из n клиентов, вывести полную информацию из базы на экран, а также организовать поиск клиентов, начавших сотрудничать с банком с заданной даты.
4) В абстрактном классе Клиент реализовать метод CompareTo так, чтобы можно было отсортировать базу данных о клиентах банка по дате открытия их счета.

2.2 Описание приложения

Приложение было создано на основе трехуровневой архитектуры, управляемый код написан на языке C#, с пользовательским интерфейсом Windows Forms. Для разработки использовалась каскадная модель, но имелся существенный недостаток, требования к приложению были весьма формальными, что в дальнейшем и отразилось на тестировании. На блок схеме изображена архитектура приложения, подробное описание приведено ниже:

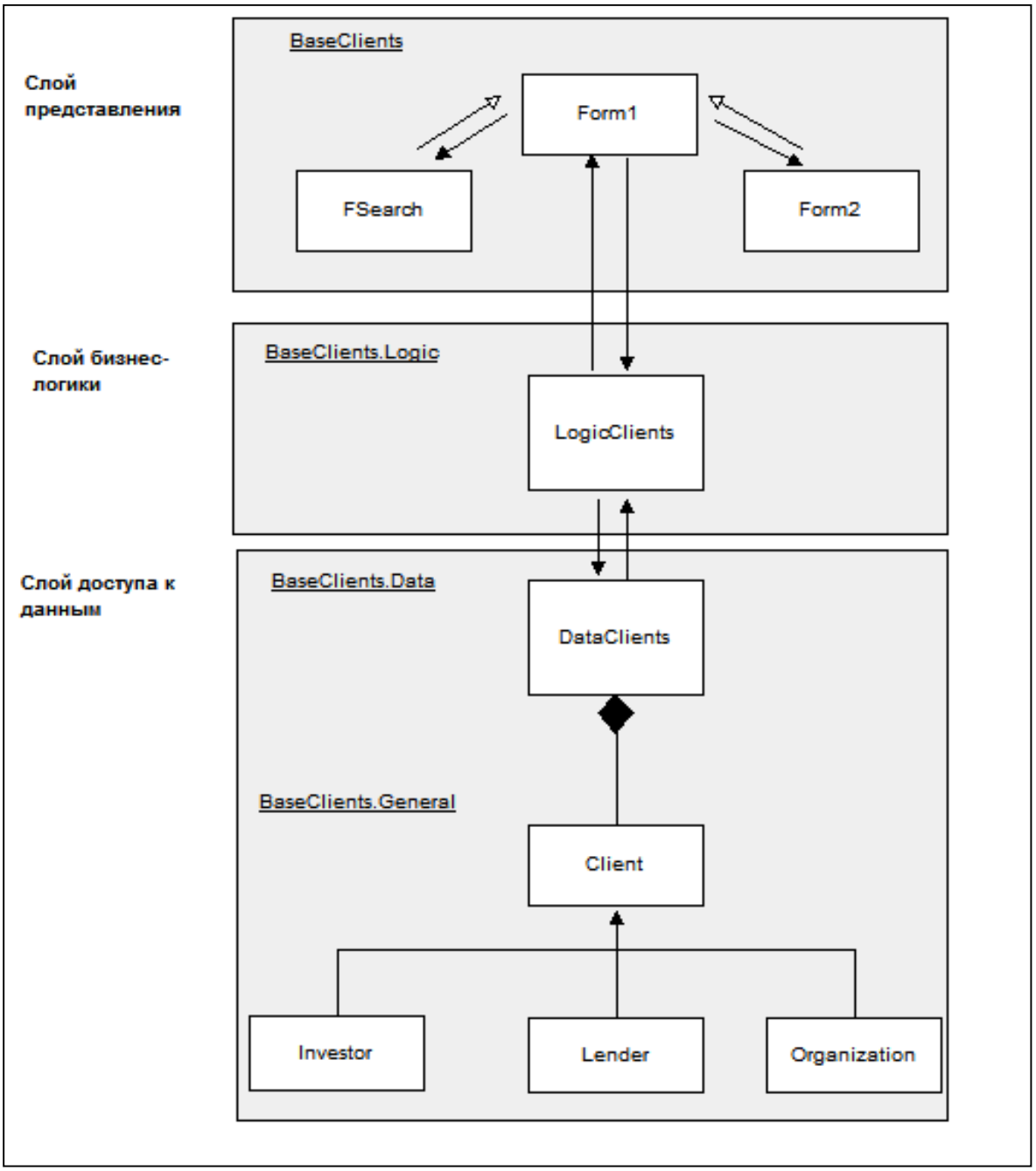

Рисунок 2.1 Архитектура приложение "База клиентов банка"

Уровень клиента это интерфейс windows forms:

- Form1– это главная форма, с меню для вызова функций и полем для вывода информации о клиентах.
- Form2 – форма для добавления клиента, которая изменяется в зависимости от типа клиента.
- FSearch – форма для поиска клиентов по дате открытие счета.

Уровень сервер-приложения это класс LogicClients. Класс DataClients является логикой погруженной в базу данных, этот класс использует композицию абстрактного класс Client, от которого, в свою очередь, наследуются классы: Investor, Lender, Organization.

Каждый производный класс содержит функции:

- string ToString() – создает строку с полной информацией о клиенте.
- bool Conform(string a) – возвращает true, если даты совпадают.
- string Write() – возвращает строку для записи в файл.

2.3 Описание методики тестирования приложения в процессе разработки

В рамках данной курсовой работы выполнялись следующие виды тестирования:

- функциональное тестирование
- модульное тестирование
- тестирование интерфейса.

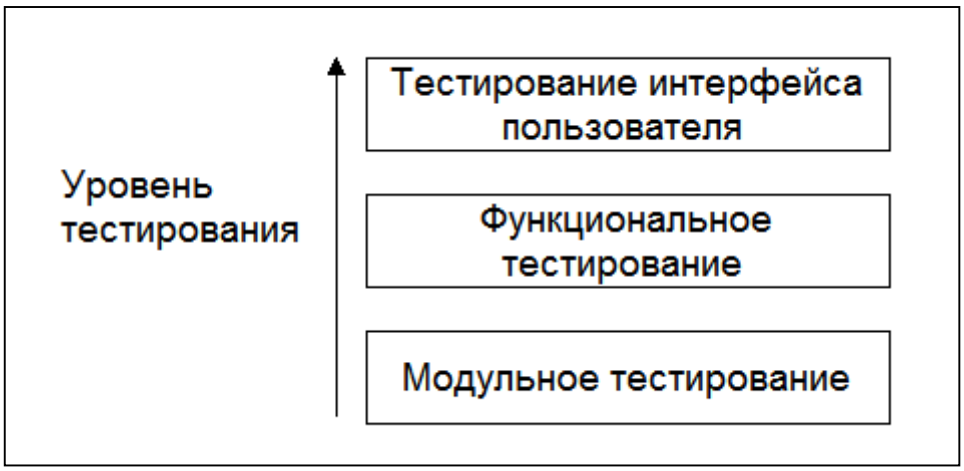

Рисунок 2.2 Этапы тестирования ПО

2.3.1 Модульное тестирование

Модульное тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Например, отдельный класс, функция или небольшие библиотеки. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов[3].

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 появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. Проводится 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.

Модульное тестирование проводилось с использованием встроенных средств Visual Studio[13]. Посмотрим на структуру классов в проекте модульного теста.

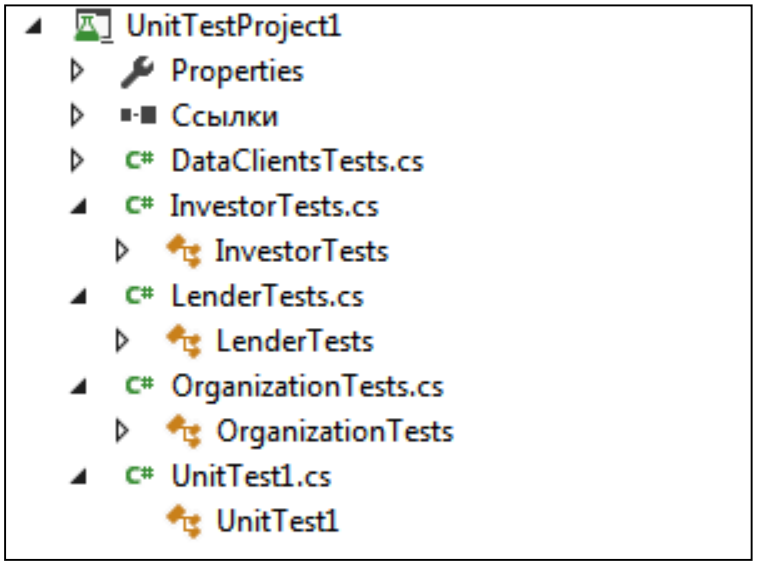

Рисунок 2.3 Структура проекта модульного теста.

Тестами покрыт слой данных, все тесты являлись позитивными, т.е. на вход подаются правильные данные. Для того чтобы во время теста не происходило вызова базы данных создаются "заглушки", которые имитируют её работу используя файловую систему[1]. В функциях возвращающих значение тест проверяет на равенство, результат выполнения функции и эталонное значение, которое должно получиться. В функциях, не возвращающих значение, проверяется на успешный вызов функции без exception. Все созданные тесты прошли успешно см. рисунок 2.4. Код тестов представлен в Приложении Д.

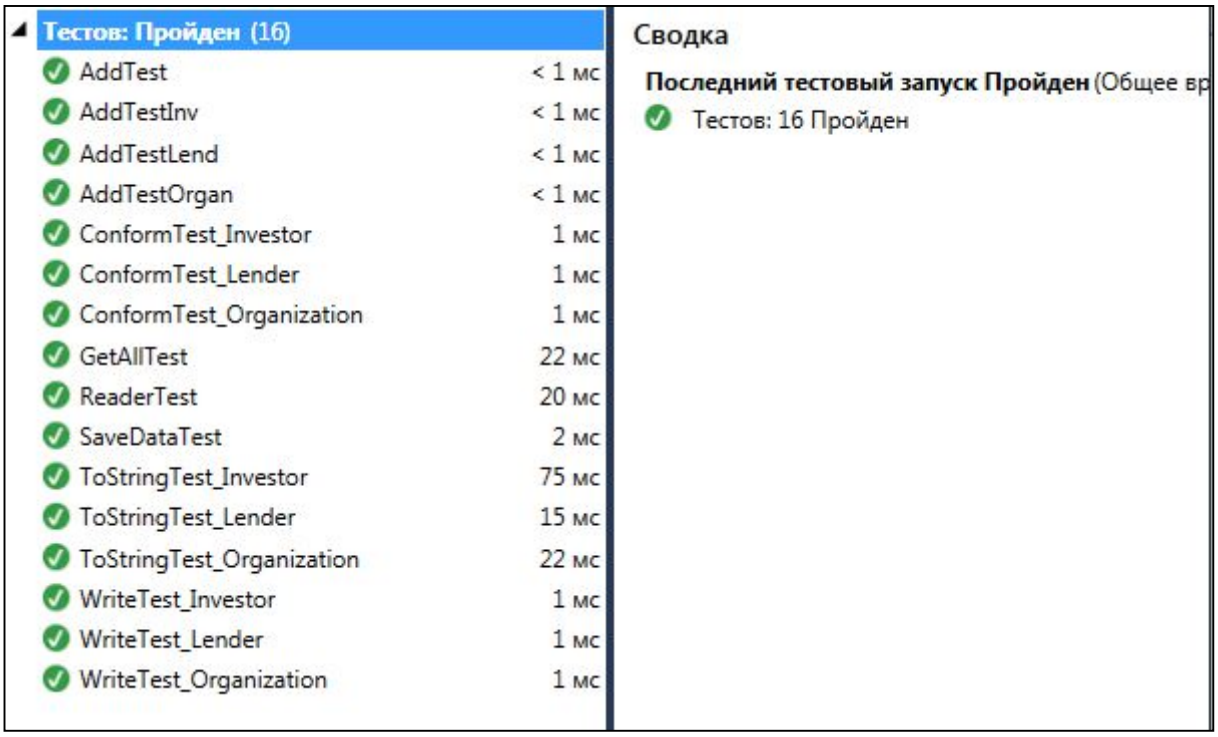

Рисунок 2.4 Модульное тестирование

2.3.2 Функциональное тестирование

Функциональное тестирование – направлено на проверку корректности работы функциональности приложения. Если говорить проще, то при функциональном тестировании проверяется, выполняет ли программный продукт все функции, которые должен[3].

Обычно разрабатывается документ программа и методика испытаний. Этот документ содержит перечень сценариев тестирования программного продукта с подробным описанием шагов. Программа и методика испытаний обязана имитировать эксплуатацию программного продукта в реальном режиме. Это означает, что сценарий тестирования должен быть построен на основе анализа операций, которые будут выполнять будущие пользователи системы[7].

Функциональные тесты проводились в ручную, необходимо было проверить поведение системы в соответствии с требованиями. В каждой форме, каждое поле и кнопка проверялись на реакцию на нажатие клавиш клавиатуры. Набор тестов создавался на основе эквивалентного разбиения.

Не все тесты прошли успешно, важные результаты тестирования приведены в виде таблиц в Приложении Е, в таблице содержатся сведения о результате прохождения теста. Такой тип отчета не дает полной картины об ошибке, но является довольно наглядным, по правилам создаются более подробные таблицы со ссылками на документы.

2.3.3 Тестирование интерфейса пользователя

Тестирование интерфейса пользователя – направлено на проверку интерфейсов приложения или его компонентов, обнаруживает ошибки интерфейса. Важно! Тестирование интерфейса пользователя и тестирование удобства использования — не одно и то же! Например, удобный интерфейс может работать некорректно, а корректно работающий интерфейс может быть неудобным[3].

Тестирование интерфейса состоит так же из анализа требований, разработки планов и сценариев и составление отчетов о проблемах в случае несовпадения поведения системы и требований либо в случае отсутствия требований на отдельные интерфейсные элементы[11]. Существует два типа требований к пользовательскому интерфейсу:

- Требования к внешнему виду пользовательского интерфейса и формам взаимодействия с пользователем.
- Требования по доступу к внутренней функциональности системы при помощи пользовательского интерфейса.

Схема тестирования интерфейса показана на рисунке 2.5. Стрелки в верхней части схемы означают, что контрольные тесты применяются не к отдельным компонентам, а к подсистемам, полученным в результате комбинирования этих компонентов.

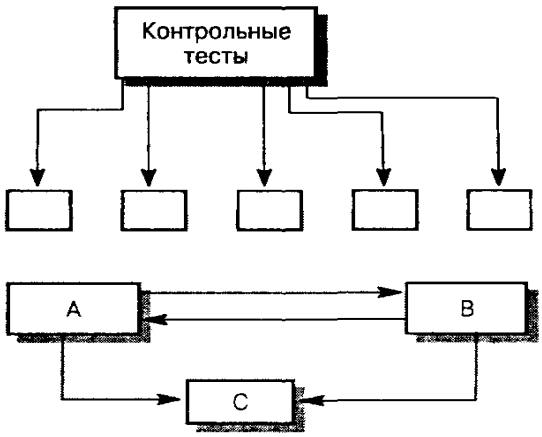

Рисунок 2.5Тестирование интерфейса

Данное тестирование также может проводиться с использованием программных средств, но у ручного метода есть ряд преимуществ:

- Поиск "косметических" дефектов – то есть дефекты, которые видны только человеческому глазу.
- Анализ успешности прохождения теста будет выполняться не по формальным признакам, а согласно человеческому восприятию.

Но у обоих методов также есть и ряд недостатков[11]:

Минусы ручного метода:

- Требуются значительные человеческие и временные ресурсы.
- При проведении регрессионного тестирования и вообще любого повторного тестирования - на каждой итерации повторного тестирования пользовательского интерфейса требуется участие тестировщика-оператора.

Минусы автоматического метода:

- Анализ успешности прохождения теста будет выполняться по формальным признакам.
- Невозможность поиска «косметических» дефектов.
- Высокая стоимость поддержки по сравнению с «обычными» функциональными тестами.

Приложение "База клиентов банка" проходило тестирование интерфейса пользователя в ручную.

Обнаружились такие дефекты:

- во всех формах добавления клиента, при вводе клиента с двумя и более неправильными полями высвечивается сообщение об ошибке на каждом поле, как и должно быть, но если изменить поля и оставить одно не правильное сообщение об ошибке на уже измененных и верных полях остается.
- у всех полей нет максимально допустимого количества символов для ввода в поле.
- если задать неверный формат даты, то во время поиск, приложение просто выведет, что клиентов с такой датой нет.
- при повторном поиске по той же дате поле Texbox не обновляется.
- во время добавления кредитора с остатком долга большим, чем сумма кредита, программа обнуляет это значение у соответствующего клиента.

ЗАКЛЮЧЕНИЕ

В работе были рассмотрены основные этапы жизненного цикла ПО, ошибки в приложениях и классификация методов тестирования. Также для практической части было создано приложение "База клиентов банка". На примере это приложения были проведены: модульное, функциональное тестирование и тестирование интерфейса пользователя. В момент тестирования были обнаружены ошибки:

- Не выводятся сообщения об ошибках в случаях: попытки добавления в поле неверного знака и оставленного пустого поля.
- Во всех формах добавления клиента, после изменения неправильных полей сообщение об ошибке на верных полях остается.
- У всех полей нет максимально допустимого количества символов для ввода в поле.
- Если задать неверный формат даты, то во время поиск, приложение просто выведет, что клиентов с такой датой нет.
- При повторном поиске по той же дате поле Texbox не обновляется.
- Во время добавления кредитора с остатком долга большим, чем сумма кредита, программа обнуляет это значение у соответствующего клиента.

Данные ошибки не критичны, но все же существенны для эксплуатации приложения конечным пользователем. Все дефекты в приложения были устранены, но для наглядности в Приложениях А, Б, В, Г приведен не исправленный код программы.

В моей работе ошибки, обнаруженные на этапе тестирования, были следствием отсутствия разработанных требований к приложению. Поэтому составление полноценных и исчерпывающих требований к продукту, один из залогов успешного тестирования, что и будет определять цель моей следующей курсовой работы.

СПИСОК ЛИТЕРАТУРЫ

1. *Калбертсон Роберт, Браун Крис, Кобб Гэри.* Быстрое тестирование. — М.: «Вильямс», 2002. — 374 с.
2. *Котляров В.П., Коликова Т.В.*. Основы тестирования программного обеспечения: Учебное пособие — М.: Интернет-Университет Информационных Технологий — БИНОМ. Лаборатория знаний, 2006. — 285с.
3. *Куликов Святослав.* Тестирование программного обеспечения. — Базовый курс, версия книги 1.0.8 от 18.03.2016 – EPAM Systems – 2015-2016 — 289 с.
4. *Лаврищева Е.М., Петрухин В.А.* Методы и средства инженерии программного обеспечения, - Учебник. Московский физико-технический институт (государственный университет), 2006 — 304 с.
5. *Майерс Гленфорд, Баджетт Том, Сандлер Кори.* Искусство тестирования программ, 3-е издание = The Art of Software Testing, 3rd Edition. — М.: «Диалектика», 2012. — 272 с.
6. *Степанченко И. В*. МЕТОДЫ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: Учеб. пособие / ВолгГТУ, Волгоград, 2006. – 74 с.
7. *Тамре Луиза.* Введение в тестирование программного обеспечения.: Пер. с англ. — М.: Издательский дом «Вильямс», 2003. —368 с.
8. *Троелсен Эндрю.* Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд.: Пер. с англ. — М.: ООО «И.Д. Вильямс», 2013, — 1312 с.
9. Статья B. Boehm and V. Basili «Software Defect Reduction Top 10 List» (IEEE Computer, IEEE Computer Society, Vol. 34, No.1, January 2001, pp. 135-137.)
10. *Shari Lawrence Pfleeger and Joanne M. Atlee.* Software Engineering: Theory and Practice ,4th ed. — NJ: Prentice Hall, 2009. — 792 p.
11. *Боциев А.Я., Виценко А.Ю., Крюков А.К. и др.* Тестирование графического интерфейса пользователя. Тренинг Intel Delta Course

[Электронный ресурс]. – Режим доступа: http://delta-course.org/docs/Delta2014S-T2-L7.pdf – Дата обращения: 13.04.2016.

12. Гибкая методология разработки программного обеспечения – Методичка Microsoft Visual Studio Team System 2008 Режим доступа: https://docviewer.yandex.ru/?url=http%3A%2F%2Fdownload.microsoft.com%2Fdocuments%2Frus%2Fmsdn%2Fmsfa2009_w.pdf&name=msfa2009_w.pdf&lang=ru&c=573c9a1408d4 – Дата обращения: 17.02.2016.
13. Создание и запуск модульных тестов для управляемого кода. Microsoft Developer Network [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms182532.aspx – Дата обращения: 01.05.2016.
14. Тестирование программного обеспечения. Microsoft Developer Network [Электронный ресурс]. – Режим доступа: https://social.msdn.microsoft.com/Forums/ru-RU/e750a78b-0c1f-4766-81a2-7cea9b4b3ea2/ – Дата обращения:11.02.2016.

ПРИЛОЖЕНИЕ А

BaseClients–>Program.cs

```
using System;
using System.Windows.Forms;

namespace BaseClients
{
 static class Program
 {
   /// <summary>
   /// Главная точка входа для приложения.
   /// </summary>
   [STAThread]
   static void Main()
   {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
   }
 }
}
```

BaseClients–>Form1.cs

```
using System;
using System.Windows.Forms;
using BaseClients.Logic;

namespace BaseClients
{
  public partial class Form1 : Form
  {
   public LogicClients Customers = new LogicClients();
   public string flagAdd = "";
   public Form1()
   {
    InitializeComponent();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
    Customers.Reader();
   }
  private void вкладчикToolStripMenuItem_Click(object sender,
    EventArgs e)
  {
   flagAdd = "Investor";
   Form2 AddForm = new Form2(this);
   AddForm.ShowDialog();
  }
  private void кредиторToolStripMenuItem_Click(object sender,
   EventArgs e)
  {
   flagAdd = "Lender";
   Form2 AddForm = new Form2(this);
   AddForm.ShowDialog();
  }
```

```
private void организацияToolStripMenuItem_Click(object sender,
    EventArgs e)
{
 flagAdd = "Organization";
 Form2 AddForm = new Form2(this);
 AddForm.ShowDialog();
}
private void выводToolStripMenuItem_Click(object sender,
    EventArgs e)
{
 AllToolStripMenuItem.Click += всеToolStripMenuItem_Click;
 InvestorsToolStripMenuItem.Click +=
         вкладчикиToolStripMenuItem_Click;
 LendersToolStripMenuItem.Click +=
    кредиторыToolStripMenuItem_Click;
 OrganizationsToolStripMenuItem.Click +=
    организацииToolStripMenuItem_Click;
}
private void организацииToolStripMenuItem_Click(object sender,
    EventArgs e)
{
 textBoxMain.Clear();
 var _clients = Customers.GetAll();
 foreach (var client in _clients)
 {
  string[] s = client.ToString().Split('|');
  if (s[0].Substring(0,11)=="Организация")
   textBoxMain.Text += s[0] + Environment.NewLine + s[1]
   + Environment.NewLine + s[2] + Environment.NewLine
   + s[3] + Environment.NewLine + Environment.NewLine;
 }
}
private void кредиторыToolStripMenuItem_Click(object sender,
    EventArgs e)
{
 textBoxMain.Clear();
 var _clients = Customers.GetAll();
 foreach (var client in _clients)
 {
  string[] s = client.ToString().Split('|');
  if (s[0].Substring(0, 8)== "Кредитор")
   textBoxMain.Text += s[0] + Environment.NewLine + s[1]
   + Environment.NewLine + s[2] + Environment.NewLine
   + s[3] + Environment.NewLine + s[4]
   + Environment.NewLine + Environment.NewLine;
 }
}
private void вкладчикиToolStripMenuItem_Click(object sender,
    EventArgs e)
{
 textBoxMain.Clear();
 var _clients = Customers.GetAll();
 foreach (var client in _clients)
 {
  string[] s = client.ToString().Split('|');
  if (s[0].Substring(0, 8)== "Вкладчик")
   textBoxMain.Text += s[0] + Environment.NewLine + s[1]
   + Environment.NewLine + s[2] + Environment.NewLine
```

```
      + s[3] + Environment.NewLine + Environment.NewLine;
     }
    }
    private void всеToolStripMenuItem_Click(object sender, EventArgs
      e)
    {
     textBoxMain.Clear();
     var _clients = Customers.GetAll();
     foreach (var client in _clients)
     {
      string[] s = client.ToString().Split('|');
      if (s[0].Substring(0, 8) == "Кредитор")
        textBoxMain.Text += s[0] + Environment.NewLine + s[1]
      + Environment.NewLine + s[2] + Environment.NewLine
      + s[3] + Environment.NewLine + s[4]
      + Environment.NewLine + Environment.NewLine;
      else
       textBoxMain.Text += s[0] + Environment.NewLine + s[1]
       + Environment.NewLine + s[2] + Environment.NewLine
       + s[3] + Environment.NewLine + Environment.NewLine;
     }
    }
    private void поискToolStripMenuItem_Click(object sender,
      EventArgs e)
    {
     textBoxMain.Clear();
     FSearch SearchForm = new FSearch(this);
     SearchForm.ShowDialog();
    }
   }
}
```

BaseClients–>Form2.cs

```
using System;
using System.Windows.Forms;

namespace BaseClients
{
 public partial class Form2 : Form
 {
  private Form1 F1;

  public Form2()
  {
   InitializeComponent();
  }

  public Form2(Form1 f1)
  {
   InitializeComponent();
   F1 = f1;
  }
  private void Form2_Load(object sender, EventArgs e)
  {
   if(F1.flagAdd == "Investor") InvestorLoad();
   if (F1.flagAdd == "Lender") LenderLoad();
   if (F1.flagAdd == "Organization") OrganizationLoad();
```

```
}

private void InvestorLoad()
{
 labelMain.Text = "Добавление вкладчика";
 textOwer.Visible = false;
 label5.Visible = false;
 label1.Text = "Фамилия";
 label3.Text = "руб. Размер вклада";
 label4.Text = "% Процент по вкладу";
}
private void LenderLoad()
{
 labelMain.Text = "Добавление кредитора";
 label1.Text = "Фамилия";
 label3.Text = "руб. Размер кредита";
 label4.Text = "% Процент по кредиту";
}
private void OrganizationLoad()
{
 labelMain.Text = "Добавление организации";
 textOwer.Visible = false;
 label5.Visible = false;
 label1.Text = "Организация";
 label3.Text = "Номер счета";
 label4.Text = "руб. Сумма на счету";
}

private bool Add_Investor()
{
 if (textName.Text == "" || textName.Text == " " )
 {
  toolTip1.Show("Вы не ввели фамилию", textName,5000);
  return false;
 }
 DateTime D;
 try
 {
  string[] temp = textDate.Text.Split('.');
  D = new DateTime(int.Parse(temp[2]), int.Parse(temp[1]),
    int.Parse(temp[0]));
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textDate, 5000);
  return false;
 }
 try
 {
  int S = int.Parse(textSize.Text);
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textSize, 5000);
  return false;
 }
 try
 {
```

```
  int I = int.Parse(textInter.Text);
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textInter, 5000);
  return false;
 }
 F1.Customers.Add(textName.Text, D, int.Parse(textSize.Text),
           int.Parse(textInter.Text));
 return true;
}
private bool Add_Lender()
{
 if (textName.Text == "" || textName.Text == " ")
 {
  toolTip1.Show("Вы не ввели фамилию", textName, 5000);
  return false;
 }
 DateTime D;
 try
 {
  string[] temp = textDate.Text.Split('.');
  D = new DateTime(int.Parse(temp[2]), int.Parse(temp[1]),
    int.Parse(temp[0]));
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textDate, 5000);
  return false;
 }
 try
 {
  int S = int.Parse(textSize.Text);
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textSize, 5000);
  return false;
 }
 try
 {
  int I = int.Parse(textInter.Text);
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textInter, 5000);
  return false;
 }
 try
 {
  int B = int.Parse(textOwer.Text);
 }
 catch
 {
  toolTip1.Show("Формат данных неверный", textOwer, 5000);
  return false;
 }
```

```
    F1.Customers.Add(textName.Text, D, int.Parse(textSize.Text),
        int.Parse(textInter.Text), int.Parse(textOwer.Text));
    return true;
   }
   private bool Add_Organization()
   {
    if (textName.Text == "" || textName.Text == " ")
    {
     toolTip1.Show("Вы не ввели название", textName, 5000);
     return false;
    }
    DateTime D;
    try
    {
     string[] temp = textDate.Text.Split('.');
     D = new
DateTime(int.Parse(temp[2]),int.Parse(temp[1]),int.Parse(temp[0]));
    }
    catch
    {
     toolTip1.Show("Формат данных неверный", textDate, 5000);
     return false;
    }
    if (textSize.Text == "" || textSize.Text == " " ||
      textSize.Text.Length != 20)
    {
     toolTip1.Show("В номере 20 цифр", textSize, 5000);
     return false;
    }
    try
    {
     int S = int.Parse(textInter.Text);
    }
    catch
    {
     toolTip1.Show("Формат данных неверный", textInter, 5000);
     return false;
    }
    F1.Customers.Add(textName.Text, D, textSize.Text,
      int.Parse(textInter.Text));
    return true;
   }

   private void buttonCancel_Click(object sender, EventArgs e)
   {
    textName.Text = "";
    textDate.Text = "";
    textSize.Text = "";
    textInter.Text = "";
    textOwer.Text = "";
   }

   private void buttonAdd_Click(object sender, EventArgs e)
   {
    bool flag=false;
    switch (F1.flagAdd)
    {
     case "Investor":
```

```
    flag = Add_Investor();
    break;
   case "Lender":
    flag = Add_Lender();
    break;
   case "Organization":
    flag = Add_Organization();
    break;
  }
  if (flag)
  {
   MessageBox.Show("Клиент добавлен");
   buttonCancel_Click(sender, e);
  }
 }
 private void Form2_FormClosing(object sender,
    FormClosingEventArgs e)
 {
  F1.Customers.SaveData();
 }
 private void textName_KeyDown(object sender, KeyEventArgs e)
 {
  switch (e.KeyCode)
  {
   case Keys.Enter:
    textDate.Focus();
    break;
  }
 }
 private void textDate_KeyDown(object sender, KeyEventArgs e)
 {
  switch (e.KeyCode)
  {
   case Keys.Enter:
    textSize.Focus();
    break;
  }
 }
 private void textSize_KeyDown(object sender, KeyEventArgs e)
 {
  switch (e.KeyCode)
  {
   case Keys.Enter:
    textInter.Focus();
    break;
  }
 }
 private void textInter_KeyDown(object sender, KeyEventArgs e)
 {
  switch (e.KeyCode)
  {
   case Keys.Enter:
    if (textOwer.Visible)
      textOwer.Focus();
    else
      buttonAdd_Click(sender, e);
    break;
  }
```

```
  }
  private void textOwer_KeyDown(object sender, KeyEventArgs e)
  {
   switch (e.KeyCode)
   {
    case Keys.Enter:
     buttonAdd_Click(sender,e);
     break;
   }
  }
 }
}
```

BaseClients–>FSearch.cs

```
using System;
using System.Windows.Forms;

namespace BaseClients
{
 public partial class FSearch : Form
 {
  private Form1 F1;
  public FSearch()
  {
   InitializeComponent();
  }

  public FSearch(Form1 f1)
  {
   InitializeComponent();
   F1 = f1;
  }
  private void FSearch_Load(object sender, EventArgs e)
  {
   this.Left = F1.Left - this.Width;
   this.Top = F1.Top;
  }
  //обработать не правильную строку даты
  private void button1_Click(object sender, EventArgs e)
  {
   if(textBox1.Text != "")
   {
    int k = 0;
    var _clients = F1.Customers.GetAll();
    foreach (var client in _clients)
    {
     if (client.Conform(textBox1.Text))
     {
     string[] s = client.ToString().Split('|');
     if (s[0].Substring(0, 8) == "Кредитор")
      F1.textBoxMain.Text += s[0] + Environment.NewLine + s[1]
      + Environment.NewLine + s[2] + Environment.NewLine
      + s[3] + Environment.NewLine + s[4]
      + Environment.NewLine + Environment.NewLine;
     else
      F1.textBoxMain.Text += s[0] + Environment.NewLine + s[1]
      + Environment.NewLine + s[2] + Environment.NewLine
      + s[3] + Environment.NewLine + Environment.NewLine;
```

```
      k++;
      }
     }
     if (k == 0)
      F1.textBoxMain.Text += ("Клиентов с такой датой нет.");
    }

   }
   private void textBox1_KeyDown(object sender, KeyEventArgs e)
   {
    switch (e.KeyCode)
    {
     case Keys.Enter:
      button1_Click(sender, e);
      break;
    }
   }
   private void textBox1_TextChanged(object sender, EventArgs e)
   {
    F1.textBoxMain.Clear();
   }
  }
}
```

ПРИЛОЖЕНИЕ Б

BaseClients.Logic–> LogicClients.cs

```
using System;
using System.Collections.Generic;
using BaseClients.General;
using BaseClients.Data;

namespace BaseClients.Logic
{
    public class LogicClients
    {
     private DataClients _arr = new DataClients();

     public void Add(string line)
     {
          _arr.Add(line);
     }
     public void Add(string s, DateTime d, int size, int inter)
     {
          _arr.Add(s, d, size, inter);
     }
     public void Add(string s, DateTime d, int size, int inter, int
          balance)
     {
          _arr.Add(s, d, size, inter, balance);
     }
     public void Add(string s, DateTime d, string num, int sum)
     {
          _arr.Add(s, d, num, sum);
     }

     public IEnumerable<Client> GetAll()
     {
          return _arr.GetAll();
     }

     public void Reader()
     {
          _arr.Reader();
     }
     public void SaveData()
     {
          _arr.SaveData();
     }
}
}
```

ПРИЛОЖЕНИЕ В

BaseClients.Data–> DataClients.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BaseClients.General;
using System.IO;

namespace BaseClients.Data
{
 public class DataClients
 {
  private List<Client> _arr = new List<Client>();
  public void Add(string line)
  {
   string[] mas = line.Split(' ');
   switch (mas[0])
   {
    case "Вкладчик":
     Add(mas[1], DateTime.Parse(mas[2]), int.Parse(mas[3]),
     int.Parse(mas[4]));
     break;
    case "Кредитор":
     Add(mas[1], DateTime.Parse(mas[2]), int.Parse(mas[3]),
     int.Parse(mas[4]), int.Parse(mas[5]));
     break;
    case "Организация":
     Add(mas[1], DateTime.Parse(mas[2]), mas[3],
     int.Parse(mas[4]));
     break;
   }
  _arr.Sort();
 }
 public void Add(string s, DateTime d, int size, int inter)
 {
  _arr.Add(new Investor(s, d, size, inter));
  _arr.Sort();
 }
 public void Add(string s, DateTime d, int size, int inter, int
     balance)
 {
  _arr.Add(new Lender(s, d, size, inter, balance));
  _arr.Sort();
 }
 public void Add(string s, DateTime d, string num, int sum)
 {
  _arr.Add(new Organization(s, d, num, sum));
  _arr.Sort();
 }
 public IEnumerable<Client> GetAll()
 {
  return _arr.ToList();
 }
 public void Reader()
 {
```

```
  using (StreamReader fileIn = new StreamReader
      (@"../../database.txt", Encoding.GetEncoding(1251)))
  {//считали базу данных
   string line;
   while ((line = fileIn.ReadLine()) != null)
   {
    Add(line);
   }
   fileIn.Close();
  }
 }
 public void SaveData()
 {
  using (StreamWriter fileOut = new StreamWriter
      (@"../../database.txt", false, Encoding.GetEncoding(1251)))
  {
   foreach (var item in _arr)
   {
    fileOut.WriteLine(item.Write());
   }
   fileOut.Close();
  }
 }
}
}
```

ПРИЛОЖЕНИЕ Г

BaseClients.General–> Client.cs

```
using System;

namespace BaseClients.General
{
      abstract public class Client : IComparable
      {
            public int CompareTo(object arg)
            {
                  Client a = (Client)arg;
                  return this.DataOpen.CompareTo(a.DataOpen);
            }
            protected DateTime DataOpen = new DateTime();
            abstract public string Show();
            abstract public bool Conform(string arg);
            abstract public string Write();

      }
}
```

BaseClients.General–> Investor.cs

```
using System;
using System.Text;

namespace BaseClients.General
{
 public class Investor : Client//вкладчик
 {
  private string suname;
  private int sizeDeposit;
  private int interestDeposit;
  public Investor()
  {
   suname = "Фамилия";
   sizeDeposit = 0;
   interestDeposit = 0;
  }
  public Investor(string s, DateTime d, int size, int inter)
  {
   Suname = s;
   DataOpen = d;
   SizeDeposit = size;
   InterestDeposit = inter;
  }
  //Свойства
  public string Suname
  {
   get
   {
    return suname;
   }
   set
   {
    if (value != null && value != " ")
     {
```

```
      suname = value;
    }
  }
}
public int SizeDeposit
{
 get
 {
  return sizeDeposit;
 }
 set
 {
  if (value > 0)
    {
      sizeDeposit = value;
    }
 }
}
public int InterestDeposit
{
 get
 {
  return interestDeposit;
 }
 set
 {
  if (value > 0)
    {
      interestDeposit = value;
    }
 }
}
//Методы
public override string ToString()
{
 string[] S = new string[4];
 StringBuilder s = new StringBuilder();
 s.AppendFormat("Вкладчик: {0}", Suname);
 S[0] = s.ToString();
 s.Clear();
 s.AppendFormat("Дата открытия вклада: ({0}.{1}.{2})",
    DataOpen.Day, DataOpen.Month, DataOpen.Year);
 S[1] = s.ToString();
 s.Clear();
 s.AppendFormat("Размер вклада: {0} руб.", SizeDeposit);
 S[2] = s.ToString();
 s.Clear();
 s.AppendFormat("Проценты по вкладу: {0}%", InterestDeposit);
 S[3] = s.ToString();
 return string.Join("|", S);
}
public override bool Conform(string a)
{
 DateTime arg = DateTime.Parse(a);
 if (DataOpen.Day == arg.Day && DataOpen.Month == arg.Month &&
    DataOpen.Year == arg.Year)
   return true;
 else return false;
```

```
  }
  public override string Write()
  {
   return "Вкладчик "+ Suname + " " + DataOpen.Day + "." +
     DataOpen.Month
     + "." + DataOpen.Year + " " + SizeDeposit + " " +
     InterestDeposit;
  }
 }
}
```

BaseClients.General–> Lender.cs

```
using System;
using System.Text;

namespace BaseClients.General
{
      public class Lender : Client//кредитор
      {
            private string suname;
            //protected int[] dataOpen;
            private int sizeCredit;
            private int interestCredit;
            private int balanceOwed;
            public Lender()
            {
                  suname = "Фамилия";
                  sizeCredit = 0;
                  interestCredit = 0;
                  balanceOwed = 0;
            }
            public Lender(string s, DateTime d, int size, int inter,
                  int balance)
            {
                  Suname = s;
                  DataOpen = d;
                  SizeCredit = size;
                  InterestCredit = inter;
                  BalanceOwed = balance;
            }
            //Свойства
            public string Suname
            {
             get
             {
              return suname;
             }
             set
             {
              if (value != null && value != " ")
              {
               suname = value;
              }
              else Console.WriteLine("Формат данных неверный.
                 Попробуйте снова.");
             }
            }
            public int SizeCredit
```

```
{
      get
      {
            return sizeCredit;
      }
      set
      {
            if (value > 0)
            {
                  sizeCredit = value;
            }
      }
}
public int InterestCredit
{
      get
      {
            return interestCredit;
      }
      set
      {
            if (value > 0)
            {
                  interestCredit = value;
            }
      }
}
public int BalanceOwed
{
      get
      {
            return balanceOwed;
      }
      set
      {
            if (value > 0 && value < SizeCredit)
            {
                  balanceOwed = value;
            }
      }
}
//Методы
public override string ToString()
{
string[] S = new string[5];
StringBuilder s = new StringBuilder();
s.AppendFormat("Кредитор: {0}", Suname);
S[0] = s.ToString();
s.Clear();
s.AppendFormat("Дата выдачи кредита: ({0}.{1}.{2})",
DataOpen.Day, DataOpen.Month, DataOpen.Year);
S[1] = s.ToString();
s.Clear();
s.AppendFormat("Размер кредита: {0} руб.", SizeCredit);
S[2] = s.ToString();
s.Clear();
s.AppendFormat("Проценты по кредиту: {0}%",
InterestCredit);
```

```
            S[3] = s.ToString();
            s.Clear();
            s.AppendFormat("Остаток долга: {0} руб.", BalanceOwed);
            S[4] = s.ToString();
            return string.Join("|", S);
            }
            public override bool Conform(string a)
            {
            DateTime arg = DateTime.Parse(a);
            if (DataOpen.Day == arg.Day && DataOpen.Month ==
            arg.Month && DataOpen.Year == arg.Year)
            return true;
            else return false;
            }
            public override string Write()
            {
            return "Кредитор " + Suname + " " + DataOpen.Day + "." +
                  DataOpen.Month + "." + DataOpen.Year + " " +
                  SizeCredit + " " + InterestCredit
                        + " " + BalanceOwed;
            }

      }
}
```

BaseClients.General–> Organization.cs

```
using System;
using System.Text;

namespace BaseClients.General
{
      public class Organization : Client
      {
            private string title;
            //protected int[] dataOpen;
            private string numberAccount;
            private int summAccount;
            public Organization()
            {
                  title = "Название";
                  numberAccount = "0";
                  summAccount = 0;
            }
            public Organization(string s, DateTime d, string num,
            int sum)
            {
                  Title = s;
                  DataOpen = d;
                  NumberAccount = num;
                  SummAccount = sum;
            }
            //Свойства
            public string Title
            {
                  get
                  {
                        return title;
                  }
```

```
      set
      {
            if (value != null && value != " ")
            {
                  title = value;
            }
      }
}
public string NumberAccount
{
      get
      {
            return numberAccount;
      }
      set
      {
            if (value != null && value != " " &&
            value.Length == 20)
            {
                  numberAccount = value;
            }
      }
}
public int SummAccount
{
      get
      {
            return summAccount;
      }
      set
      {
            if (value > 0)
            {
                  summAccount = value;
            }
      }
}
//Методы
public override string ToString()
{
string[] S = new string[4];
StringBuilder s = new StringBuilder();
s.AppendFormat("Организация: {0}", Title);
S[0] = s.ToString();
s.Clear();
s.AppendFormat("Дата открытия счета: ({0}.{1}.{2})",
DataOpen.Day, DataOpen.Month, DataOpen.Year);
S[1] = s.ToString();
s.Clear();
s.AppendFormat("Номер счета: {0}", NumberAccount);
S[2] = s.ToString();
s.Clear();
s.AppendFormat("Сумма на счету: {0} руб.", SummAccount);
S[3] = s.ToString();
return string.Join("|", S);
}
public override bool Conform(string a)
{
```

```
            DateTime arg = DateTime.Parse(a);
            if (DataOpen.Day == arg.Day && DataOpen.Month ==
               arg.Month && DataOpen.Year == arg.Year)
               return true;
            else return false;
          }
          public override string Write()
          {
            return "Организация " + Title + " " + DataOpen.Day +
               "." + DataOpen.Month + "." + DataOpen.Year + " " +
               NumberAccount + " " + SummAccount;
          }
      }
}
```

ПРИЛОЖЕНИЕ Д

UnitTestProject1–> DataClientsTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;

namespace BaseClients.Data.Tests
{
      [TestClass()]
      public class DataClientsTests
      {
            [TestMethod()]
            public void ReaderTest()
            {
                  DataClients A = new DataClients();
                  A.Reader();
            }

            [TestMethod()]
            public void AddTest()
            {
                  DataClients A = new DataClients();
                  string line = "Вкладчик Лукашова 30.9.2014 450 11";
                  A.Add(line);
            }
            [TestMethod()]
            public void SaveDataTest()
            {
                  DataClients A = new DataClients();
                  A.Reader();
                  A.SaveData();
            }

            [TestMethod()]
            public void AddTestInv()
            {
                  DataClients A = new DataClients();
                  string line = "Вкладчик Лукашова 30.9.2014 450 11";
                  string[] mas = line.Split(' ');
                  A.Add(mas[1], DateTime.Parse(mas[2]),
                  int.Parse(mas[3]), int.Parse(mas[4]));
            }

            [TestMethod()]
            public void AddTestLend()
            {
                  DataClients A = new DataClients();
                  string line = "Кредитор Олегов 21.2.2012 53422 12
                  242";
                  string[] mas = line.Split(' ');
                  A.Add(mas[1], DateTime.Parse(mas[2]),
                  int.Parse(mas[3]), int.Parse(mas[4]),
                  int.Parse(mas[5]));
            }

            [TestMethod()]
            public void AddTestOrgan()
```

```
            {
                DataClients A = new DataClients();
                string line = "Организация СГУ 10.12.2000
                    12345678901234567890 130000";
                string[] mas = line.Split(' ');
                A.Add(mas[1], DateTime.Parse(mas[2]), mas[3],
                    int.Parse(mas[4]));
            }

            [TestMethod()]
            public void GetAllTest()
            {
                DataClients A = new DataClients();
                A.Reader();
                A.GetAll();
            }
        }
    }
```

UnitTestProject1–> InvestorTests.cs

```
    using Microsoft.VisualStudio.TestTools.UnitTesting;
    using System.IO;
    using System;
    using System.Collections.Generic;
    using System.Text;

    namespace BaseClients.General.Tests
    {
        [TestClass()]
        public class InvestorTests
        {
            List<Investor> Reader()
            {
                List<Investor> _arr = new List<Investor>();
                using (StreamReader fileIn = new StreamReader
                    (@"../../testInvesrot.txt",
                    Encoding.GetEncoding(1251)))
                {//считали базу данных
                    string line;
                    while ((line = fileIn.ReadLine()) != null)
                    {
                        string[] mas = line.Split(' ');
                        Investor A = new Investor(mas[1],
                        DateTime.Parse(mas[2]),
                        int.Parse(mas[3]), int.Parse(mas[4]));
                        _arr.Add(A);
                    }
                    fileIn.Close();
                }
                return _arr;
            } //заглушка
            [TestMethod()]
            public void ToStringTest_Investor()
            {
                List<Investor> Aa=Reader();
                for (int i = 0; i < Aa.Count; i++)
                {
```

```
                        var expected = String.Format("Вкладчик:
                        {0}|Дата открытия вклада:
                        ({1}.{2}.{3})|Размер вклада: {4} руб.|Проценты
                        по вкладу: {5}%", Aa[i].Suname,
                        Aa[i].DataOpen.Day, Aa[i].DataOpen.Month,
                        Aa[i].DataOpen.Year, Aa[i].SizeDeposit,
                                        Aa[i].InterestDeposit);
                        Assert.AreEqual(expected, Aa[i].ToString());

                  }

            }

            [TestMethod()]
            public void ConformTest_Investor()
            {
                  List<Investor> Aa = Reader();
                  for (int i = 0; i < Aa.Count - 1; i++)
                  {
                        string S=String.Format("{0}.{1}.{2}",
                              Aa[i + 1].DataOpen.Day,
                               Aa[i + 1].DataOpen.Month,
                              Aa[i + 1].DataOpen.Year);
                        Assert.IsFalse(Aa[i].Conform(S));
                  }
            }

            [TestMethod()]
            public void WriteTest_Investor()
            {
             List<Investor> Aa = Reader();
             for (int i = 0; i < Aa.Count; i++)
             {
               var expected = "Вкладчик " + Aa[i].Suname + " " +
                  Aa[i].DataOpen.Day + "." + Aa[i].DataOpen.Month +
                  "." + Aa[i].DataOpen.Year + " " +
                  Aa[i].SizeDeposit + " " + Aa[i].InterestDeposit;
               Assert.AreEqual(expected, Aa[i].Write());
             }
           }
      }
}
```

UnitTestProject1–> LenderTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace BaseClients.General.Tests
{
      [TestClass()]
      public class LenderTests
      {
            List<Lender> Reader()
            {
              List<Lender> _arr = new List<Lender>();
```

```
  using (StreamReader fileIn = new StreamReader
      (@"../../testLender.txt",
      Encoding.GetEncoding(1251)))
  {//считали базу данных
   string line;
   while ((line = fileIn.ReadLine()) != null)
   {
    string[] mas = line.Split(' ');
    Lender A = new Lender(mas[1],
     DateTime.Parse(mas[2]),int.Parse(mas[3]),
     int.Parse(mas[4]), int.Parse(mas[5]));
     _arr.Add(A);
    }
    fileIn.Close();
  }
 return _arr;
}
[TestMethod()]
public void ToStringTest_Lender()
{
     List<Lender> Aa = Reader();
     for (int i = 0; i < Aa.Count; i++)
     {
          var expected = String.Format("Кредитор:
          {0}|Дата выдачи кредита: ({1}.{2}.{3})|Размер
          кредита: {4} руб.|Проценты по кредиту:
     {5}%|Остаток долга: {6} руб.", Aa[i].Suname,
          Aa[i].DataOpen.Day, Aa[i].DataOpen.Month,
          Aa[i].DataOpen.Year, Aa[i].SizeCredit,
          Aa[i].InterestCredit, Aa[i].BalanceOwed);
          Assert.AreEqual(expected, Aa[i].ToString());
     }

}

[TestMethod()]
public void ConformTest_Lender()
{
     List<Lender> Aa = Reader();
     for (int i = 0; i < Aa.Count - 1; i++)
     {
          string S = String.Format("{0}.{1}.{2}",
           Aa[i + 1].DataOpen.Day,
          Aa[i + 1].DataOpen.Month,
          Aa[i + 1].DataOpen.Year);
          Assert.IsFalse(Aa[i].Conform(S));
     }
}

[TestMethod()]
public void WriteTest_Lender()
{
     List<Lender> Aa = Reader();
     for (int i = 0; i < Aa.Count; i++)
     {
          var expected = "Кредитор " + Aa[i].Suname + "
          " +  Aa[i].DataOpen.Day + "." +
          Aa[i].DataOpen.Month + "." +
```

```
                        Aa[i].DataOpen.Year + " " + Aa[i].SizeCredit +
                        " " + Aa[i].InterestCredit      + " " +
                        Aa[i].BalanceOwed;
                        Assert.AreEqual(expected, Aa[i].Write());
                    }
                }
            }
        }
```

UnitTestProject1–> OrganizationTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace BaseClients.General.Tests
{
    [TestClass()]
    public class OrganizationTests
    {
        List<Organization> Reader()
        {
            List<Organization> _arr = new List<Organization>();
            using (StreamReader fileIn = new StreamReader
                (@"../../testOrgan.txt",
                Encoding.GetEncoding(1251)))
            {//считали базу данных
                string line;
                while ((line = fileIn.ReadLine()) != null)
                {
                string[] mas = line.Split(' ');
                Organization A = new Organization(mas[1],
                DateTime.Parse(mas[2]), mas[3],
                    int.Parse(mas[4]));
                _arr.Add(A);
                }
                fileIn.Close();
            }
            return _arr;
        }
        [TestMethod()]
        public void ToStringTest_Organization()
        {
            List<Organization> Aa = Reader();
            for (int i = 0; i < Aa.Count; i++)
            {
                var expected = String.Format("Организация:
                {0}|Дата открытия счета: ({1}.{2}.{3})|Номер
                счета: {4}|Сумма на счету: {5} руб.",
                Aa[i].Title, Aa[i].DataOpen.Day,
                Aa[i].DataOpen.Month, Aa[i].DataOpen.Year,
                Aa[i].NumberAccount,  Aa[i].SummAccount);
                Assert.AreEqual(expected, Aa[i].ToString());
            }

        }
```

```
        [TestMethod()]
        public void ConformTest_Organization()
        {
            List<Organization> Aa = Reader();
            for (int i = 0; i < Aa.Count - 1; i++)
            {
                string S = String.Format("{0}.{1}.{2}",
                Aa[i + 1].DataOpen.Day,
                Aa[i + 1].DataOpen.Month,
                Aa[i + 1].DataOpen.Year);
                Assert.IsFalse(Aa[i].Conform(S));
            }
        }

        [TestMethod()]
        public void WriteTest_Organization()
        {
            List<Organization> Aa = Reader();
            for (int i = 0; i < Aa.Count; i++)
            {
             var expected = "Организация " + Aa[i].Title + " "
            + Aa[i].DataOpen.Day  + "." + Aa[i].DataOpen.Month +
            "." + Aa[i].DataOpen.Year + " " +
            Aa[i].NumberAccount + " " + Aa[i].SummAccount;
                Assert.AreEqual(expected, Aa[i].Write());
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Е

Функциональное тестирование главной формы.

Текущее положение курсора	Нажатие
Добавить-Вкладчик	Открытие формы "**Добавить вкладчика**"
Добавить-Кредитор	Открытие формы "**Добавить кредитора**"
Добавить-Организация	Открытие формы "**Добавить организацию**"
Вывод-Вкладчики	Вывод в textbox всех вкладчиков в порядке возрастания даты открыти счета
Вывод-Кредиторы	Вывод в textbox всех кредиторов в порядке возрастания даты открыти счета
Вывод-Организации	Вывод в textbox всех организаций в порядке возрастания даты открыти счета
Вывод-Все	Вывод в textbox всех клиентов в порядке возрастания даты открыти счета
Поиск	Открытие формы поиска

Таблица 1 Ожидаемые результаты

Текущее положение курсора	Нажатие
Добавить-Вкладчик	√
Добавить-Кредитор	√
Добавить-Организация	√
Вывод-Вкладчики	√
Вывод-Кредиторы	√
Вывод-Организации	√
Вывод-Все	√
Поиск	√

Таблица 2 Результаты тестов

	Любая клавиша клавиатуры
поле TextBox	Без изменений

	Любая клавиша клавиатуры
поле TextBox	X

Таблица 4 Форма поиска. Ожидаемые результаты

Текущая позиция курсора	Текущее входное состояние	Результат поиска	Числовые клавиши	Неверные и буквенные символы	<ENTER>	<TAB>	<BACKSPACE>	<DELETE>	Управляющие клавиши	Left arrow
поле Даты	**пустое поле**		Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на кнопку **Поиск**	Без изменений	Без изменений	Без изменений	Без изменений
	неверные значения				ошибка неверное значение		Удаление одного символа слева	Удаление одного символа справа		Переход на один символ влево
	верные значения	**Найдены клиента**			обнуление textbox и вывод найденного					
		Не найдены клиенты			обнуление textbox и вывод сообщения, что не найден					
	дважды введена одна и таже дата	**Найдены клиента**			обнуление textbox и вывод найденного					
		Не найдены клиенты			обнуление textbox и вывод сообщения, что не найден					
	дважды введены неверные значения				обнуление textbox и вывод сообщения, что не найден клиент					

Таблица 5 Форма поиска. Результаты тестов

Текущая позиция курсора	Текущее входное состояние	Результат поиска	Числовые клавиши	Неверные и буквенные символы	<ENTER>	<TAB>	<BACKSPACE>	<DELETE>	Управляющие клавиши	Left arrow
поле Даты	пустое поле		√	X	X	√	√	√	√	√
	неверные значения		√	X	X	√	√	√	√	√
	верные значения	Найдены клиента	√	X	√	√	√	√	√	√
		Не найдены клиенты	√	X	√	√	√	√	√	√
	дважды введена одна и таже дата	Найдены клиента	√	X	X	√	√	√	√	√
		Не найдены клиенты	√	X	X	√	√	√	√	√
	дважды введены неверные значения		√	X	X	√	√	√	√	√

Таблица 6 Форма добавления вкладчика. Ожидаемые результаты

Текущая позиция курсора	Текущее входное состояние	Буквенные клавиши	Числовые клавиши	Неверные символы	<ENTER>	<TAB>	<BACKSPACE>	<DELETE>	Управляющие клавиши	Left arrow
поле **Фамилия**	пустое поле	Добавить символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Дата открытия**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	Добавить символ	Добавить символ	ошибка неверный символ	переход на поле **Дата открытия**	переход на поле **Дата открытия**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Дата открытия**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Размер вклада**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Размер вклада**	переход на поле **Размер вклада**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Размер вклада**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Процент по вкладу**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Процент по вкладу**	переход на поле **Сумма на счету**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Процент по вкладу**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на кнопку **Добавить**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	нажатие кнопки **Добавить**	переход на кнопку **Добавить**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево

Таблица 6 Продолжение

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавишы**	**Числовые клавишы**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавишы**	**Left arrow**
кнопка **Отмена**	поля пустые	Без измене-	Без измене-	Без изменений	Без измене-ний	переход на поле **Фамилия**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Добавить**
	поля заполнены	Без измене-	Без измене-	Без изменений	Стирание данных из полей	переход на поле **Фамилия**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Добавить**
кнопка **Добавить**	поля пустые	Без измене-	Без измене-	Без изменений	ошибка, сообщение на поле **Фамилия**	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**
	верно заполненные поля	Без измене-	Без измене-	Без изменений	Добавление клиента, появление сообщения	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**
	поле **"Фамилия"** имеет неверные значения	Без измене-	Без измене-	Без изменений	ошибка, сообщение на поле **Фамилия**	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**
	поле **"Дата открытия"** имеет неверные значения	Без измене-	Без измене-	Без изменений	ошибка, сообщение на поле **Дата открытия**	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**
	поле **"Размер вклада"** имеет неверные значения	Без измене-	Без измене-	Без изменений	ошибка, сообщение на поле **Размер вклада**	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**
	поле **"Процент по вкладу"** имеет неверные значения	Без измене-	Без измене-	Без изменений	ошибка, сообщение на поле **Процент по вкладу**	Переход на кнопку **Отмена**	Без изменений	Без измене-	Без изменений	Переход на кнопку **Отмена**

Таблица 7 Форма добавления вкладчика. Результаты тестов

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавишы**	**Числовые клавишы**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавишы**	**Left arrow**
поле **Фамилия**	пустое поле	√	X	X	X	√	√	√	√	√
	поле с символами	√	X	X	√	√	√	√	√	√
поле **Дата открытия**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Размер вклада**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Процент по вкладу**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
кнопка **Отмена**	поля пустые	√	√	√	√	√	√	√	√	√
	поля заполнены	√	√	√	√	√	√	√	√	√
кнопка **Добавить**	поля пустые	√	√	√	√	√	√	√	√	√
	верно заполненные поля	√	√	√	√	√	√	√	√	√
	поле **"Фамилия"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Дата открытия"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Размер вклада"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Процент по вкладу"** имеет неверные значения	√	√	√	√	√	√	√	√	√

Таблица 8 Форма добавления кредитора. Ожидаемые результаты

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавишы**	**Числовые клавишы**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавишы**	**Left arrow**
поле **Фамилия**	пустое поле	Добавить символ	Добавить символ	ошибка неверный	ошибка пустое поле	переход на поле **Дата открытия**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	Добавить символ	Добавить символ	ошибка неверный символ	переход на поле **Дата открытия**	переход на поле **Дата открытия**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Дата открытия**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный	ошибка пустое поле	переход на поле **Размер кредита**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Размер кредита**	переход на поле **Размер кредита**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Размер кредита**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Процент по кредиту**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Процент по кредиту**	переход на поле **Сумма на счету**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Процент по кредиту**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный	ошибка пустое поле	переход на поле **Остаток долга**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Остаток долга**	переход на поле **Остаток долга**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Остаток долга**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный	ошибка пустое поле	переход на кнопку **Добавить**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	нажатие кнопки **Добавить**	переход на кнопку **Добавить**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево

Таблица 8 Продолжение

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавишы**	**Числовые клавишы**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавишы**	**Left arrow**
кнопка **Отмена**	поля пустые	Без изменений	Без изменений	Без изменений	Без изменений	переход на поле **Фамилия**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Добавить**
	поля заполнены	Без изменений	Без изменений	Без изменений	Стирание данных из полей	переход на поле **Фамилия**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Добавить**
кнопка **Добавить**	поля пустые	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Фамилия**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	верно заполненные поля	Без изменений	Без изменений	Без изменений	Добавление клиента, появление сообщения	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Фамилия"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Фамилия**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Дата открытия"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Дата открытия**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Размер кредита"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Размер кредита**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Процент по кредиту"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Процент по кредиту**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Остаток долга"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Остаток долга**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**

Таблица 9 Форма добавления кредитора. Результаты тестов

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавишы**	**Числовые клавишы**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавишы**	**Left arrow**
поле **Фамилия**	пустое поле	√	X	X	X	√	√	√	√	√
	поле с символами	√	X	X	√	√	√	√	√	√
поле **Дата открытия**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Размер кредита**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Процент по кредиту**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Остаток долга**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
кнопка **Отмена**	поля пустые	√	√	√	√	√	√	√	√	√
	поля заполнены	√	√	√	√	√	√	√	√	√
кнопка **Добавить**	поля пустые	√	√	√	√	√	√	√	√	√
	верно заполненные поля	√	√	√	√	√	√	√	√	√
	поле **"Фамилия"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Дата открытия"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Размер кредита"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Процент по кредиту"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Остаток долга"** имеет неверные значения	√	√	√	√	√	√	√	√	√

Таблица 10 Форма добавления организации. Ожидаемые результаты

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавиши**	**Числовые клавиши**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавиши**	**Left arrow**
поле **Организация**	пустое поле	Добавить символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Дата открытия**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	Добавить символ	Добавить символ	ошибка неверный символ	переход на поле **Дата открытия**	переход на поле **Дата открытия**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Дата открытия**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Номер счета**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Номер счета**	переход на поле **Номер счета**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Номер счета**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на поле **Сумма на счету**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	переход на поле **Сумма на счету**	переход на поле **Сумма на счету**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево
поле **Сумма на счету**	пустое поле	ошибка неверный символ	Добавить символ	ошибка неверный символ	ошибка пустое поле	переход на кнопку **Добавить**	Без изменений	Без изменений	Без изменений	Без изменений
	поле с символами	ошибка неверный символ	Добавить символ	ошибка неверный символ	нажатие кнопки **Добавить**	переход на кнопку **Добавить**	Удаление одного символа слева	Удаление одного символа справа	Без изменений	Переход на один символ влево

Таблица 10 Продолжение

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавиши**	**Числовые клавиши**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавиши**	**Left arrow**
кнопка **Отмена**	поля пустые	Без изменений	Без изменений	Без изменений	Без изменений	переход на поле **Организация**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Добавить**
	поля заполнены	Без изменений	Без изменений	Без изменений	Стирание данных из полей	переход на поле **Организация**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Добавить**
кнопка **Добавить**	поля пустые	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Организация**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	верно заполненные поля	Без изменений	Без изменений	Без изменений	Добавление клиента, появление сообщения	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Организация"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Организация**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Дата открытия"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Дата открытия**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Номер счета"** имеет неверные значения или длинну !=20	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Номер счета**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**
	поле **"Сумма на счету"** имеет неверные значения	Без изменений	Без изменений	Без изменений	ошибка, сообщение на поле **Сумма на счету**	Переход на кнопку **Отмена**	Без изменений	Без изменений	Без изменений	Переход на кнопку **Отмена**

Таблица 11 Форма добавления организации. Результаты тестов

Текущая позиция курсора	**Текущее входное состояние**	**Буквенные клавиши**	**Числовые клавиши**	**Неверные символы**	**<ENTER>**	**<TAB>**	**<BACKSPACE>**	**<DELETE>**	**Управляющие клавиши**	**Left arrow**
поле **Организация**	пустое поле	√	√	X	X	√	√	√	√	√
	поле с символами	√	√	X	√	√	√	√	√	√
поле **Дата открытия**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Номер счета**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
поле **Сумма на счету**	пустое поле	X	√	X	X	√	√	√	√	√
	поле с символами	X	√	X	√	√	√	√	√	√
кнопка **Отмена**	поля пустые	√	√	√	√	√	√	√	√	√
	поля заполнены	√	√	√	√	√	√	√	√	√
кнопка **Добавить**	поля пустые	√	√	√	√	√	√	√	√	√
	верно заполненные	√	√	√	√	√	√	√	√	√
	поле **"Организация"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Дата открытия"** имеет неверные значения	√	√	√	√	√	√	√	√	√
	поле **"Номер счета"** имеет неверные значения или длинну !=20	√	√	√	√	√	√	√	√	√
	поле **"Сумма на счету"** имеет неверные значения	√	√	√	√	√	√	√	√	√